Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА АРКАДНОЇ ГРИ НА БАЗІ TYPESCRIPT І БІБЛІОТЕКИ PHASER

DEVELOPMENT OF AN ARCADE GAME BASED ON TYPESCRIPT AND THE PHASER LIBRARY

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗ-41
Охріменко Роман Ігорович
_____
(підпис)

Керівник:
к.т.н., доцент
Ящук Андрій Анатолійович
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» 06 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«2» 02 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Охріменку Роману Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Розробка аркадної гри на базі TypeScript і бібліотеки Phaser*

Керівник роботи: *к.т.н., доцент, Ящук Андрій Анатолійович*
затверджені наказом закладу вищої освіти від «*30*» *грудня 2023 р. № 477/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи, функціонально-структурна схема роботи об'єкта проектування, розробка моделі бази даних об'єкта проектування.

5. Перелік графічного матеріалу: *8 рисунків*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ящук А. А.*		
Специфікації вимог до розробленої системи	*Ящук А. А.*		
Розробка об'єкта проєктування	*Ящук А. А.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту		67 %	
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«2»_лютого_2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	

Здобувач вищої освіти _____ (підпис) *Охріменко Р. І.* (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) *Ящук А. А.* (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти: *Охріменко Роман Ігорович*

Тема: *«Розробка аркадної гри на базі TypeScript і бібліотеки Phaser».*

Коротка характеристика кваліфікаційної роботи та прийнятих рішень: *Кваліфікаційна робота бакалавра складається з трьох розділів, в яких проаналізовані проблематика та поставлені завдання за темою роботи, здійснено теоретичне дослідження та практичну реалізацію аркадної гри на базі TypeScript і бібліотеки Phaser, здійснено тестування розробленої системи*

Самостійні розробки і пропозицій автора: *Автор самостійно розробив аркадну гру на базі TypeScript і бібліотеки Phaser.*

Практичне значення роботи *полягає в тому, що розробка може використовуватися в ігровій індустрії і відповідно гарного проведення часу людьми, що грають цю гру..*

Недоліки: *Істотних недоліків в роботі не виявлено.*

Загальний висновок: *У кваліфікаційній роботі бакалавра здійснено аналіз предметної області, вибір методів і засобів для розробки, розроблено програмний продукт, здійснено його тестування. Усі поставлені завдання виконано в повному обсязі.*

Кваліфікаційна робота здобувача освіти Охріменка Романа Ігоровича рекомендується до захисту екзаменаційній комісії та заслуговує позитивної оцінки.

Рецензент кваліфікаційної роботи: Саварин Павло Вікторович, к.п.н., доцент кафедри ЦОТ

(підпис)

«07» червня 2024 р.

Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

ВІДГУК
КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач вищої освіти *Охріменко Роман Ігорович*
(прізвище, ім'я, по батькові)
група *ІПЗ-41*

Тема кваліфікаційної роботи:
Розробка аркадної гри на базі Typescript і бібліотеки Phaser

Керівник *Яцук Андрій Анатолійович*

Актуальність теми
З кожним роком комп'ютерні ігри не втрачають своєї популярності і продовжують розвиватися. Ігри відіграють важливу роль у сучасному суспільстві.

Об'єкт дослідження
Методи і алгоритми розробки комп'ютерних ігор

Характеристика теоретичного рівня, наявності самостійних розробок і практичної значимості роботи:
Автор аналізує аналоги розроблюваної гри, обгрунтовує засоби і методи розробки, здійснює розробку гри використовуючи відповідні технології.. Зміст роботи відповідає обраній темі.

Зауваження та недоліки: *Суттєвих недоліків в роботі не виявлено.*

Загальний висновок:
Кваліфікаційна робота бакалавра виконана на високому рівні. Робота виконана відповідно до вимог, логічно й послідовно описує хід виконання поставленого завдання. Пояснювальна записка відповідає стандартам до її оформлення. Робота може бути оцінена на високу оцінку

Керівник _____ **(*Яцук А.А.*)**
(підпис) (прізвище, ім'я, по батькові)

«07» червня 2024 р.

Міністерство освіти і науки України

Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА АРКАДНОЇ ГРИ НА БАЗІ TYPESCRIPT І БІБЛІОТЕКИ PHASER

DEVELOPMENT OF AN ARCADE GAME BASED ON TYPESCRIPT AND THE PHASER LIBRARY

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗ-41
Охріменко Роман Ігорович

_____
(підпис)

Керівник:
к.т.н., доцент
Ящук Андрій Анатолійович

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна

_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Охріменку Роману Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *Розробка аркадної гри на базі TypeScript і бібліотеки Phaser*

Керівник роботи: *к.т.н., доцент, Ящук Андрій Анатолійович*
затверджені наказом закладу вищої освіти від «*30*» *грудня 2023 р.* № *477/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, аналіз і вибір засобів проектування, опис функціонального наповнення об'єкта проектування, розробка й обґрунтування системного наповнення, оцінка ергономічних та надійністних параметрів проектованої системи, функціонально-структурна схема роботи об'єкта проектування, розробка моделі бази даних об'єкта проектування.

5. Перелік графічного матеріалу: *8 рисунків*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ящук А. А.*		
Специфікації вимог до розробленої системи	*Ящук А. А.*		
Розробка об'єкта проєктування	*Ящук А. А.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	*___%*		
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«2»_лютого_2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	

Здобувач вищої освіти _____ *Охріменко Р. І.*
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ *Ящук А. А.*
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Охріменко Р. І. Розробка аркадної гри на базі TypeScript і бібліотеки Phaser. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків і пропозицій, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблеми розробки комп'ютерних ігор і сформульовано завдання. В другому розділі визначено вимоги до розроблюваної системи, а також засоби, методи і алгоритми розробки. У третьому розділі розроблено аркадну комп'ютерну гру на базі Phaser і здійснено її тестування. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробки демонструють можливості TypeScript і бібліотеки Phaser для створення комп'ютерних ігор.

Ключові слова: комп'ютерна гра, аркада, TypeScript, Phaser.

ANNOTATION

Okhrimenko R. I. Development of an Arcade Game Based on TypeScript and the Phaser Library. Manuscript.

Bachelor's qualifying thesis of the educational program "Software Engineering". Lutsk National Technical University. Lutsk, 2024.

The bachelor's qualifying thesis consists of an introduction, three sections, conclusions and proposals, a list of used sources and annexes.

In the first chapter, an analysis of the current state of the problem of developing computer games was carried out and the task was formulated. The second section defines the requirements for the developed system, as well as means, methods and development algorithms. In the third chapter, an arcade computer game based on Phaser was developed and tested. The conclusions summarize the information presented in the previous parts. The development results demonstrate the possibilities of TypeScript and the Phaser library for creating computer games.

Keywords: computer game, arcade, TypeScript, Phaser.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1 АНАЛІЗ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ 8
1.1 Аналіз сучасного стану проблеми 8
1.2 Постановка завдання на кваліфікаційну роботу бакалавра 10
Висновки до розділу 1 11
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ 12
2.1 Аналіз, визначення вимог до розроблюваної аркадної гри та проектування програмного забезпечення 12
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання 16
Висновки до розділу 2 18
РОЗДІЛ 3 РОЗРОБКА АРКАДНОЇ ГРИ НА БАЗІ PHASER 20
3.1 Практична реалізація аркадної гри з використанням TypeScript і Phaser 20
3.2 Розробка ігрових персонажів 25
3.3 Анімації в грі 33
3.4 Додавання звуків у грі 34
3.5 Реалізація взаємодій у грі 35
3.6 Розробка карт рівня 38
3.7 Тестування та налагодження розробленої гри 41
Висновки до розділу 3 43
ВИСНОВКИ 45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 46
ДОДАТКИ 48

ВСТУП

Актуальність теми. Перша відеогра гра була представлена публіці ще в 1940 році [1]. З кожним роком комп'ютерні ігри не втрачають своєї популярності і продовжують розвиватися. Ігри відіграють важливу роль у сучасному суспільстві. Деякі з них розказують цілі історії, що робить їх унікальними витворами мистецтва. Сьогодні ігри не тільки розважають і відволікають від проблем, а й знайомлять і згуртовують людей, допомагають розвивати логіку і спритність, і навіть дають можливість заробити і стати популярним.

Ігри доступні на різних платформах – ПК, консолях, мобільних пристроях і навіть у веб-браузерах.

Веб-браузерні ігри залишаються популярними завдяки їх доступності та простоті.

Метою роботи є розробка браузерної аркадної гри на базі технологій TypeScript і бібліотеки Phaser.

Реалізація мети вимагає вирішення наступних завдань:

– здійснити аналіз сучасного стану ігрової індустрії та проблеми розробки комп'ютерних ігор;

– визначити вимоги до розроблюваної комп'ютерної гри;

– визначити і обґрунтувати технології, методи і засоби для здійснення розробки зазначеної гри;

– розробити аркадну комп'ютерну гру з використанням обраних технологій;

– здійснити тестування і налагодження розробленої гри.

Об'єктом дослідження є методи і алгоритми розробки комп'ютерних ігор.

Предметом дослідження є аркадна комп'ютерна гра на базі TypeScript і Phaser.

Новизна роботи полягає у власній реалізації аркадної комп'ютерної гри з використанням технологій TypeScript і Phaser.

Практичне значення розробки: розроблювана гра несе в собі розважальну функцію. Розроблювана гра, за рахунок використання веб-технологій, може запускатися в будь-якому сучасному браузері незалежно від операційної системи.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Гра – це структурована діяльність, що моделює інший вид активності з метою розваги. Від інших видів діяльності, таких як робота, вона відрізняється тим, що не має на меті досягнення «корисної» мети для гравця. Втім, ця межа часто буває розмитою, оскільки існує багато ігор, які можна розглядати як роботу [2].

Грати в ігри можна як для задоволення, так і для отримання певної винагороди. Ігри можуть бути індивідуальними або командними, а також включати змагання з іншими гравцями. Ключовими елементами гри є цілі, правила, виклики та взаємодія. Ігри зазвичай включають фізичну або розумову стимуляцію гравця, а часто і те й інше.

Відеогра – це гра, яка являє собою комп'ютерну програму, що використовує мультимедійні можливості комп'ютера. Для керування відеоіграми використовуються ігрові контролери, такі як геймпади, миші, клавіатури, сенсорні екрани та інші.

Ігрова механіка – це правила або обмеження, що керують діями гравця, а також реакція гри на ці дії. Сукупність усіх ігрових механік утворює організацію ігрового процесу та може визначати жанр гри.

Ігровий цикл (або core-gameplay, core-loop) – це зациклений набір основних ігрових механік, які разом формують ігровий досвід. Це ті дії, які гравець виконує протягом усього ігрового процесу. Основний ігровий цикл є центральним елементом гри, навколо якого вона побудована. З часом він може змінюватися та ускладнюватися з додаванням нових ігрових механік або розвитком старих.

Ігровий цикл повинен бути зрозумілим, легко виконуваним, гнучким і різноманітним, мати зв’язок з іншими ігровими циклами, приносити гравцю задоволення [3].

Сучасний стан комп’ютерних ігор відображає значну еволюцію як у графіці та геймплеї, так і у технологіях розробки.

Ігри доступні на різних платформах – персональний комп’ютер, консоль, мобільні пристрої, веб-браузери.

Завдяки потужним графічним процесорам, сучасні ігри можуть мати фотореалістичну графіку і підтримують високі роздільні здатності та частоти кадрів.

На сьогодні поширеними є найрізноманітніші жанри ігор: від класичних RPG та FPS до інноваційних VR-ігор і ігор з доповненою реальністю.

Ігровий жанр – це категорія ігор, які мають спільні характеристики ігрового процесу. Основним критерієм для його опису є дії, що їх виконує гравець, на відміну від літературних жанрів, де важливішим є сюжет і стиль. В іграх подібні характеристики часто називають сеттингом [4].

Чітка класифікація ігрових жанрів досить складна, оскільки одна гра може належати до кількох жанрів одночасно. Проте така класифікація дуже корисна, адже гравці зазвичай віддають перевагу певному типу ігор, і при анонсі конкретного жанру можуть легко визначити, чи буде гра їм цікавою.

Серед популярних жанрів можна виділити екшени (платформери, ігри-поєдинки, стрілялки, аркадні ігри, симулятори, гонки), стратегії (економічні, карткові, військові), рольові ігри (навчальні, пригодницькі, квести, пазли) та інші.

Варто окремо зупинимось на жанрі «аркада». Аркадні ігри є одним з найстаріших жанрів. Основне завдання в таких іграх – збирання спеціальних об’єктів на рівні, іноді потрібно зафарбувати рівень (наприклад, у «три в ряд»), знайти вихід або вчасно натискати клавіші. Часто такі ігри нескінченні, і мета полягає в тому, щоб набрати якомога більше очок. Вид в таких іграх зазвичай

зверху, іноді збоку. При поєднанні з іншими жанрами, в аркадних іграх можуть з’являтися битви з ворогами або пастки, з яких треба вибратися.

Мультіплеєрні ігри і платформи, такі як Steam, Epic Games Store, і GOG, сприяють розвитку геймерських спільнот.

Популярними також є веб-браузерні ігри завдяки їх доступності та простоті.

Наприклад, популярна гра Agar.io [5] – багатокористувацька гра, де гравці керують клітинами і змагаються за виживання. Іншим прикладом є популярна гра 2048 [6] – логічна гра, де гравці об’єднують числа, щоб досягти плитки з числом 2048.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У відповідності до теми кваліфікаційної роботи бакалавра і після проведеного аналізу сучасного стану ігрової індустрії та проблеми розробки комп’ютерних сформовано наступні завдання на кваліфікаційну роботу бакалавра:

- визначення вимог до розроблюваної аркадної комп’ютерної гри;
- визначення і обґрунтування технологій, методів і засобів, які дозволять здійснити розробку аркадної комп’ютерної гри;
- розробити аркадну комп’ютерну гру з застосуванням обраних технологій (TypeScript і Phaser), забезпечити захоплюючий ігровий процес з чіткими і зрозумілими правилами гри, переходом на наступні рівні, взаємодією героя з ворогами, та іншими об’єктами ігрового світу;
- здійснити тестування і налагодження розробленої аркадної гри.

Висновки до розділу 1

Проаналізовано сучасний стан відеоігор. Відеоігри не втрачають своєї популярності. Існує велика кількість жарнів ігор, а також ігрових платформ. Браузерні ігри є популярними завдяки їх доступності і простоті.

В результаті аналізу і у відповідності до теми кваліфікаційної роботи бакалавра було сформульовано завдання, що полягають у розробці браузерної аркадної комп’ютерної гри з використанням технологій TypeScript і Phaser.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваної аркадної гри та проектування програмного забезпечення

Правила розроблюваної гри і ігровий процес будуть схожими до правил гри «Knightmare» (рис. 2.1), створеної в 1986 році [7], зокрема:

– герой-одинак;
– платформа «shoot'em'up»;
– вертикальний фон з автопрокручуванням;
– велика кількість ворогів, що переміщуються зверху екрану до героя;
– наявність босів рівня.

Рисунок 2.1 – Класична версія гри «Knightmare» [8]

Сформулюємо вимоги до розробки аркадної гри на базі TypeScript та Phaser

Загальні вимоги. У відповідності до теми мова програмування, що буде застосовуватися при розробці – TypeScript. Ігровий рушій Phaser. Розроблювана 2D-гра призначена для роботи у веб-браузері.

Вимоги до ігрового процесу. Гравець керує персонажем, який просувається по ігровому рівню. Управління персонажем здійснюється за допомогою клавіш зі стрілками.

Персонаж може атакувати ворогів за допомогою пострілів (натискання клавіш пробіл або інших визначених клавіш).

Вороги. У грі передбачається наявність різноманітних ворогів з різними характеристиками (швидкість, сила атаки, здоров'я). Вороги повинні з'являтися на рівнях у певних місцях або випадковим чином. Вороги атакують гравця при наближенні.

Рівні. В розроблюваній грі повинно бути кілька рівнів з поступово зростаючою складністю. Кожен рівень має унікальний дизайн. Вкінці кожного рівня є бос, перемога над яким відкриває наступний рівень.

Об'єкти на рівні. Персонаж може збирати об'єкти (монети), які збільшують його здоров'я. На кожному рівні повинні бути перешкоди, які перешкоджають переміщенню гравця.

Здоров'я. Персонаж має певну кількість здоров'я, яке зменшується при отриманні ушкоджень.

Ігровий інтерфейс. Інтерфейс повинен відображати поточний рівень здоров'я, поточний рівень.

Технічні вимоги. Проєкт має бути структурованим у відповідності до найкращих практик TypeScript та Phaser: використання модулів для розділення логіки гри, керування станами, обробки подій тощо.

Оптимізація. Гра повинна працювати плавно у сучасних веб-браузерах. Повинні використовуватися методи оптимізації для зменшення навантаження на процесор та оперативну пам'ять.

Аудіо. Ігровий процес повинен супроводжуватися фоновими музичними треками для кожного рівня. Також повинні бути присутні звукові ефекти для дій гравця (атака, збирання предметів, отримання ушкоджень тощо).

Вимоги до тестування. Потрібно здійснити тестування всіх основних функцій гри (управління персонажем, атаки, взаємодія з об'єктами тощо).

Потрібно здійснити перевірку коректності відображення ігрового інтерфейсу.

Також потрібно здійснити перевірку гри на різних пристроях і браузерах для забезпечення стабільної роботи.

Всі виявлені баги повинні бути задокументовані та виправлені до випуску гри.

Розроблювана гра у відповідності до теми кваліфікаційної роботи бакалавра буде створена на основі технологій HTML5, TypeScript і бібліотеки Phaser.

Розробимо діаграму варіантів використання (Use Case Diagram) для опису розроблюваної гри. Діаграми варіантів використання ілюструють функціональні можливості системи і описують очікувану поведінку системи, допомагають розробникам проаналізувати зв’язки між варіантами використання та персонажами [9].

– Для створення UML діаграми варіантів використання можна визначити основні функціональні можливості гри, включаючи взаємодії між гравцем та системою. Діаграма варіантів використання показана на рисунку 2.2.

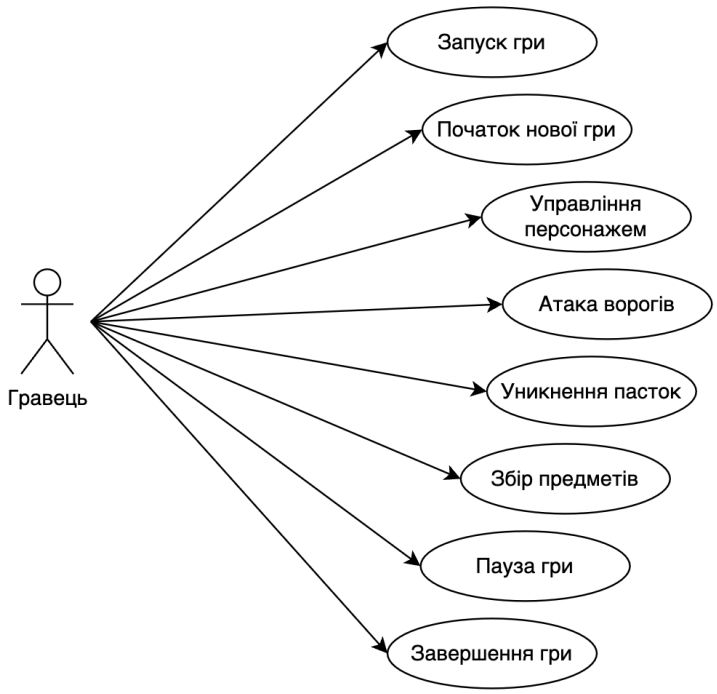

Рисунок 2.2 – Діаграма варіантів використання для розроблюваної гри

Діаграма активності UML – це блок-схема, яка описує всі дії системи. Вона показує все від початку до кінця, визначаючи різні шляхи прийняття рішень і кроки, які мають відбутися, щоб перейти від однієї діяльності до іншої. Кроки можуть бути хронологічними, розгалуженими або одночасними.

Для розроблюваної гри діаграма активності показана на рисунку 2.3.

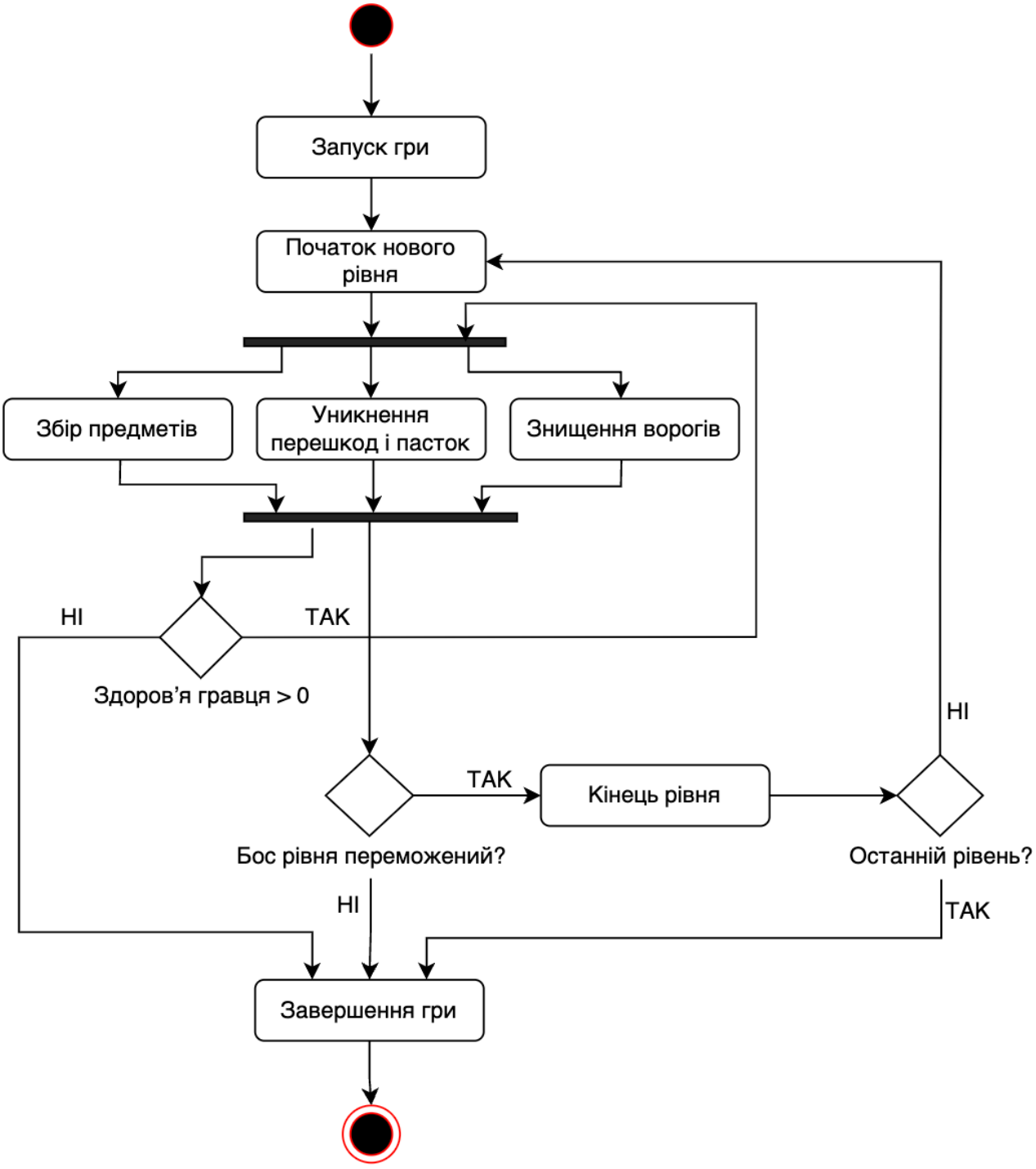

Рисунок 2.3 – Діаграма активності для розроблюваної гри

Місія головного героя полягає в тому, щоб пройти по дорозі, борючись з ворогами і перемагаючи босів кожного рівня. Коли головний герой рухається вперед, він втрачає одиниці здоров'я, тому він повинен економити його, уникаючи надмірних рухів. На його шляху можна знайти монети, які збільшують здоров'я.

Кожен рівень закінчується фінальною ареною, яка охороняється босом рівня. Наш герой повинен перемогти боса і пройти через ворота, які ведуть на наступний рівень.

Меню гри показує тільки заголовок гри і текст, що закликає гравця натиснути пробіл для початку гри.

Коли гравець натискає пробіл, він переходить до наступного екрана гри: екрана презентації рівня, який показує текст з номером рівня.

Ігровий процес відбувається на екрані рівня. Тут здійснюється керування героєм за допомогою клавіш зі стрілками і постріли, щоб знищувати ворогів.

Коли головний герой втрачає енергію, довжина панелі здоров'я в нижній частині екрану зменшується. Кожен раз, коли головний герой стикається з ворогом або потрапляє під постріл, він втрачає певну кількість здоров'я. Але є можливість збирати монети на шляху, і тоді здоров'я збільшується до 100 %.

Гра закінчується, коли панель «Здоров'я» повністю спорожніє. Після цього гравець перенаправляється на екран меню для нового раунду гри.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Як зазначено раніше, розроблювана гра є браузерною і буде розроблена з використанням технологій TypeScript і бібліотеки Phaser.

Браузерні комп'ютерні ігри мають низку особливостей, які накладають деякі обмеження на технології, що застосовуються при розробці звичайних комп'ютерних та мобільних ігор. Одне з таких обмежень – це обмежені ресурси

продуктивності, веб-сторінка в браузері має обмежений доступ до оперативної пам’яті та процесорного часу. Ще одне обмеження – необхідність завантаження всіх або майже всіх файлів гри при кожному запуску. Крім того, різні браузери можуть застосовувати специфічну оптимізацію, що створює додаткові труднощі для ігрового рушія, наприклад, зупинка виконання коду у неактивних вкладках або зниження кількості кадрів в секунду для економії енергії. Код і матеріали гри з браузера зазвичай можна легко завантажити та змінити. Через ці недоліки відомі розробники рідко працюють з браузерними іграми. Проте, браузерні ігри також мають переваги, такі як кросплатформна підтримка та доступність.

Мова JavaScript – це нетипізована скриптова мова програмування, яка відповідає стандарту ECMAScript [10]. Вона може виконуватись на будь-яких пристроях з підтримкою браузерів, що полегшує кросплатформну розробку. JavaScript є об’єктно-орієнтованою мовою з використанням прототипів.

JavaScript є однією з найпопулярніших мов для розробки веб-браузерних ігор завдяки своїй доступності і підтримці в усіх сучасних браузерах. JavaScript підтримує швидку розробку та тестування прямо в браузері. TypeScript є надмножиною JavaScript і додає статичну типізацію, що полегшує розробку великих проектів, допомагає уникнути багатьох помилок під час компіляції та полегшує підтримку коду.

Існує багато бібліотек і фреймворків, які спрощують процес створення ігор.

Візуальна частина гри реалізована на ігровому рушії Phaser [11]. Цей рушій розроблений компанією «Photon Storm» і базується на візуальній бібліотеці Pixi.js. Pixi.js надає зручні можливості роботи з відображенням зображень за допомогою Canvas та WebGL, залежно від можливостей браузера, що робить цей рушій гнучким і продуктивним у відображенні ігрової графіки.

Phaser є популярним фреймворком для створення 2D-ігор на JavaScript. Він надає багато можливостей для розробників, включаючи роботу з графікою, фізикою, звуком та анімацією.

Особливості Phaser:

– підтримка WebGL і Canvas для рендерингу графіки;

– вбудовані фізичні системи (Arcade Physics, P2, Matter.js);

– наявні інструменти для створення складних анімацій;

– велика кількість додаткових модулів, які спрощують розробку специфічних функцій.

Для розробки гри використовувалось інтегроване середовище розробки Visual Studio Community [12].

Visual Studio надає широку підтримку розробки JavaScript, а також з використанням мови програмування TypeScript. Існує можливість писати код JavaScript або TypeScript у Visual Studio для багатьох типів програм і служб [13].

Для створення і редагування графіки використовувалися редактори графіки Adobe Illustrator та Adobe Photoshop [14].

Висновки до розділу 2

Було визначено основні вимоги до розроблюваної аркадної гри та проведено проектування програмного забезпечення. Зокрема, розроблено ігровий процес, схожий на гру «Knightmare».

Було сформульовано загальні та технічні вимоги до гри, включаючи використання TypeScript як основної мови програмування та Phaser як ігрового рушія. Гра повинна працювати у веб-браузері та забезпечувати плавний ігровий процес завдяки оптимізації для зменшення навантаження на процесор та оперативну пам’ять.

Детально описано вимоги до ігрового процесу: управління персонажем, атаки ворогів, наявність різних рівнів з поступово зростаючою складністю та унікальним дизайном, наявність босів рівня, а також інтеграція об’єктів рівня, таких як монети, для збільшення здоров’я персонажа. Важливою частиною було визначення технічних вимог до проекту, структурованість коду відповідно до найкращих практик TypeScript та Phaser, а також вимоги до аудіо та ігрового інтерфейсу.

Розглянуто особливості браузерних ігор та їх обмеження, такі як обмежені ресурси продуктивності та необхідність завантаження файлів гри при кожному запуску. Було відзначено переваги використання JavaScript та TypeScript для розробки веб-браузерних ігор, а також переваги ігрового рушія Phaser для створення 2D-ігор.

Здійснено вибір інструментів та середовищ розробки, зокрема інтегрованого середовища розробки Visual Studio Community, редакторів графіки Adobe Illustrator та Adobe Photoshop для створення та редагування графіки гри.

РОЗДІЛ 3
РОЗРОБКА АРКАДНОЇ ГРИ НА БАЗІ PHASER

3.1 Практична реалізація аркадної гри з використанням TypeScript і Phaser

Розробка гри на базі Phaser починається зі створення нового проєкту «HTML Application built with TypeScript» в середовищі Visual Studio [15].

Після цього до проєкту додаються файли Phaser: phaser.d.ts, phaser.js, phaser.min.js. і рядок «<script src='phaser.js'></script>» у файлі «default.htm».

Шаблон TypeScript вже містить демонстраційний код, але він нам не потрібен, тому видаляємо його і замінюємо кодом, що показаний в лістингу 3.1.

Лістинг 3.1 – Код файлу «app.ts»

```
class ArcadeGame {

    // Конструктор класу CustomGame
    constructor() {
        // Створення нового екземпляру гри Phaser з параметрами: ш
ирина, висота, тип рендерингу, ID контейнера і об'єкт з методами p
reload та create
        this.myGame = new Phaser.Game(800, 600, Phaser.AUTO, 'game
Container', { preload: this.preloadAssets, create: this.initialize
Game });
    }

    // Властивість для зберігання об'єкта гри
    myGame: Phaser.Game;

    // Метод для завантаження ресурсів гри
    preloadAssets() {
        // Завантаження зображення з ключем 'gameLogo' з файлу 'ph
aser2.png'
        this.myGame.load.image('gameLogo', 'phaser2.png');
    }

    // Метод для створення початкового стану гри
    initializeGame() {
        // Додавання спрайту з логотипом на центр екрану
        const logoSprite = this.myGame.add.sprite(this.myGame.worl
d.centerX, this.myGame.world.centerY, 'gameLogo');
        // Встановлення якоря спрайту до центру (0.5, 0.5)
        logoSprite.anchor.setTo(0.5, 0.5);
    }
```

```
}

// Функція, яка виконується при завантаженні сторінки
window.onload = () => {
    // Створення нового екземпляру гри CustomGame
    const myArcadeGame = new ArcadeGame();
};
```

Кінець лістингу 3.1

Після компіляції відкривається браузер, де можна побачити логотип Phaser (рис. 3.1).

Рисунок 3.1 – Логотип Phaser, що відображається в базовому шаблоні

Phaser містить велику кількість вбудованих функцій, корисних для різних ігрових стилів і сценаріїв. Наприклад, у ньому є поняття спрайтів. Спрайти – це двовимірні растрові зображення, що представляють ігрових персонажів, таких як герої та вороги. Крім того, спрайтові файли часто містять кілька кадрів персонажів, які утворюють їхню анімацію. Багато ігрових бібліотек працюють зі спрайтами, але Phaser має вбудований механізм виявлення зіткнень і управління ними, що легко налаштовується та реалізується.

Як і багато інших ігрових фреймворків, Phaser працює на основі циклів оновлення та рендерингу, що реалізуються через методи update() і render(). Ці цикли поділяють два основні аспекти виконання гри: кожен цикл оновлення дає змогу оновлювати стан гри та фонові стани персонажів, такі як позиція та рахунок, обробляючи логічні дані та поведінку. Натомість цикл рендерингу

визначає, яка графіка має бути відображена. Наприклад, якщо у нас є спрайт героя та його тінь, цикл рендерингу визначає порядок їх появи на екрані.

У лістингу 3.2 показано змінений, доповнений і адаптований до вимог нашої гри код класу «ArcadeGame».

Лістинг 3.2 – Клас «ArcadeGame», що викликається подією «window.onload»

```
class ArcadeGame {
    myGame: Phaser.Game;
    statusBar: Phaser.BitmapText;

    constructor() {
        // Ініціалізація гри Phaser з розмірами 512x512 пікселів,
автоматичним вибором рендерера та вказанням елементу HTML з id 'co
ntent'
        this.game = new Phaser.Game(512, 512, Phaser.AUTO, 'conten
t', {
            create: this.create.bind(this),
            preload: this.preload.bind(this)
        });
    }

    preload() {
        // Метод завантаження ресурсів гри
        // Завантаження зображення для статусної панелі
        this.game.load.image('statusBar', 'path/to/statusBar.png')
;
        // Тут можна додати більше ресурсів, які потрібно завантаж
ити перед запуском гри
    }

    create() {
        // Метод створення об'єктів гри
        // Додавання статусної панелі до гри
        this.statusBar = this.game.add.bitmapText(10, 10, 'carrier
_command', 'Health: 100', 16);
        // Тут можна додати більше об'єктів гри, наприклад, гравця
, ворогів, перешкоди тощо
    }

    update() {
        // Метод оновлення стану гри
        // Тут потрібно додати логіку для оновлення стану об'єктів
гри кожного кадру
    }

    render() {
        // Метод рендерингу
```

```
        // Тут потрібно додати логіку для рендерингу специфічних е
лементів гри кожен кадр
    }

    // Тут додаткові методи для гри
}

// Запуск гри після завантаження сторінки
window.onload = () => {
    var game = new ArcadeGame();
};
```

Кінець лістингу 3.2

Гра зазвичай потребує багато ресурсів, таких як зображення, звуки, спрайт-листи, карти плиток, JSON, XML, які автоматично обробляються і зберігаються в глобальному кеші, готові для використання в грі. Phaser дозволяє завантажувати кожен ресурс за допомогою одного рядка коду. Ви можете завантажити ресурси у проекті Phaser, викликаючи метод game.load у функції під назвою preload. При старті програми, Phaser завжди шукає функцію з іменем preload і завантажує визначені в ній ресурси. Додамо код для завантаження ресурсів в метод «preload()» (лістинг 3.3).

Лістинг 3.3 – Метод «ArcadeGame.preload()»

```
preload() {
    // Зображення для заставок та меню
    this.game.load.spritesheet('mainMenu', 'assets/img/menuBackgro
und.png', 512, 384);
    this.game.load.spritesheet('introScreen', 'assets/img/introScr
een.png', 512, 384);

    // JavaScript файли для різних станів гри
    this.game.load.script('initState', 'scripts/gameStates/initSta
te.js');
    this.game.load.script('mainMenu', 'scripts/gameStates/mainMenu
.js');
    this.game.load.script('intro', 'scripts/gameStates/intro.js');
    this.game.load.script('endGame', 'scripts/gameStates/endGame.j
s');
    this.game.load.script('gameLevel', 'scripts/gameStates/gameLev
el.js');
    this.game.load.script('credits', 'scripts/gameStates/credits.j
s');
```

```
    // Класи для гравця, ворогів, босів та інші
    this.game.load.script('hero', 'scripts/entities/hero.js');
    this.game.load.script('heroBullet', 'scripts/entities/heroBull
et.js');
    this.game.load.script('heroState', 'scripts/entities/heroState
.js');
    this.game.load.script('mainBoss', 'scripts/entities/mainBoss.j
s');
    this.game.load.script('enemy', 'scripts/entities/enemy.js');
    this.game.load.script('powerUp', 'scripts/items/powerUp.js');

    // Зображення для вступу до рівнів
    this.game.load.image('level1', 'assets/img/lvl1.jpg');
    this.game.load.image('level2', 'assets/img/lvl2.jpg');
    this.game.load.image('level3', 'assets/img/lvl3.jpg');

    // Спрайт-листи для всіх персонажів у грі
    this.game.load.spritesheet('heroSprite', 'assets/sprites/heroS
prite.png', 32, 32);
    this.game.load.spritesheet('powerUpSprite', 'assets/sprites/po
werUpSprite.png', 32, 32);
    this.game.load.spritesheet('bossSprite', 'assets/sprites/bossS
prite.png', 96, 96);
    this.game.load.spritesheet('minionSprite', 'assets/sprites/min
ionSprite.png', 32, 32);
    this.game.load.spritesheet('heroBulletSprite', 'assets/sprites
/heroBulletSprite.png', 32, 32);

    // Аудіо та музичні ресурси
    this.game.load.audio('startSound', ['assets/audio/start-level.
wav']);
    this.game.load.audio('introSound', ['assets/audio/sound-intro.
wav']);

    // Бітмапні шрифти для гри
    this.game.load.bitmapFont('gameFont', 'assets/fonts/gameFont.p
ng', 'assets/fonts/gameFont.xml');
}
```

Кінець лістингу 3.3

Наведений метод preload() обробляє наступні типи ресурсів: спрайти, зображення, аудіо, бітмапні шрифти.

Варто звернути увагу на те, що всі ці ресурси завантажуються за допомогою рядкового ключа, який пізніше використовується для ідентифікації правильного ресурсу при його необхідності. Завантажуючи ресурси заздалегідь,

ми можемо бути впевнені, що програма не буде страждати від затримок під час гри, оскільки всі ресурси вже доступні в пам’яті.

Phaser підтримує класичні спрайти з фіксованим розміром кадрів, файли JSON з texture packer та Flash CS6 CC, а також XML файли. Для цього проекту використано спрайти з фіксованим розміром кадрів (рис. 3.2).

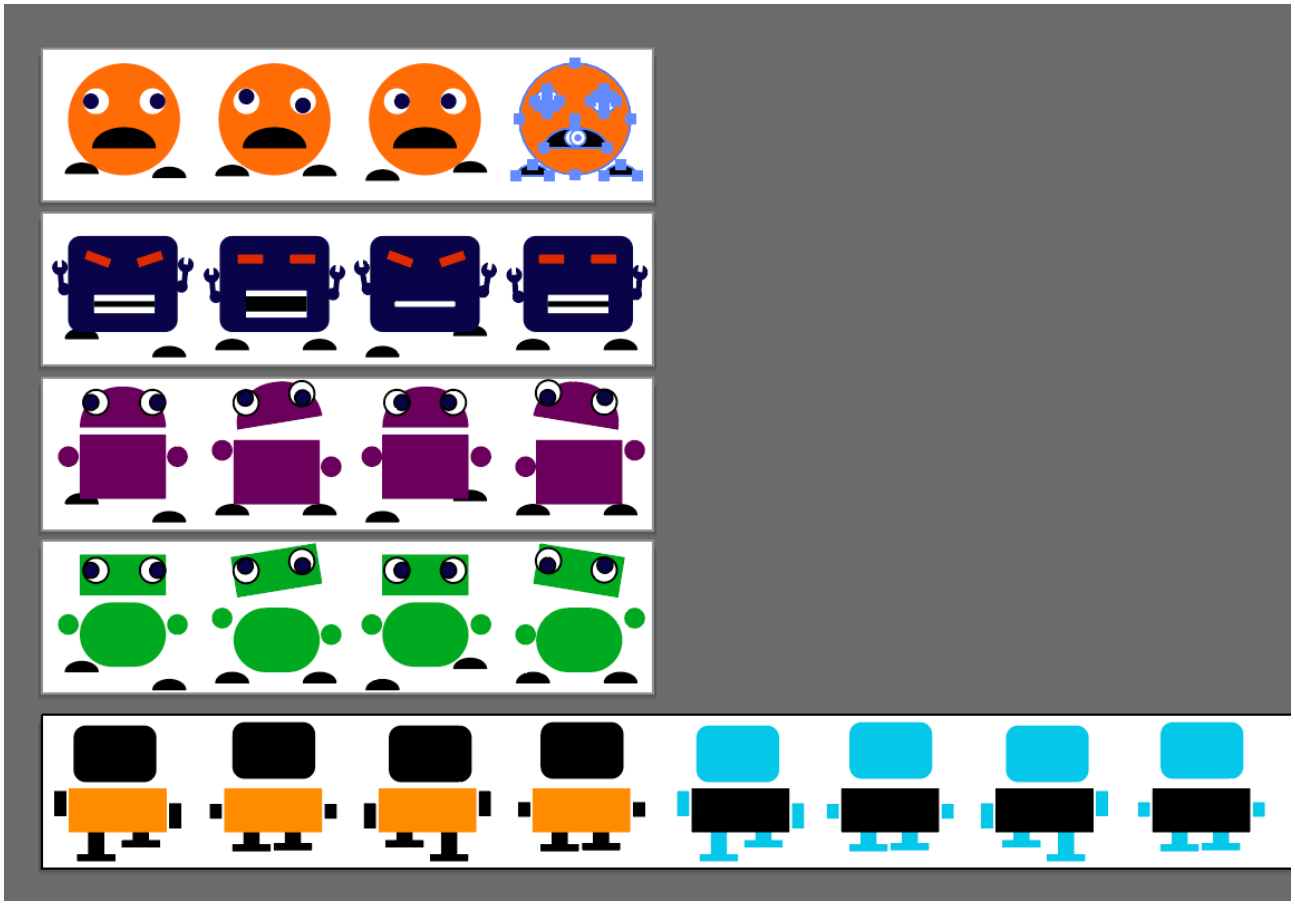

Рисунок 3.2 – Приклад спрайту ігрових персонажів, що застосовується в грі

3.2 Розробка ігрових персонажів

Наш герой представлений класом Player, який реалізує інтерфейс IPlayer. Цей інтерфейс містить поле швидкості, поле спрайту і булеве поле, що вказує, чи заряджена зброя (рис. 3.3).

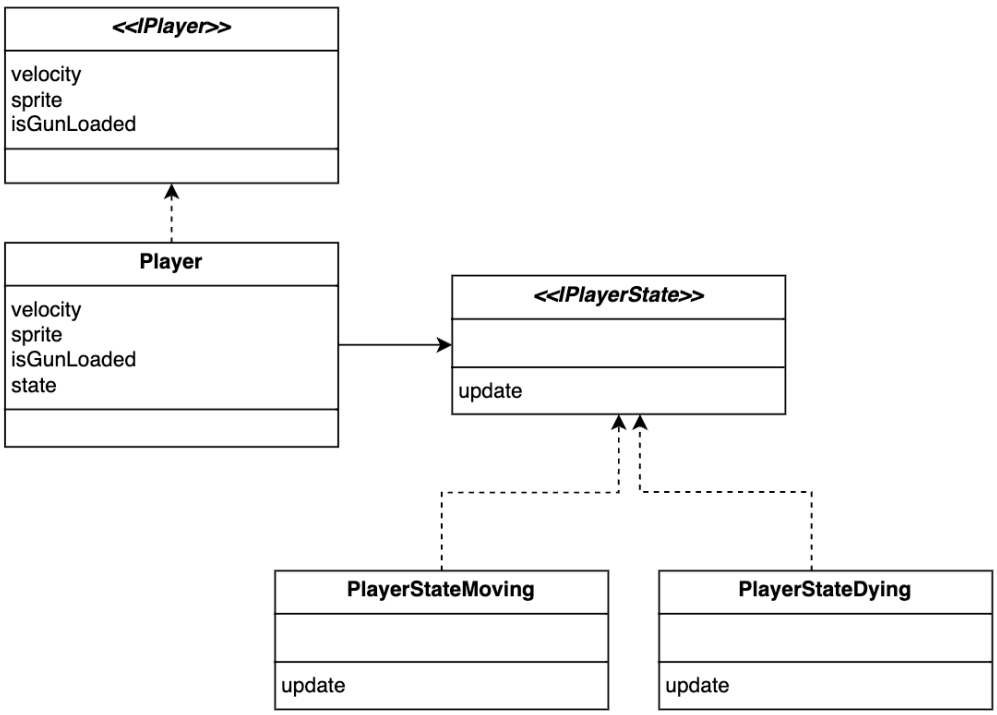

Рисунок 3.3 – Діаграма класів гравця

Швидкість персонажа – це постійна величина, яка визначає, як швидко наш герой переміщується, коли гравець натискає клавіші зі стрілками. Швидкість реалізовано за допомогою методу «moveLeft()» (лістинг 3.4).

Лістинг 3.4 – Метод «Player.moveLeft()»

```
moveLeft() {
    // Рухаємо спрайт вліво
    this.character.body.velocity.x = -this.speed;
}
```

Кінець лістингу 3.4

Спрайт – це об’єкт, який зберігає графічне зображення нашого героя (та ворогів) і визначає, як вони будуть відображатися на екрані. Вміст методу «setup()», в якому створюється спрайт героя, показано у лістингу 3.5.

Лістинг 3.5 – Присвоєння спрайту в методі «Player.setup()»

```
// Створюємо спрайт героя
this.character = this.game.add.sprite(
```

```
        // задаємо параметри центру, висоти
        this.game.world.centerX - 10,
        this.game.world.height - 64, 'hero');
```

Кінець лістингу 3.5

У гравця є можливість здійснювати постріли. Булеве значення змінної «isGunCharged» вказує, чи заряджена зброя. Кожен постріл нашого героя розряджає зброю, що запобігає одночасному випуску багатьох снарядів (лістинг 3.6).

Лістинг 3.6 – Управління зарядом зброї в методі «PlayerState.update()»

```
// Перевіряємо, чи заряджена зброя та натиснуто пробіл
if (this.hero.isGunCharged && keyboard.isDown(Phaser.KeyCode.SPACE
BAR)) {
    // змінюємо значення isGunCharged
    this.hero.isGunCharged = false;
    // викликаємо метод shoot() для пострілу
    this.hero.shoot();
}

// Якщо пробіл не натиснуто
else if (!keyboard.isDown(Phaser.KeyCode.SPACEBAR)) {
    // змінюємо значення isGunCharged на true
    this.hero.isGunCharged = true;
}
```

Кінець лістингу 3.6

Об’єкт «Player» має подію «wasHit()», яка викликається у разі зіткнення ворогів і босів зі спрайтом гравця (лістинг 3.7).

Лістинг 3.7 – Метод «Player.wasHit()»

```
// Викликається при ударі героя
wasHit() {
    // відтворюємо звук удару
    this.hitSound.play();
    // відтворюємо анімацію удару
    this.character.animations.play('hit');
    // зменшити здоров’я
    this.decreaseHealth(2);
}
```

Кінець лістингу 3.7

Класи поведінки, призначені класу «Player» як поля стану, визначають можливі стани: біг та смерть, які реалізуються класами «PlayerStateRunning» і «PlayerStateDying». Конструктор класу «Player» визначає, що зброя заряджена. У методі «setup» класу «Player» ми визначаємо спрайт, анімації, базову швидкість, швидкість переміщення та розміри героя.

Коли герой отримує удар, ми відтворюємо звук і показуємо анімацію пошкодження (лістинг 3.8).

Лістинг 3.8 – Метод «Player.wasHit()»

```
wasHit() {
    // Відтворюємо звук пошкодження
    this.hitSound.play();
    // Запускаємо анімацію пошкодження
    this.character.animations.play('hit');
    // Зменшуємо здоров'я
    this.decreaseHealth(1);
}
```

Кінець лістингу 3.8

Здоров’я героя зменшується відповідно (лістинг 3.9).

Лістинг 3.9 – Метод «Player.decreaseHealth()»

```
decreaseHealth(amount: number): boolean {
    // Перевіряємо, чи залишається здоров'я після удару
    if (this.health - amount > 0) {
        this.health -= amount;
        this.level.updateHealthBar();
        return true;
    } else {
        this.health = 0;
        this.level.updateHealthBar();
        this.state = new PlayerStateDying(this);
        this.deathSound.play();
        this.level.playerStateChanged(this.state);
        return false;
    }
}
```

Кінець лістингу 3.9

При події «відновлення здоров’я» ми відтворюємо відповідний звук і збільшуємо запас здоров’я на відповідній панелі (лістинг 3.10).

Лістинг 3.10 – Методи «Player.recovered()» та «Player.increaseHealth()»

```
recovered(amount: number) {
    // Відтворюємо звук підзарядки
    this.recoverSound.play();
    // Збільшуємо запас здоров'я
    this.increaseHealth(amount);
}

increaseHealth(amount: number): boolean {
    // Перевіряємо, чи не перевищує здоров'я максимум
    if (this.health + amount < 100) {
        this.health += amount;
        this.level.updateHealthBar();
        return true;
    } else {
        this.health = 100;
        this.level.updateHealthBar();
        return false;
    }
}
```

Кінець лістингу 3.10

Коли герой відроджується, ми відновлюємо його початковий стан і встановлюємо стандартну анімацію ходьби. Це гарантує, що гравець продовжить гру.

Метод «walking()» переміщує спрайт гравця вгору і виконується автоматично без втручання гравця. Він виконується лише для компенсації вертикального автопрокручування, яке постійно тягне фон рівня вниз. В іншому випадку спрайт гравця прокручувався б вниз разом з усім іншим і зникав би з екрану (лістинг 3.11).

Лістинг 3.11 – Метод «Player.walking()» компенсує вертикальне автопрокручування, рухаючи гравця вгору

```
walking() {
    // Перевіряємо, чи не натиснута жодна клавіша зі стрілками
    if (this.noCursorKeyDown()) {
```

```
        this.character.body.velocity.y = -this.walkingSpeed;
    }
}
```

Кінець лістингу 3.11

Методи бігу оновлюють напрямок швидкісного вектора відповідно до натиснутих клавіш зі стрілками (лістинг 3.12).

Лістинг 3.12 – Методи переміщення гравця

```
walkingUp() {
    // Рухаємо спрайт вгору
    this.character.body.velocity.y = -this.speed;
}

walkingRight() {
    // Рухаємо спрайт вправо
    this.character.body.velocity.x = this.speed;
}

walkingDown() {
    // Рухаємо спрайт вниз
    this.character.body.velocity.y = this.speed;
}


walkingLeft() {
    // Рухаємо спрайт вліво
    this.character.body.velocity.x = -this.speed;
}
```

Кінець лістингу 3.12

У лістингу 3.13 показано метод «PlayerStateRunning.update()», який викликає методи «Player.run*».

Лістинг 3.13 – Метод «PlayerStateRunning.update()»

```
update(keyboard: Phaser.Keyboard, cursors: Phaser.CursorKeys, came
ra: Phaser.Camera) {
    if (cursors.up.isDown) {
        // Рухаємо гравця вгору
        this.hero.runUp();
    }
       // в іншому випадку
    else if (cursors.down.isDown) {
```

```
        // Рухаємо гравця вниз, якщо він не виходить за межі екрана
         if (this.hero.sprite.body.y < camera.y + camera.height
- this.hero.sprite.height) {
             this.hero.runDown();
         }
    }

    if (cursors.left.isDown) {
        // Рухаємо гравця вліво
        this.hero.runLeft();
    }
    else if (cursors.right.isDown) {
        // Рухаємо гравця вправо
        this.hero.runRight();
    }
}
```

Кінець лістингу 3.13

Подія стрільби викликає метод «firePlayersBullet()», у якому відтворюється звук пострілу і до гри додається новий об’єкт зі спрайтом кулі. Цей снаряд може вразити ворогів і завдати їм шкоди (лістинг 3.14).

Лістинг 3.14 – Метод BaseLevel.firePlayersBullet()

```
firePlayersBullet() {
    // Створюємо новий об'єкт кулі
    let playersBullet: PlayersBullet = new PlayersBullet(this, thi
s.layer, this.bulletSound, this.hero, this.boss);
    playersBullet.setup();
    this.playersBullets.push(playersBullet);
    // Зменшуємо здоров'я після пострілу
    if (this.hero.decreaseHealth(2)) {
        this.updateHealthBar();
    }
}
```

Кінець лістингу 3.14

Коли наш герой отримує пошкодження, його здоров’я зменшується відповідно до типу пошкодження. Якщо залишкове здоров’я знижується до нуля, то герой отримує смертельний удар. У такому випадку ми показуємо анімацію загибелі героя, а потім грає відповідна музика. Після цього гравця перекидають на стартове меню.

З іншого боку, коли здоров’я збільшується, енергетична панель зростає відповідно до отриманої кількості. Максимальний рівень здоров’я на панелі – 100 одиниць.

Кожен клас ворогів успадковується від абстрактного класу, який містить індикатор зарядженої зброї, вектор швидкості, позицію ворога та номер ідентифікатора ворога (рис. 3.4).

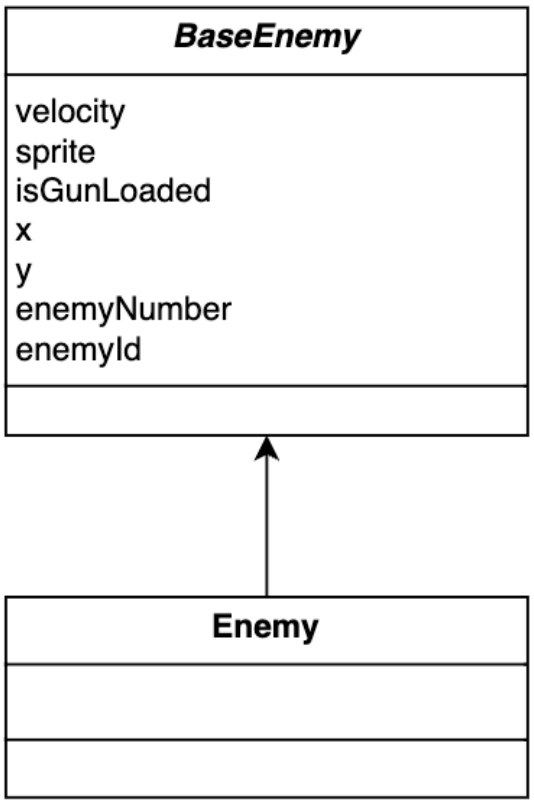

Рисунок 3.4 – Діаграма класів ворога

Клас «BaseEnemy» (додаток А) містить метод, який обробляє зіткнення з гравцем. Коли зіткнення підтверджується, гравець отримує шкоду, що відповідає силі ворога. Після цього ворог знищується. Також «BaseEnemy» дозволяє перевіряти зіткнення ворогів між собою. У цьому випадку жоден з них не знищується, але це забезпечує те, що вони не перекриваються і це додає реалізму грі.

3.3 Анімації в грі

В процесі гри камера переміщується знизу вгору. Це реалізовано в методі «scrollLevel()» (лістинг 3.15).

Лістинг 3.15 – Метод «BaseLevel.scrollLevel()» для прокручування рівнів

```
scrollLevel() {
    // Прокручуємо камеру рівня вгору
    this.game.camera.y -= this.calculateScrollStep();

    if (this.game.camera.y > 0) {
    // Переміщуємо статус бар та панель здоров'я разом з камерою
        this.healthBar.position.y -= this.calculateScrollStep();
        this.statusBar.position.y -= this.calculateScrollStep();
        if (this.player.state instanceof PlayerStateRunning) {
            this.player.walking();
        }
    }

    // Запускаємо подію прокрутки з затримкою
    this.game.time.events.add(Phaser.Timer.SECOND / 30, this.scrol
lLevel.bind(this));
}
```

Кінець лістингу 3.15

Для цього проєкту для анімації героя і ворогів використано спрайти з фіксованим розміром кадрів (лістинг 3.16).

Лістинг 3.16 – Сигнатура методу «AnimationManager.add()»

```
add(name: string, frames?: number[] | string[],
    frameRate?: number, loop?: boolean, useNumericIndex?: boolean)
: Phaser.Animation;
```

Кінець лістингу 3.16

За допомогою класу «AnimationManager» у Phaser, ми можемо додавати нові анімації кадрів, які потім будуть використовуватися для представлення різних станів гравця (лістинг 3.17).

Лістинг 3.17 – Метод «Player.setup()»

```
setup() {
    // Налаштовуємо анімації спрайту гравця
    this.sprite.animations.play('run'); // Починаємо з анімації
ходьби
    this.sprite.animations.add('hit', [4, 5, 6, 7, 4, 5, 6, 7], 5,
true);
    this.sprite.animations.add('run', [0, 1, 2, 3], 4, true);
    this.sprite.animations.add('die', [4, 5, 6, 7, 4, 5, 6, 7], 5,
true);
}
```

Кінець лістингу 3.17

3.4 Додавання звуків у грі

Звуки в цій грі відіграють не лише роль ресурсу, що робить гру цікавішою, але й мають важливу функцію: вони визначають інтервал між подіями. Наприклад, коли користувач натискає клавішу пробілу на екрані меню, починає грати музика, але перший рівень не розпочнеться, доки музика не закінчиться. Код, для відтворення музики показаний в лістингу 3.18.

Лістинг 3.18 – Метод «Menu.update()»

```
update() {
    if (this.game.input.keyboard.isDown(Phaser.KeyCode.SPACEBAR))
{
        // Робимо текст "PUSH SPACE KEY" прозорим
        this.pushSpaceKey.alpha = 0;
        // Налаштовуємо блимаючу анімацію для тексту
        this.game.add.tween(this.pushSpaceKey).to({ alpha: 1 }, 10
0,Phaser.Easing.Linear.None, true, 0, 50, true);
        // Відтворюємо звук старту
        this.startSound.play();
    }
}
```

Кінець лістингу 3.18

Спочатку програма перевіряє, чи натиснута клавіша пробілу. Якщо натиснута, то текст «PUSH SPACE KEY» стає прозорим (тобто, ми встановлюємо альфа-канал кольору на нуль).

Потім ми змушуємо текст блимати безкінечно, змінюючи альфа-канал кольору з 0 на 1 за допомогою методу «tween()». Без зупинки анімації ми відтворюємо музику старту.

Для того, щоб програма знала, коли закінчується музика, додамо код в методі «create()», де ініціалізується поле «startSound» (лістинг 3.19).

Лістинг 3.19 – Метод «Menu.create()»

```
create() {
    // Ініціалізація звуку старту
    this.startSound = this.game.add.audio('start');
    // Додавання події onStop для звуку старту
    this.startSound.onStop.add(function () {
        // Після завершення звуку переходимо до splash1
        this.game.state.start('splash1');
    }.bind(this));
}
```

Кінець лістингу 3.19

Тут можна бачити реалізацію події «onStop» для звуку «startSound». Коли звук закінчується, виконується функція, яка запускає рівень «splash1». Аналогічно, кожен інший звук у грі визначає, коли починаються наступні події: фонова музика, постріл, збільшення чи зменшення здоров’я, кінець гри, зіткнення.

Реалістичність гри можна покращити, запровадивши фізичні обмеження, такі як зіткнення. Інакше спрайти могли б накладатися один на одного або просто зникати з екрану, як привиди.

3.5 Реалізація взаємодій у грі

Взаємодія «гравець-карта». Карта рівня має встановлені межі (лівий та правий краї, а також перешкоди на шляху), тому ми не можемо дозволити нашому герою їх обходити. Phaser має чудову бібліотеку для виявлення зіткнень, яка дозволяє визначити, що гравець не може накладатися на інший об’єкт у грі.

Ми робимо це, викликаючи метод «game.physics.arcade.collide()», передаючи як аргументи спрайт гравця та об’єкт шару рівня (лістинг 3.20).

Лістинг 3.20 – Обробка зіткнень у методі Player.update()

```
this.game.physics.arcade.collide(this.sprite, this.layer);
```

Кінець лістингу 3.20

Взаємодія «гравець-ворог». Ми не можемо дозволити, щоб гравець та вороги накладалися один на одного. Ми визначаємо, що вони зіштовхуються, викликаючи метод «game.physics.arcade.collide()», але в цьому випадку ми також надаємо функцію, яка визначатиме поведінку після зіткнення гравця з тілом ворога: гравець отримує пошкодження або спрайт ворога знищується, видаляючи його з гри (лістинг 3.21).

Лістинг 3.21 – Обробка зіткнень у методі «Enemy.checkPlayerCollisions()»

```
checkPlayerCollisions() {
    this.game.physics.arcade.collide(this.sprite, this.player.spri
te, function () {

        // Гравець отримує шкоду від ворога
        this.level.playerWasHit(this);

        // Знищуємо спрайт ворога
        this.sprite.destroy();
    }.bind(this));
}
```

Кінець лістингу 3.21

Взаємодія «куля-ворог». Кулі є чудовим способом позбутися ворогів на далекій відстані, не зазнаючи ударів. Коли куля випущена, ми перевіряємо, чи зіштовхнулася вона з ворогом, і в такому випадку знищуємо і ворога, і кулю (лістинг 3.22).

Лістинг 3.22 – Обробка зіткнень у методі «PlayerBullet.update()»

```
this.level.enemies.forEach(enemy => {
```

```
    this.game.physics.arcade.collide(this.sprite, enemy.sprite, fu
nction () {
        // Зупиняємо кулю
        this.destroyed = false;

        // Визначаємо поведінку при попаданні кулі у ворога
        this.level.playerBulletHit(this, enemy);

        // Знищуємо спрайт кулі
        this.sprite.destroy();

        // Знищуємо спрайт ворога
        enemy.sprite.destroy();
    }.bind(this));
});
```

Кінець лістингу 3.22

Взаємодія «ворог-ворог». Вороги в нашій грі не повинні накладатися один на одного. Коли вони зіштовхуються, вони залишаються розділеними, ми нічого не знищуємо. Це додає трохи більше реалізму до гри (лістинг 3.23).

Лістинг 3.23 – Обробка зіткнень у методі «Enemy.checkEnemyCollisions()»

```
checkEnemyCollisions(enemies: BaseEnemy[]) {
    enemies.forEach(other => {
        if (this.id != other.id) {
            this.game.physics.arcade.collide(this.sprite, other.sp
rite, function () {
                // Нічого не робимо при зіткненні ворогів
            }.bind(this));
        }
    });
}
```

Кінець лістингу 3.23

Взаємодія «гравець-бос». Зіткнення з босом майже таке ж, як зіткнення з менш сильним ворогом, тобто, завдає шкоди гравцеві. Але в цьому випадку бос не знищується (лістинг 3.24).

Лістинг 3.24 – Обробка зіткнень у методі «Boss.update()»

```
this.game.physics.arcade.collide(this.sprite, this.player.sprite,
function () {
```

```
    if (this.player.sprite.animations.currentAnim.name == 'run') {
        // Гравець отримує удар від боса
        this.player.wasHit(this);
    }
}.bind(this));
```

Кінець лістингу 3.24

Взаємодія «куля-бос». Бос рівня може витримати кілька куль перед знищенням. Але наведений нижче код описує лише поведінку з точки зору кулі гравця. Управління шкодою боса знаходиться всередині класу «Boss» (лістинг 3.25).

Лістинг 3.25 – Обробка зіткнень у методі PlayerBullet.update()

```
this.game.physics.arcade.collide(this.sprite, this.boss.sprite, fu
nction () {
    // Знищуємо спрайт кулі
    this.sprite.destroy();
    // Визначаємо поведінку при попаданні кулі у боса
    this.level.playerBulletHit(this, this.boss);
    // Бос отримує удар
    this.boss.wasHit();
    this.destroyed = false;
}.bind(this));
```

Кінець лістингу 3.25

3.6 Розробка карт рівня

Як було сказано раніше, кожен рівень має фонове зображення, яке прокручується вниз, коли наш герой рухається по шляху. Кожен рівень також має відповідний текстовий файл, де кожен символ представляє позицію 32x32 на фоновому зображенні.

На прикладі файлу «Map01.txt» (додаток Б), крапка «.» представляє порожнє місце 32x32, а «X» може бути будь-якою перешкодою на зображенні, такою як стіни, труби, лабіринти тощо. Інші символи, такі як малі літери («a», «b», «c», «d»), відповідають різним ворогам, які повинні бути розміщені під час прокрутки карти.

Коли рівень запускається, він завантажує текстовий файл і заповнює матрицю вмістом ASCII файлу (лістинг 3.26).

Лістинг 3.26 – Зчитування карти у методі «BaseLevel.setupMap()»

```
let lines : string[] = this.readFile("/assets/maps/Map0" +
                       this.levelNumber + ".txt").split('\n');
for (let y = 0; y < lines.length; y++) {
    let line : string = lines[y];
    let lineArray : number[] = new Array(line.length);
    for (let x = 0; x < line.length; x++) {
        let char : string = line[x];
        if (char == 'X') {
            // Розміщення перешкод на карті
            this.map.putTile(1, x, y, this.layer);
            lineArray[x] = 1;
        } else {
            lineArray[x] = 0;
        }
    }
}
```

Кінець лістингу 3.26

Після завантаження матриці, необхідно створити всі об’єкти, знайдені на карті, такі як вороги та монети.

Клас «BaseLevel» містить екземпляри ворогів та монет. Ці масиви об’єктів заповнюються відповідно до вмісту матриці «mapAsStringArray». Кожен з цих ворогів створюється з позицією, знайденою на карті (лістинг 3.27).

Лістинг 3.27 – Метод «BaseLevel.setupEnemies()»

```
setupEnemies(mapAsStringArray: string[]) {
    // Ініціалізуємо масив ворогів
    this.enemies = [];
    // Символи, які представляють різні типи ворогів у файлі карти
    let enemyCodes: string = 'abcde';
    // Проходимо по кожному рядку файлу карти
    for (let y = 0; y < mapAsStringArray.length; y++) {
        let line: string = mapAsStringArray[y];
        // Проходимо по кожному символу в рядку
        for (let x = 0; x < line.length; x++) {
            let char: string = line[x];
            // Перевіряємо, чи символ відповідає ворогу
            let indexOf: number = enemyCodes.indexOf(char);
```

```
            if (indexOf >= 0) {
                // Ініціалізуємо змінні для ворога та його ідентиф
ікатора
                let enemy: BaseEnemy;
                let id: number = this.enemies.length + 1;
                // Визначаємо тип ворога на основі символу
                switch (indexOf) {
                    case 0:
                        // Створення ворога типу A
 enemy = new EnemyA(this, this.game, this.layer, this.bulletSound,
                    this.player, x * 32, y * 32, indexOf + 1, id);
                        break;
                    case 1:
                        // Створення ворога типу B
                        // ...
                        break;
                    case 2:
                        // Створення ворога типу C
                        // ...
                        break;
                    case 3:
                        // Створення ворога типу D
                        // ...
                        break;
                    case 4:
                        // Створення ворога типу E
                        // ...
                        break;
                }
                // Налаштування ворога
                enemy.setup();
                // Додавання ворога до масиву ворогів
                this.enemies.push(enemy);
            }
        }
    }
}
```

Кінець лістингу 3.27

Ми зчитуємо текстовий файл карти рівня та розділяємо його на рядки. Кожен рядок перетворюється на масив, де кожен символ представляє позицію на карті. Символ «X» розміщує перешкоду на відповідній позиції, інші символи залишаються порожніми. У методі «setupEnemies()» ми створюємо ворогів відповідно до символів карти, використовуючи малу літеру для ідентифікації типу ворога.

Вороги створюються з урахуванням їх позиції на карті, типу та унікального ідентифікатора.

Такий підхід дозволяє легко налаштовувати рівні, додаючи нові елементи та ворогів, просто змінюючи текстовий файл карти.

Процес проходження рівня в розробленій грі показаний на рисунку 3.5.

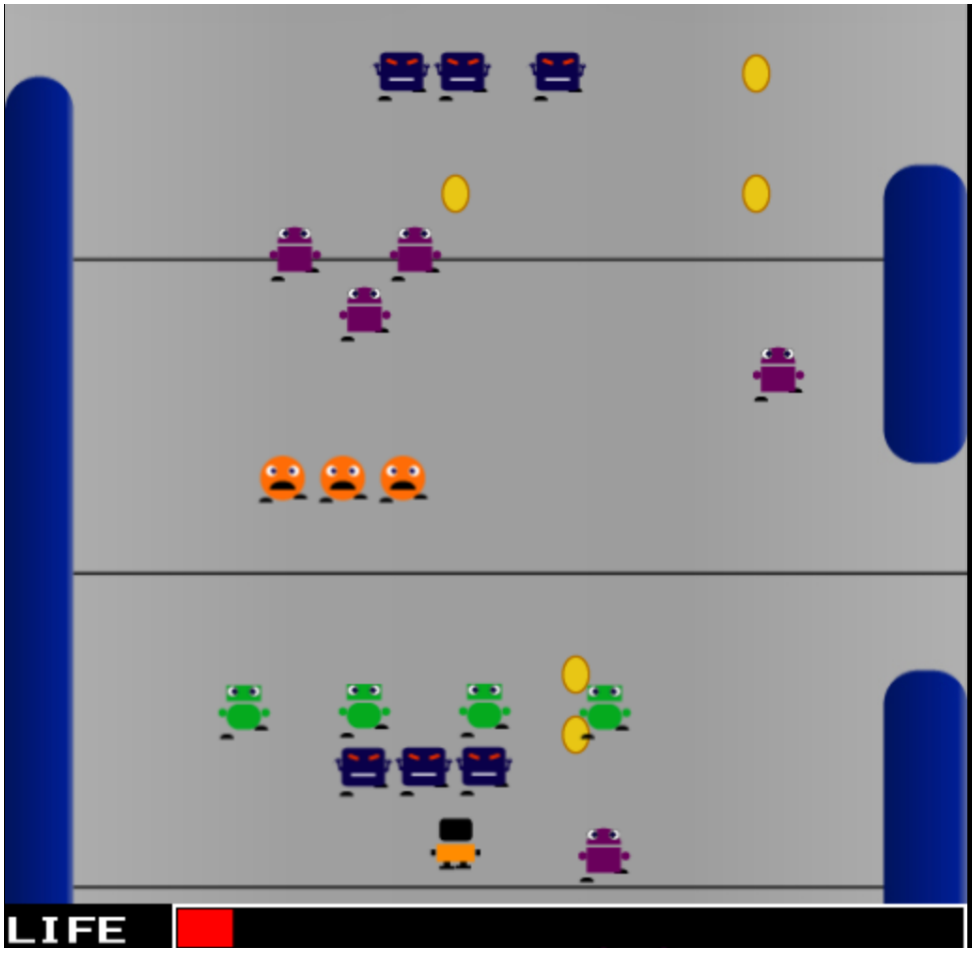

Рисунок 3.5 – Ігровий процес

3.7 Тестування та налагодження розробленої гри

Тестування та налагодження гри є критичними етапами в розробці для забезпечення її стабільності та якості.

План тестування включає в себе всі аспекти гри, які потрібно було перевірити:

– функціональність;

– продуктивність;

– юзабіліті;

– сумісність.

Юніт-тестування полягає в тестуванні окремих компонентів гри. Для цього використовувався фреймворк тестування Jasmine для JavaScript.

Приклад юніт-тесту наведено в лістингу 3.28.

Лістинг 3.28 – Приклад юніт-тесту для методу setupEnemies:

```
describe('BaseLevel', () => {
    it('should setup enemies correctly based on the map array', ()
=> {
        let mapArray = [
            '.....',
            '..a..',
            '..b..',
            '..c..',
            '.....'
        ];

        let baseLevel = new BaseLevel();
        baseLevel.setupEnemies(mapArray);

        expect(baseLevel.enemies.length).toBe(3);
        expect(baseLevel.enemies[0] instanceof EnemyA).toBe(true);
        expect(baseLevel.enemies[1] instanceof EnemyB).toBe(true);
        expect(baseLevel.enemies[2] instanceof EnemyC).toBe(true);
    });
});
```

Кінець лістингу 3.28

Інтеграційне тестування перевіряє взаємодію між компонентами гри. Тут ми перевіряємо, як різні частини гри працюють разом.

Приклад інтеграційного тесту для перевірки колізій між гравцем та ворогами наведено в лістингу 3.29.

Лістинг 3.29 – Інтеграційний тест

```
describe('Collision Detection', () => {
    it('should detect collision between player and enemy and handl
e it correctly', () => {
        let player = new Player();
```

```
        let enemy = new EnemyA();
        player.sprite.x = enemy.sprite.x;
        player.sprite.y = enemy.sprite.y;
        game.physics.arcade.collide(player.sprite, enemy.sprite, (
) => {
            player.wasHit(enemy);
            enemy.sprite.destroy();
        });
        expect(player.health).toBeLessThan(player.initialHealth);
        expect(enemy.sprite.exists).toBe(false);
    });
});
```

Кінець лістингу 3.29

Перевірка продуктивності допомагає впевнитися, що гра працює плавно на різних пристроях. Тут важливо оцінити частоту кадрів (FPS) і використання ресурсів. Тестування продуктивності здійснювали використовуючи інструменти розробника у веб-браузері Google Chrome. Мінімальна частота кадрів становила 68 кадрів за секунду, що є достатнім для комфортного ігрового процесу.

Перевірка юзабіліті дозволяє переконатися, що гра зручна для гравців. Це включає тестування інтерфейсу користувача, керування і загального досвіду користувача.

Здійснено тестування сумісності, працездатність гри перевірялася у різних браузерах: Google Chrome, Opera, Mozilla Firefox, Microsoft Edge, Safari, на операційних системах Windows 10 і MacOS, і встановлено, що гра працює однаково на всі цих платформах.

В процесі написання коду використовувалися інструменти розробника (Chrome DevTools) для виявлення синтаксичних помилок і налагодження JavaScript-коду.

Висновки до розділу 3

Проведено детальну розробку аркадної гри з використанням TypeScript і Phaser. Практична реалізація проекту включала кілька важливих етапів.

Було створено новий проект, у який додано файли Phaser та створено базову структуру гри, яка включає завантаження і ініціалізацію гри.

Було створено клас «Player» для головного героя гри з методами для його руху, атак, управління здоров'ям та іншими взаємодіями. Було також розроблено класи ворогів з відповідними методами для їхнього руху та зіткнень.

Реалізовано анімації для різних дій персонажів, включаючи ходьбу, атаки, отримання ушкоджень та загибель. Це забезпечило більш реалістичний і динамічний ігровий процес.

Було реалізовано відтворення звуків для різних дій гравця, таких як постріли, збір предметів та отримання ушкоджень.

Було розроблено механізми взаємодії між гравцем, ворогами, об’єктами та босами. Це включало обробку зіткнень, стрільбу та взаємодію з перешкодами.

Створено текстові файли для кожного рівня, які визначають розташування перешкод, ворогів та інших елементів. Це дозволило легко налаштовувати рівні та додавати нові елементи до гри.

Проведено юніт-тестування та інтеграційне тестування для перевірки функціональності окремих компонентів гри та їх взаємодії. Також було здійснено перевірку продуктивності, юзабіліті та сумісності гри на різних пристроях і в різних браузерах.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи бакалавра було виконано учі поставлені завдання.

Проведено детальний аналіз сучасного стану ігрової індустрії, включаючи тенденції, популярні жанри та технології. Було виявлено ключові проблеми розробки ігор, такі як необхідність кросплатформної підтримки, оптимізація продуктивності та забезпечення плавного ігрового процесу.

На основі аналізу було сформульовано вимоги до розроблюваної аркадної гри. Ці вимоги включали специфікації ігрового процесу, вимоги до графіки, звукового супроводу, анімацій та інтерфейсу користувача. Також було визначено технічні вимоги для забезпечення стабільної роботи гри в різних браузерах.

Обрано технології TypeScript і Phaser для розробки гри. Обґрунтування вибору включало переваги використання TypeScript для забезпечення статичної типізації та зручності підтримки коду, а також можливості Phaser для роботи з 2D-графікою, анімаціями та фізикою. Було також розглянуто інструменти розробки, такі як Visual Studio Community та графічні редактори Adobe Illustrator та Photoshop.

Здійснено практичну реалізацію гри, включаючи створення ігрового проекту, завантаження та обробку ресурсів, розробку ігрових персонажів та взаємодій, анімацій та звукового супроводу. Гра була створена з використанням TypeScript і Phaser, що дозволило забезпечити плавний ігровий процес та відповідність встановленим вимогам.

Проведено тестування гри для перевірки функціональності, продуктивності, юзабіліті та сумісності з різними браузерами та пристроями. Виявлені баги були задокументовані та виправлені. Гра успішно пройшла тестування, що підтвердило її стабільну роботу та відповідність усім встановленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Strong National Museum of Play. Video Game History Timeline – the strong National Museum of Play. URL: https://www.museumofplay.org/video-game-history-timeline/ (дата звернення: 07.02.2024).

2. Game – Wikipedia. URL: https://en.wikipedia.org/wiki/Game (дата звернення: 07.02.2024).

3. Despain Wendy. 100 Principles of Game Design. New Riders. 223 p. 2019.

4. Ігрові жанри. URL: https://training.qatestlab.com/blog/technical-articles/games-genres/ (дата звернення: 12.02.2024).

5. Agar.io. URL: https://agar.io/ (дата звернення: 07.02.2024).

6. 2048. Join the tiles, get to 2048! URL: https://play2048.co/ (дата звернення: 12.02.2024).

7. Knightmare (1986 video game) – Wikipedia. URL: https://en.wikipedia.org/wiki/Knightmare_(1986_video_game) (Дата звернення: 02.03.2024).

8. Konami. Knightmare – Abandonware DOS. URL: https://www.abandonwaredos.com/abandonware-game.php?abandonware=Knightmare&gid=981 (дата звернення: 04.03.2024).

9. What is a UML Diagram? | Different Types and Benefits | Miro. URL: https://miro.com/diagramming/what-is-a-uml-diagram/ (дата звернення: 07.02.2024).

10. JavaScript | MDN. ECMAScript URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 17.03.2024).

11. Phaser – A fast, fun and free open source HTML5 game framework. URL: https://phaser.io/ (дата звернення: 17.03.2024).

12. Visual Studio. URL: https://visualstudio.microsoft.com/ (дата звернення: 17.03.2024)

13. JavaScript and TypeScript in Visual Studio. URL: https://learn.microsoft.com/uk-ua/visualstudio/javascript/javascript-in-visual-studio?view=vs-2022 (дата звернення: 12.04.2024).

14. Adobe Creative Cloud | Подробиці та продукти | Adobe. URL: https://www.adobe.com/ua/creativecloud.html (дата звернення:18.04.2024).

15. Using Phaser with TypeScript. URL: https://phaser.io/tutorials/how-to-use-phaser-with-typescript (дата звернення: 03.05.2024).

ДОДАТКИ

Додаток А – Класи BaseEnemy і Player

```
class BaseEnemy {
    // Індикатор зарядженої зброї ворога
    isWeaponLoaded: boolean;
    // Вектор швидкості ворога
    velocity: Phaser.Point;
    // Позиція ворога на сцені
    position: Phaser.Point;
    // Номер ідентифікатора ворога
    enemyID: number;

    // Метод для обробки зіткнень з гравцем
    handleCollisionWithPlayer() {
        if (this.checkCollision(this.player)) {
            // Гравець отримує шкоду, відповідну силі вор
ога
            this.player.receiveDamage(this.strength);
            // Ворог знищується після завдання шкоди
            this.destroy();
        }
    }

    // Метод для обробки зіткнень з іншими ворогами
    handleCollisionWithEnemies(otherEnemies: BaseEnemy[])
{
        otherEnemies.forEach(enemy => {
            if (this.checkCollision(enemy)) {
                // Запобігаємо перекриттю ворогів
                this.separate(enemy);
            }
        });
    }
}

class Player {
    // Метод для обробки пошкодження гравця
    receiveDamage(damage: number) {
        this.health -= damage;
        if (this.health <= 0) {
            this.die();
        } else {
            this.sprite.animations.play('hit');
        }
    }
```

```
    // Метод для смерті гравця
    die() {
        this.sprite.animations.play('die');
        this.gameOver();
    }
}
```

Додаток Б – Приклад файлу карти рівня

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X............................X
X....X...X...X...X...X...X.....X
X...a.....b.....c.....d.....e..X
X....X...X...X...X...X...X.....X
X............................X
X....X...X...X...X...X...X.....X
X...e.....d.....c.....b.....a..X
X....X...X...X...X...X...X.....X
X............................X
X....X...X...X...X...X...X.....X
X...a.....b.....c.....d.....e..X
X....X...X...X...X...X...X.....X
X............................X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```