Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ДВОМІРНА ГРА - ЛАБІРИНТ ЗА ДОПОМОГОЮ JAVASCRIPT

TWO-DIMENSIONAL GAME - MAZE USING JAVASCRIPT

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Вознюк Богдан Володимирович

(підпис)

Керівник:
к.т.н., доц.
Гринюк Сергій Васильович

(підпис)

Кваліфікаційну роботу
допущено до захисту
« _____ » червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н.Чернящук
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Вознюк Богдан Володимирович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Двомірна гра - лабіринт за допомогою JavaScript*

Керівник роботи *к.т.н., доцент Гринюк Сергій Васильович*

затверджені наказом закладу вищої освіти від «28» грудня 2022 року № 982/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області та різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Вступ
Загальні відомості про сферу розробки ігор
Структура об'єкту розробки
Розробка гри за допомогою JavaScript
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Кваліфікаційна робота містить наступний графічний матеріал: знімки екрану середовищ розробки ігор, зображення лістингів коду, графічні складові елементи об'єкту розробки (гри-лабіринту), знімки екрану інструментів, які були використані під час розробки, та Знімок екрану з файловою структурою

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Загальні відомості про сферу розробки ігор	*Гринюк С.В.*		
Структура об'єкту розробки	*Гринюк С.В.*		
Розробка гри за допомогою JavaScript	*Гринюк С.В.*		
Висновки	*Гринюк С.В.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Обгрунтування теми*	15.11.2022 р.	Виконано
2.	*Огляд літератури на тему розробки ігор*	09.12.2022 р.	Виконано
3.	*Загальні відомості про сферу розробки ігор*	10.01.2023 р.	Виконано
4.	*Структура об'єкту розробки*	09.02.2023 р.	Виконано
5.	*Розробка гри за допомогою JavaScript*	22.03.2023 р.	Виконано
6.	*Висновки та пропозиції*	05.04.2023 р.	Виконано
7.	*Формування списку використаних джерел*	07.04.2023 р.	Виконано
8.	*Нормоконтроль*	20.05.2023 р.	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	02.06.2023 р	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	06.06.2023 р.	Виконано

Здобувач вищої освіти _____ (Вознюк Б.В.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Гринюк С.В.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Вознюк Б.В. Двомірна гра – лабіринт за допомогою JavaScript. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, двох додатків.

У першому розділі проведено аналіз стану проблеми та здійснено порівняльний аналізу середовищ та інструментів для розробки ігор, і здійснено вибір засобів розробки

У другому розділі було проведено обґрунтування обраних технологій та представлено функціонально–структурну схему складових гри.

У третьому розділі здійснено безпосередню розробку гри, а саме описано підготовку візуальних елементів гри, створення класів, об'єктів та обробників ігрових подій. Також детально розглянуто процес тестування та відлагодження гри. Розділ завершується описом процесу завершення розробки та завантаження гри на хостинг.

Об'єкт дослідження – технологій розробки веб–ігор.

Предмет дослідження – розробка двомірної веб–гри за допомогою мови програмування JavaScript.

Метою роботи є розробка гри за допомогою мови програмування JavaScript, дослідження можливостей та особливостей розробки ігор з використанням даної мови програмування, а також вирішення практичних завдань, пов'язаних з створенням візуальних елементів гри, роботою з класами, об'єктами та обробниками ігрових подій, тестуванням та відлагодженням гри.

Ключові слова: JavaScript, HTML Canvas, розробка ігор, веб–ігри, графіка, класи, об'єкти, обробники ігрових подій.

ANNOTATION

Vozniuk B.V. Two–dimensional game – maze using JavaScript. Manuscript.

Bachelor's qualification work of the «Computer Engineering» educational program, specialization 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

The qualification work consists of an introduction, three sections, conclusions, a list of references, and two appendices.

First section provides an analysis of the problem and a comparative analysis of environments and tools for game development, as well as the selection of development tools.

Seconds section justifies the chosen technologies and presents the functional–structural diagram of the game components.

Third section focuses on the direct development of the game, including the description of preparing visual elements, creating classes, objects, and event handlers. The testing and debugging process of the game is thoroughly examined. The chapter concludes with a description of the finalization of the development process and game hosting.

Object of research – web game development technologies.

Subject of research – development of a two–dimensional web game using the JavaScript programming language.

The aim of the work is to develop a game using the JavaScript programming language, explore the possibilities and peculiarities of game development using this programming language, and address practical tasks related to creating visual elements, working with classes, objects, and event handlers, as well as testing and debugging the game.

Keywords: JavaScript, HTML Canvas, game development, web games, graphics, classes, objects, event handlers.

ЗМІСТ

ВСТУП

Актуальність теми полягає в тому, що ігрова індустрія постійно розвивається і набуває все більшої популярності. Ігри стають не тільки розважальними продуктами, але і потужними інструментами для навчання, тренування та взаємодії з користувачами. Також, JavaScript є однією з найпоширеніших мов програмування веб–розробки і володіє широким спектром можливостей для розробки ігор. Використання JavaScript у розробці ігор дозволяє створювати веб–ігри, які доступні на різних платформах без необхідності встановлення додаткового програмного забезпечення.

Мета роботи полягає у розробці гри за допомогою мови програмування JavaScript і вивчення особливостей розробки ігор в веб–середовищі. Основними завданнями є аналіз стану проблеми у сфері розробки ігор, вибір та обґрунтування засобів розробки, створення візуальних елементів гри, робота з класами, об'єктами та обробниками ігрових подій, тестування та відлагодження гри, а також завершення розробки та завантаження гри на хостинг.

Об'єкт дослідження – сфера розробки сучасних веб–ігор.

Предмет дослідження – розробка веб–гри за допомогою мови програмування JavaScript.

Завдання, які необхідно виконати:

– проаналізувати інформаційні джерела, пов’язані зі сферою розробки ігор.

– спроектувати структуру об’єкту розробки.

– створити графічні елементи для гри, використовуючи графічні редактори.

– провести тестування та відлагодження гри.

– завантажити гру на окремий сервер (хостинг), для забезпечення віддаленого доступу.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ ПРО СФЕРУ РОЗРОБКИ ІГОР

1.1 Аналіз стану проблеми

Веб–ігри стають все більш популярними серед користувачів, особливо в мобільних та онлайн–середовищах. Створення 2D гри–лабіринту може бути цікавим та захоплюючим проектом, який зможе залучити гравців своєю геймплейною механікою та викликати цікаві емоції. Використання мови JavaScript та HTML5 Canvas для розробки гри–лабіринту забезпечує швидку розробку, високу переносимість між різними платформами та простоту розгортання на веб–сервері. Мова JavaScript є широко використовуваною веб–технологією, що має велику спільноту розробників, що відкриває доступ до великої кількості ресурсів та бібліотек для розробки.

Прикладною галуззю об'єкта розробки є галузь розваг та геймдевелопменту, з фокусом на створенні веб–ігор.

Для розробки веб–ігор засобами JavaScript та HTML5 Canvas, необхідно мати веб–браузер та редактор коду [1]. Додатково, для роботи з графікою необхідний GIMP – графічний редактор для створення графічних елементів, та Tiled Map Editor – редактор для створення карт лабіринту.

Розробка вихідного об'єкту вимагає базові знання з програмування на JavaScript, розробки веб–додатків, роботи з графікою та графічними редакторами.

Формат інформаційних потоків у вигляді текстів (діалоги), зображень (ігрові спрайти та ассети) та аудіо (звукові ефекти), відповідно до вимог гри та функціональності, яку потрібно реалізувати.

Вихідним об'єктом є 2D гра–лабіринт, з багатьма рівнями, можливістю керування головним героєм, взаємодією з об'єктами гри, реалізацією правил гри та відображенням графічних елементів на екрані гравця.

1.2 Здійснення аналізу та вибору засобів розробки

Для розробки браузерних 2D ігор, існує багато доступних засобів, інструментів та способів розробки. Найпопулярнішими на сьогоднішній день рішеннями є:

– Phaser (табл. 1.1). Phaser є одним з популярних фреймворків для розробки браузерних 2D ігор. Він базується на мові програмування JavaScript і надає широкий набір функцій, таких як анімація, фізика, звук, спрайти, колізії та інші. Phaser має велику спільноту розробників та велику кількість документації, що робить його досить популярним вибором для розробки браузерних ігор [2].

Таблиця 1.1 – Характеристика Phaser

Переваги	Недоліки
Широкий набір функціональності, включаючи анімацію, фізику, звук та інші.	Вимагає знань в програмуванні на JavaScript.
Велика спільнота розробників та велика кількість документації.	Не має графічного інтерфейсу розробки, все виконується через код.
Підтримка мобільних пристроїв та браузерів.	

– CreateJS (табл. 1.2). CreateJS є комплексним набором бібліотек для розробки браузерних ігор, включаючи EaselJS для роботи з графікою, TweenJS для анімації, та інші. Він використовує мову програмування JavaScript і надає зручні засоби для розробки 2D ігор з графічним інтерфейсом [3].

Таблиця 1.2 – Характеристика CreateJS

Переваги	Недоліки
Комплексний набір бібліотек для різних аспектів розробки ігор.	Не такий популярний, як Phaser, тому може бути менше документації.
Графічний інтерфейс розробки	Обмежена функціональність порівняно з деякими іншими фреймворками.

– Unity (табл. 1.3). Unity (рис. 1.1) є одним з найпопулярніших інструментів розробки ігор, який використовується для розробки як 2D, так і 3D ігор. Він забезпечує високий рівень гнучкості та функціональності, включаючи фізику, анімацію, колізії, штучний інтелект та багато інших можливостей. Unity використовує мову програмування C# для розробки ігор [4].

Таблиця 1.3 – Характеристика Unity

Переваги	Недоліки
Високий рівень функціональності та гнучкості.	Складність вивчення та використання, особливо для початківців.
Підтримка розробки як 2D, так і 3D ігор.	
Велика спільнота розробників та велика кількість ресурсів та документації.	Вимагає встановлення та налаштування спеціального ПЗ.
Можливість експорту готових ігор на різні платформи, такі як веб, мобільні пристрої, ПК та інші.	

Рисунок 1.1 – Інтерфейс Unity

– Pixi.js (табл. 1.4). Pixi.js є швидким та легким фреймворком для розробки 2D ігор в браузері, використовуючи WebGL для апаратного прискорення графіки. Він пропонує високий рівень продуктивності та можливостей, таких як рендеринг спрайтів, фільтри, анімація та інші [5].

Таблиця 1.4 – Характеристика Pixi.js

Переваги	Недоліки
Висока продуктивність завдяки використанню WebGL та апаратного прискорення графіки.	Не має підтримки для 3D–графіки, фізики та інших розширених можливостей, які можуть бути потрібні для деяких типів ігор.
Легкий у використанні та має чистий синтаксис.	
Забезпечує велику кількість можливостей для розробки браузерних 2D ігор.	Менша спільнота розробників порівняно з більш відомими фреймворками.

– HTML5 Canvas (табл. 1.5). HTML5 Canvas є вбудованим веб–технологією для рендерингу графіки в браузері, яка може бути використана для розробки браузерних 2D ігор. Вона використовує мови програмування JavaScript, HTML та CSS для розробки ігор [6].

Таблиця 1.5 – Характеристика HTML5 Canvas

Переваги	Недоліки
Легко використовувати вже вбудовану веб–технологію без необхідності встановлення додаткових інструментів.	Обмежена функціональність порівняно з іншими фреймворками та інструментами розробки, такими як фізика, анімація, штучний інтелект та інші.
Широке застосування в розробці веб–ігор та веб–додатків.	Вимагає ручного написання багато коду, що може бути витратним на час та ресурси.

Через простоту використання та доступність «з коробки» вибір впав саме на HTML5 Canvas. Розробка гри на базі цієї технології дозволить краще вивчити саме «чисті» JavaScript, HTML та CSS, що в перспективі дає ширші можливості у виборі сфери застосування отриманих знань.

При обговоренні інструментів для розробки ігор, варто також не забувати про засоби для створення візуальної частини, а саме – ігрових рівнів, персонажів,

текстур, і т.д [7]. Існує кілька різних засобів та інструментів, які можуть бути використані для створення 2D графіки. Декілька з таких засобів включають:

– Редактори рівнів у фреймворках та ігрових двигунах. Багато ігрових двигунів та фреймворків мають вбудовані редактори рівнів, які дозволяють розробникам створювати 2D рівні безпосередньо у візуальному середовищі. Наприклад, Unity, Unreal Engine, Godot та інші фреймворки мають вбудовані редактори рівнів з великим набором інструментів для створення 2D рівнів, таких як розміщення об'єктів, налаштування фізики, робота зі шарами та інші функції.

Самостійні редактори рівнів. Існують також окремі редактори рівнів, спеціалізовані на створенні 2D рівнів. Наприклад, Level Editor [8] (рис. 1.2), Tiled [9] (рис. 1.3), Tilemap Studio та інші, ці редактори надають засоби для створення 2D рівнів, таких як розміщення тайлів, налаштування колізій, робота з об'єктами, налаштування ефектів та інших властивостей рівнів.

Рисунок 1.2 – Level Editor (LEd) [8]

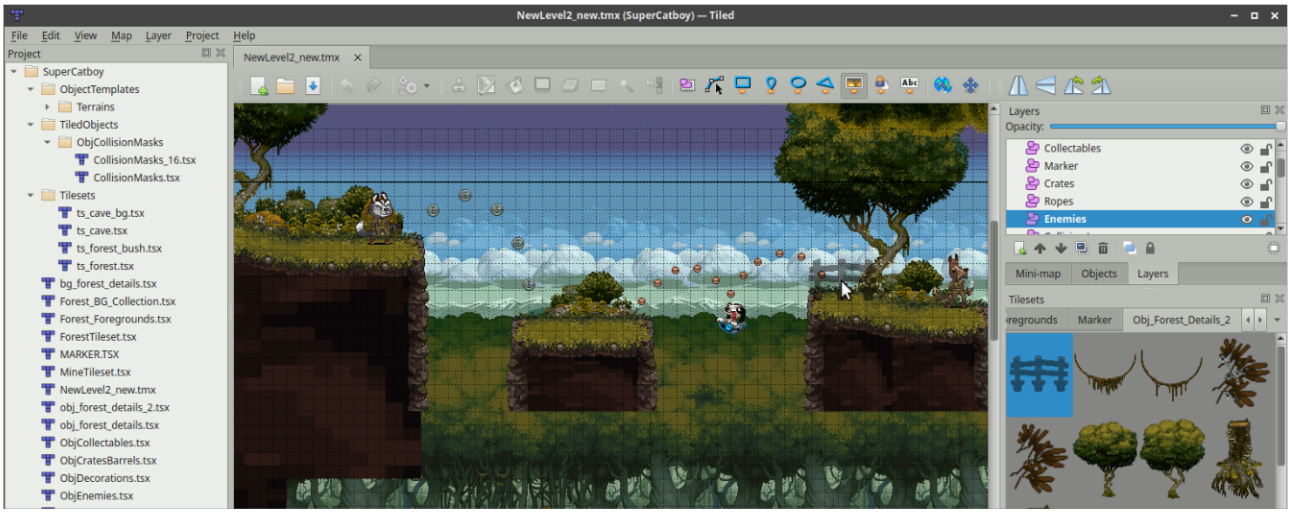

Рисунок 1.3 – Tiled Map Editor [9]

– Редактори графіки. Графічні редактори, такі як Photoshop (рис. 1.4), GIMP (рис. 1.5), Aseprite, Pyxel Edit та інші, можуть бути використані для створення графічних елементів 2D рівнів, таких як тла, тайли, перешкоди, об'єкти, персонажі та інші графічні елементи.

Рисунок 1.4 – Photoshop

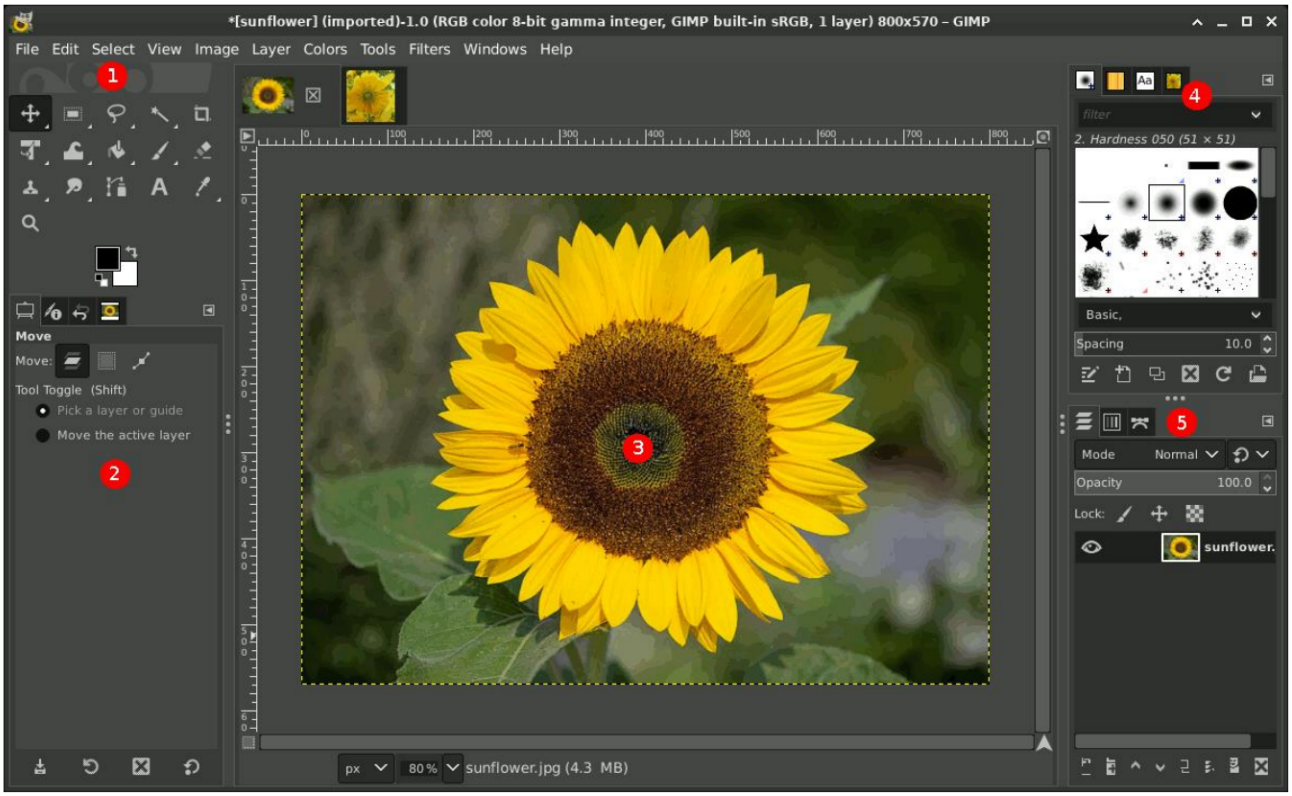

Рисунок 1.5 – GIMP

– Кодування: Для деяких розробників, створення 2D рівнів може включати написання власного коду, такого як скрипти або сценарії, для реалізації функцій рівнів, таких як логіка гри, рух об'єктів, динамічні події та інші взаємодії на рівні. Сюди також можна віднести процедурну генерацію. Процедурна генерація – це метод створення вмісту, такого як графіка, звук, рівні, персонажі, об'єкти тощо, випадковим або алгоритмічним шляхом за допомогою комп'ютерних програм або алгоритмів. Замість того, щоб ручним чином створювати кожен елемент окремо, процедурна генерація дозволяє автоматизувати процес створення великої кількості вмісту.

– Фізичні двигуни. Фізичні двигуни, такі як Box2D, Chipmunk, Matter.js та інші, можуть бути використані для реалізації фізики в 2D рівнях. Вони надають можливості для моделювання реалістичної фізики об'єктів, таких як гравітація, колізії, рух та інші фізичні взаємодії [10].

– Інструменти для анімації. Анімація є важливою частиною 2D рівнів, і для її створення можуть використовуватись спеціальні інструменти, такі як Spine,

DragonBones, Spriter та інші. Вони дозволяють створювати анімації персонажів, об'єктів, ефектів та інших елементів рівнів.

– Ресурсні бібліотеки та пакети графічних елементів: Існують також різні ресурсні бібліотеки та пакети графічних елементів, такі як Kenney.nl, OpenGameArt.org, GameDevMarket та інші, які містять готові графічні ресурси, такі як тайли, тла, персонажі та інші елементи, які можуть бути використані для швидкої розробки 2D рівнів.

Для розробки рівнів був обраний редактор рівнів Tiled Map Editor, а для створення ігрової візуальної графіки (головний персонаж, противники, інтерфейс, ігрові об'єкти) був використаний Gimp.

GIMP (GNU Image Manipulation Program) – це вільний, відкритий і безкоштовний графічний редактор, який використовується для редагування та обробки зображень. GIMP є одним з найпопулярніших альтернативних редакторів фотографій до комерційного Adobe Photoshop, і він доступний для використання на різних операційних системах, включаючи Windows, macOS та Linux [11].

GIMP надає широкий спектр функцій для редагування зображень, таких як корекція кольору та контрасту, видалення елементів, ретушування, малювання, створення графічних ефектів, робота з шарами, масками, фільтрами та іншими інструментами для обробки зображень. GIMP також підтримує роботу з різними форматами зображень, включаючи JPEG, PNG, GIF, TIFF та багато інших.

Visual Studio Code (VS Code) (рис. 1.6) – це інтегроване середовище розробки (IDE), розроблене компанією Microsoft, яке широко використовується розробниками для розробки різноманітних програм. VS Code надає потужний редактор коду з відмінною підсвіткою синтаксису, автодоповненням, перевіркою помилок та відладкою, що робить його відмінним вибором для розробки JavaScript, HTML та CSS коду, що використовується в браузерних іграх [12].

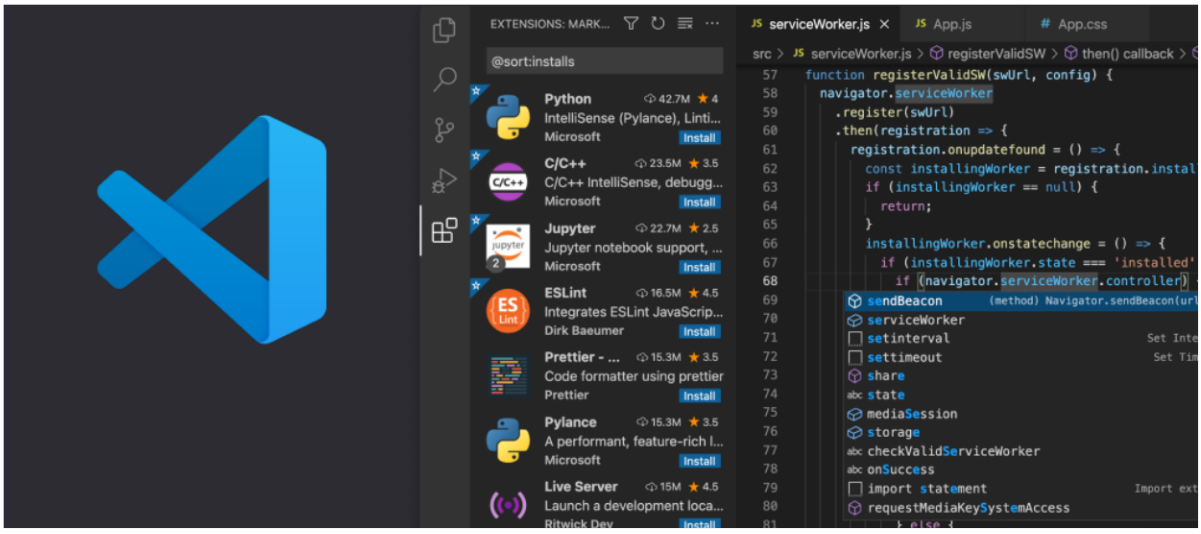

Рисунок 1.6 – Microsoft Visual Studio Code

VS Code має велику кількість розширень, розроблених спільнотою розробників, що дозволяє розширювати функціональність редактора. Під час розробки було використане розширення «Live Server» (рис. 1.7). Це розширення надає можливість легко запускати локальний веб–сервер, і кожного разу, коли розробник зберігає зміни у своєму коді, сторінка автоматично оновлюється, що дозволяє бачити результати в реальному часі без необхідності вручну перезавантажувати сторінку.

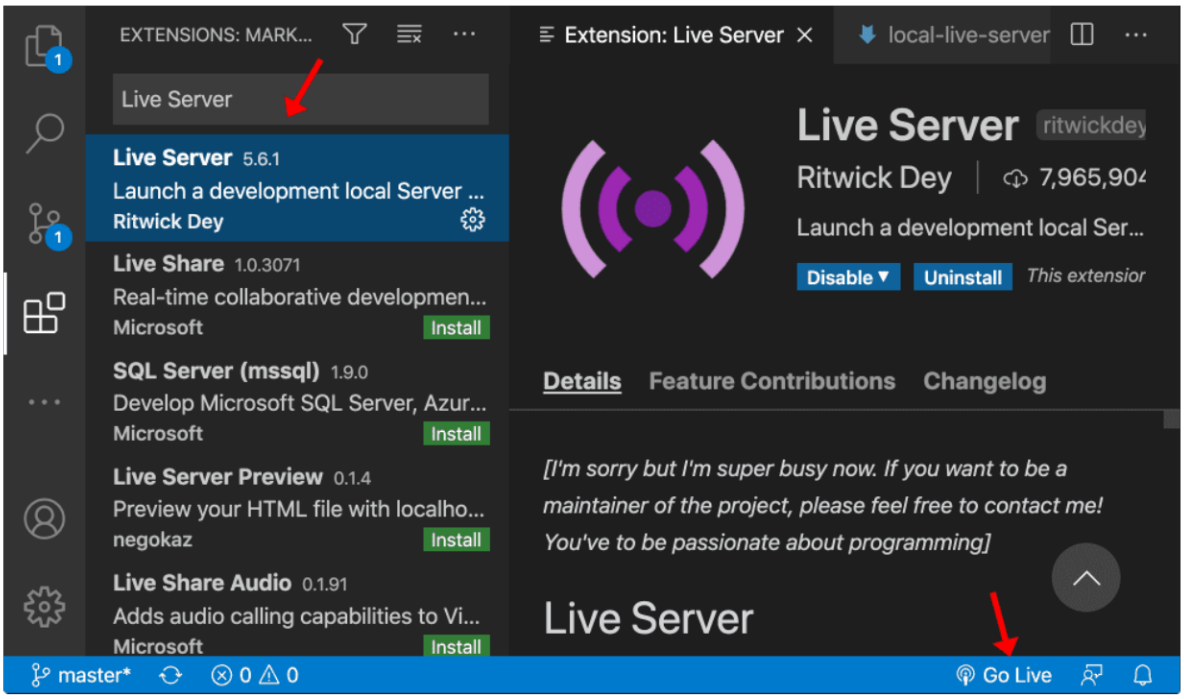

Рисунок 1.7 – Розширення “Live Server”

РОЗДІЛ 2
СТРУКТУРА ОБ’ЄКТУ РОЗРОБКИ

2.1 Обґрунтування обраних технологій

За призначенням, обрані технології можна розділити за двома категоріями: технічні (JavaScript та HTML5 Canvas) та візуальні (GIMP і Tiled Map Editor).

Як можна зрозуміти з класифікації, «візуальні» інструменти будуть використані для створення графіки гри. Tiled Map Editor – це інструмент, який дозволяє створювати 2D карти. Його можна використовувати для створення рівнів гри, розміщення тайлів, об'єктів, колізій та інших елементів. Tiled Map Editor підтримує різні типи тайлсетів та шарів, що дозволяє створювати складні рівні з різними елементами. GIMP – це графічний редактор, який буде використаний для створення графіки гри, а конкретніше, спрайтів (зображення персонажів, об'єктів тощо), тайлсетів (зображення, які використовуються для створення карт), анімації, обробки текстур та іншої графіки. Використовуючи Tiled Map Editor, можна імпортувати тайлсети, створені у GIMP, і розміщувати їх на ігровій карті.

JavaScript та HTML5 Canvas у свою ж чергу будуть відповідати за все, що буде відбуватися «під капотом» гри. JavaScript дозволить додати взаємодію між елементами та динаміку до гри, а конкретніше, створити логіку гри, руху персонажів, взаємодію з об'єктами гри, анімації та багато іншого [13]. HTML5 Canvas у свою ж чергу дозволяє малювати графіку безпосередньо на веб–сторінці, а створювати графічні ефекти, рендерити графіку, анімації, керувати мишкою та клавішами, обробляти колізій та багато іншого. Після створення рівнів гри у Tiled Map Editor, вони будуть імпортовані у проект саме за допомогою JavaScript та HTML5 Canvas. Також, за їх допомогою, будуть створені анімації у грі. Анімовані спрайти персонажів, створені в GIMP, будуть анімовані шляхом циклічної зміни кадрів під час руху об’єктів.

Також, засоби JavaScript будуть використані для відтворення музики та звукових ефектів у грі.

2.2 Функціонально–структурна схема складових гри

Директорія з файлами гри, складається з 6 папок, 5 файлів зі скриптами, та файлу index.html (рис. 2.1). Вміст кожної папки проекту описаний у таблиці 2.1.

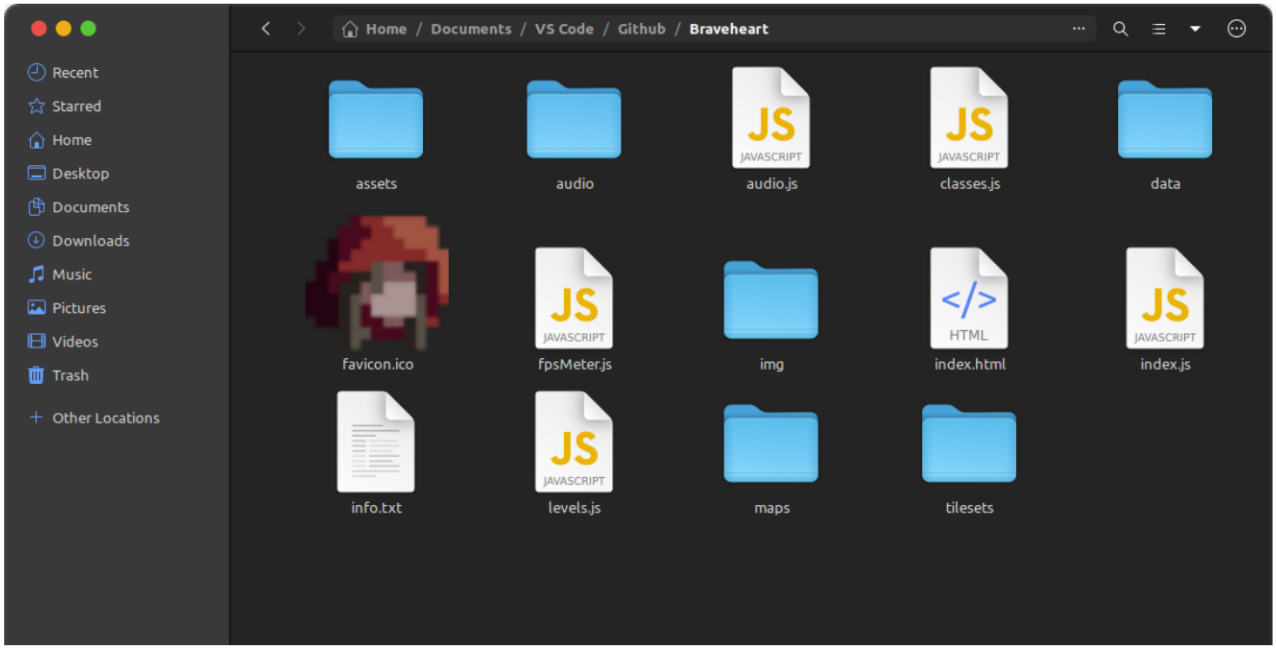

Рисунок 2.1 – Файлова структура проекту

Таблиця 2.1 – Вміст папок проекту

Папка	Вміст
assets	Проекти GIMP, та .png файли деяких текстур.
audio	.mp3 файли музики та звукових ефектів.
data	Файл collisions.js, у якому описані координати усіх меж, що розташовані на кожному рівні, а також папка dialogs, що містить .png файли спливаючих діалогів.
img	Усі .png, що використані у грі (візуальні об’єкти, персонажі, зображення мап, тощо).
maps	Усі файли–проекти мап Tiled Map Editor
tilesets	Файли–текстури Tiled Map Editor

У скриптових файлах, що розміщені в кореневій директорії проекту, описуються класи, ігрові події, їх обробники, функції, ефекти, тощо.

Додатково, у файлі index.html (додаток А) за допомогою тегу <script> відбувається імпорт додаткових JavaScript бібліотек (лістинг 2.2) Howler (для відтворення аудіо) та GSAP (для створення анімації переходу між рівнями) [14].

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/howler/2.2.3/
howler.min.js"
        integrity="sha512-
6+YN/9o9BWrk6wSfGxQGpt3EUK6XeHi6yeHV
+TYD2GR0Sj/cggRpXr1BrAQf0as6XslxomMUxXp2vIl+fv0QRA=="
        crossorigin="anonymous" referrerpolicy="no-
referrer"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.11.4/g
sap.min.js"
        integrity="sha512-f8mwTB
+Bs8a5c46DEm7HQLcJuHMBaH/UFlcgyetMqqkvTcYg4g5VXsYR71b3qC8
2lZytjNYvBj2pf0VekA9/FQ=="
        crossorigin="anonymous" referrerpolicy="no-
referrer"></script>
```

Рисунок 2.2 – Імпорт зовнішніх JavaScript

Після завершення розробки, гру необхідно буде завантажити на хостинг. Так як загальний розмір усієї вихідної директорії не дуже великий (близько 60 МБ), можна використувати практично будь–який дешевий або безкоштовний хостинг. Було обране рішення від компанії InfinityFree, що надає безкоштовну та необмежену у часі платформу для хостингу невеликих веб–сайтів та веб–додатків, та логічно зрозумілий домен, у вигляді «назва_проекту» .infinityfreeapp.com. Цей варіант цілком задоволяє усі вимоги проекту [15].

Для завантаження файлів на сервер буде використаний FTP–клієнт FileZilla (рис. 2.3).

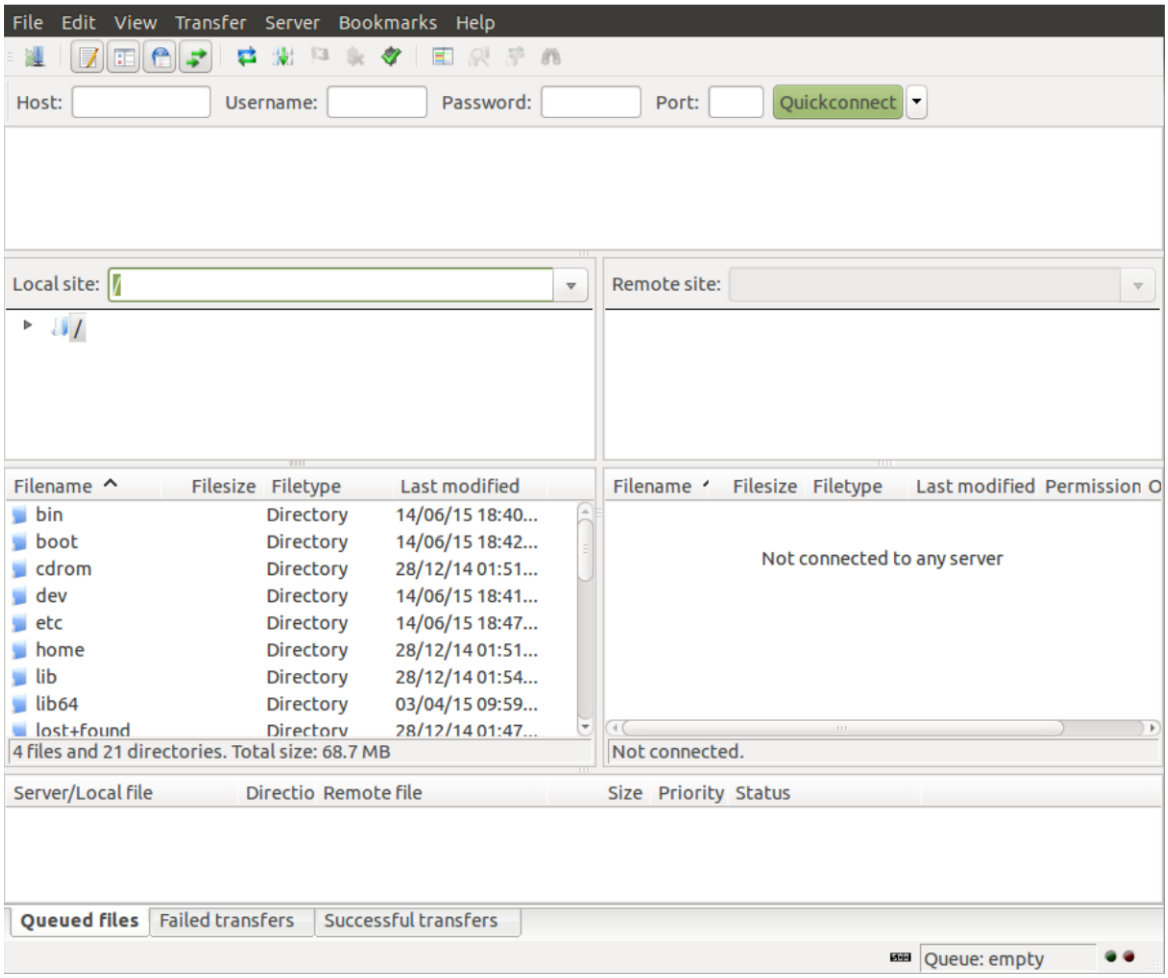

Рисунок 2.3 – FTP–клієнт FileZilla

FileZilla – це безкоштовний відкритий FTP–клієнт, який дозволяє встановлювати з'єднання з FTP–серверами та передавати файли між комп'ютером користувача і сервером. Він підтримує перетягування файлів, керування через керуючий знімок, підтримку проксі–серверів, шифрування передачі даних за SSL/TLS і багато інших функцій. FileZilla є дуже поширеним і популярним засобом для роботи з FTP–серверами, особливо в веб–розробці, де він використовується для завантаження файлів на веб–сервер для публікації веб–сайтів [16].

FTP, або File Transfer Protocol, є протоколом передачі файлів, який використовується для передачі файлів між комп'ютерами в мережі Інтернет. Він є одним з найпоширеніших способів передачі файлів, особливо в контексті веб–розробки та управління веб–сайтами [17].

РОЗДІЛ 3
РОЗРОБКА ГРИ ЗА ДОПОМОГОЮ JAVASCRIPT

3.1 Підготовка візуальних елементів гри

Першочергово, необхідно створити усі візуальні елементи, що знадобляться при розробці гри. Насамперед це ігрові персонажі (головний герой, противники), мапа, ігрові об’єкти (наприклад, портали) та візуальні ефекти [18].

Анімовані об’єкти (головний герой (червона шапочка) та противники (скелет та бджола)) були створені у GIMP у вигляді об’єднаних слайдів (на рис. 3.1, рис. 3.2, рис. 3.3, рис. 3.4), які будуть змінюватися безпосередньо під час руху.

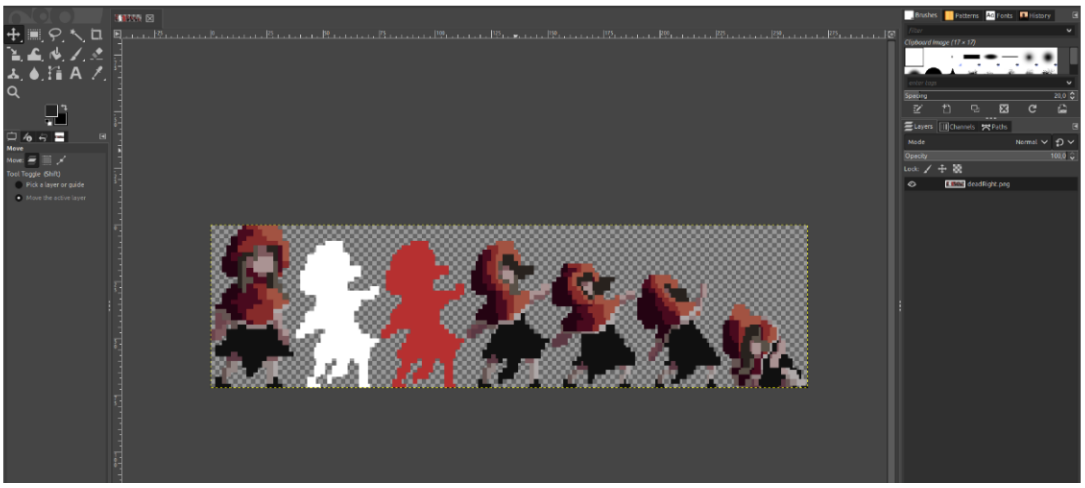

Рисунок 3.1 – Анімація смерті головного героя

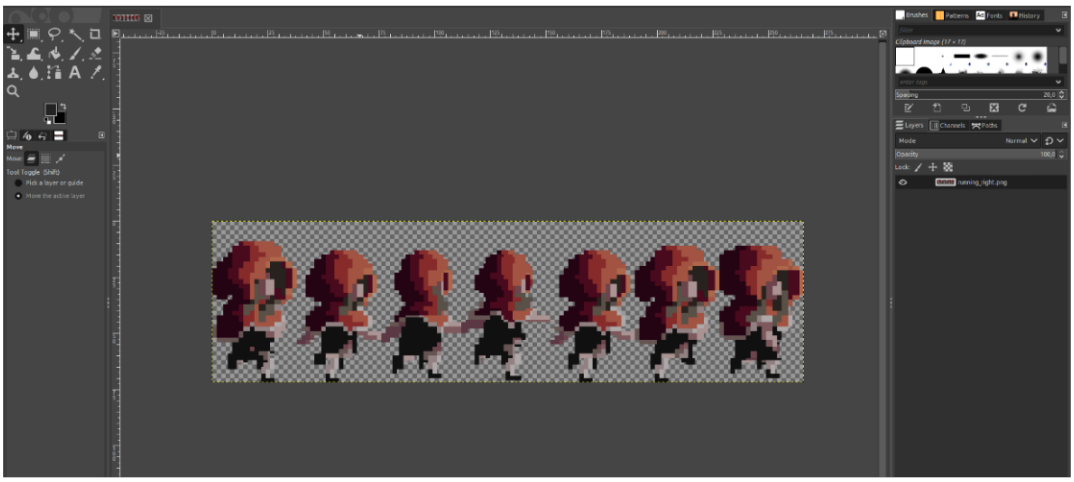

Рисунок 3.2 – Анімація руху головного героя

Рисунок 3.3 – Анімація руху скелета

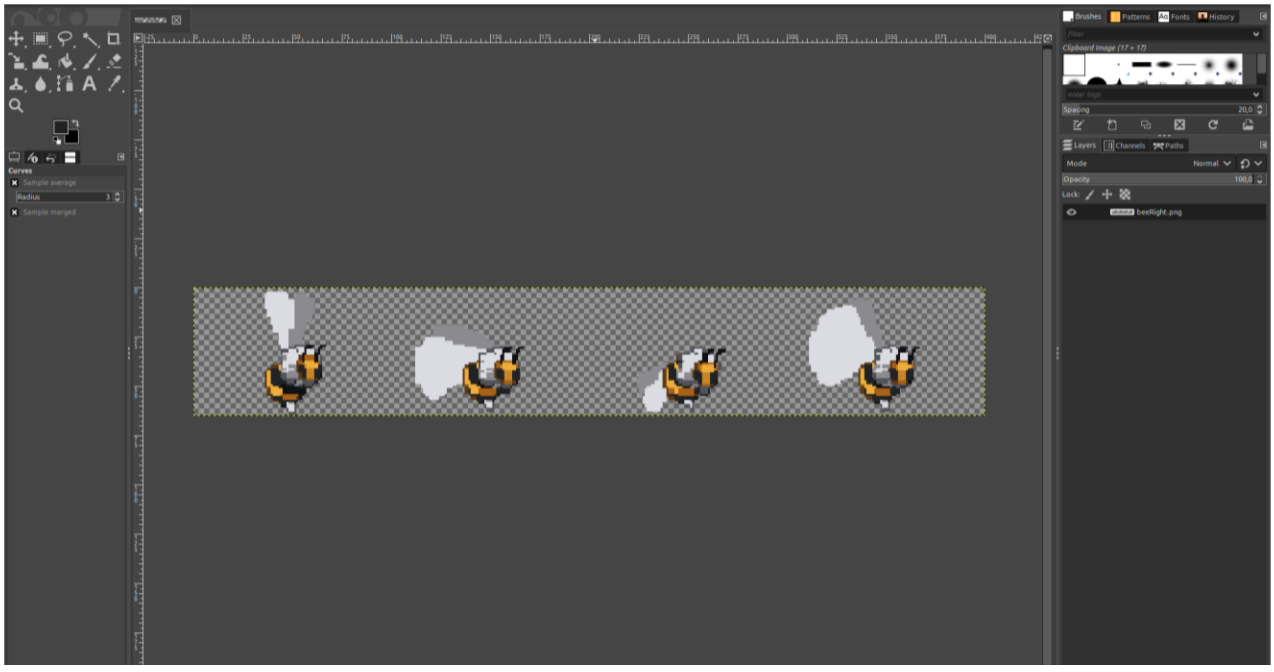

Рисунок 3.4 – Анімація руху бджоли

Мапа гри (рис. 3.5, рис. 3.6) була створена у Tiled Map Editor , та складається з 17 окремих, з’єднаних між собою частин.

Рисунок 3.5 – Повна мапа гри (верхня половина)

Рисунок 3.6 – Повна мапа гри (нижня половина)

Основна концепція створення карт в Tiled Map Editor полягає в роботі з тайлсетами (наборами тайлів) (рис. 3.7, рис. 3.8) та слоями (логічними рівнями) на цих тайлсетах. При розробці, тайлсетів була завантажена з безкоштовних джерел в Інтернеті, а інша половина – створена власноруч у програмі GIMP.

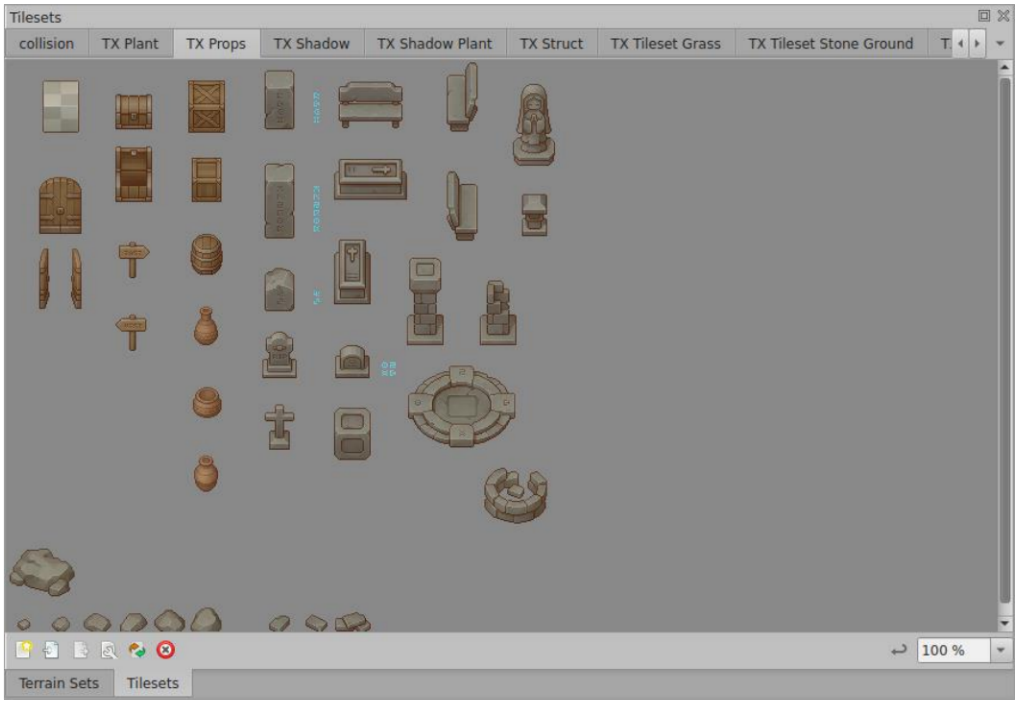

Рисунок 3.7 – Тайлсети з елементами оточення

Рисунок 3.8 – Тайлсети з деревами та кущами

Варто згадати, що окремо також у Tiled були створені елементи переднього плану (рис. 3.9), такі як листя дерев, дахи будівель, тощо. Ці елементи будуть «перекривати» персонажів гри, тобто, знаходитися над ними. Візуально це можна представити у вигляді 3 шарів, де перший слой – це безпосередньо мапа, другий слой – персонажі (головний герой, противники), а третій – елементи переднього плану.

Рисунок 3.9 – Елементи переднього плану з прозорим фоном

Додатково, у Tiled, для кожної локації був створений шар з колізіями меж локації (рис. 3.10). Колізія в контексті створення ігрових карт відноситься до властивості об'єктів або шарів, яка визначає їхню здатність взаємодії з іншими об'єктами в грі. Колізія може визначати, чи можуть об'єкти перешкоджати один одному, взаємодіяти з фізичними об'єктами, проходити через певні області на карті, або взаємодіяти зі світом гри взагалі.

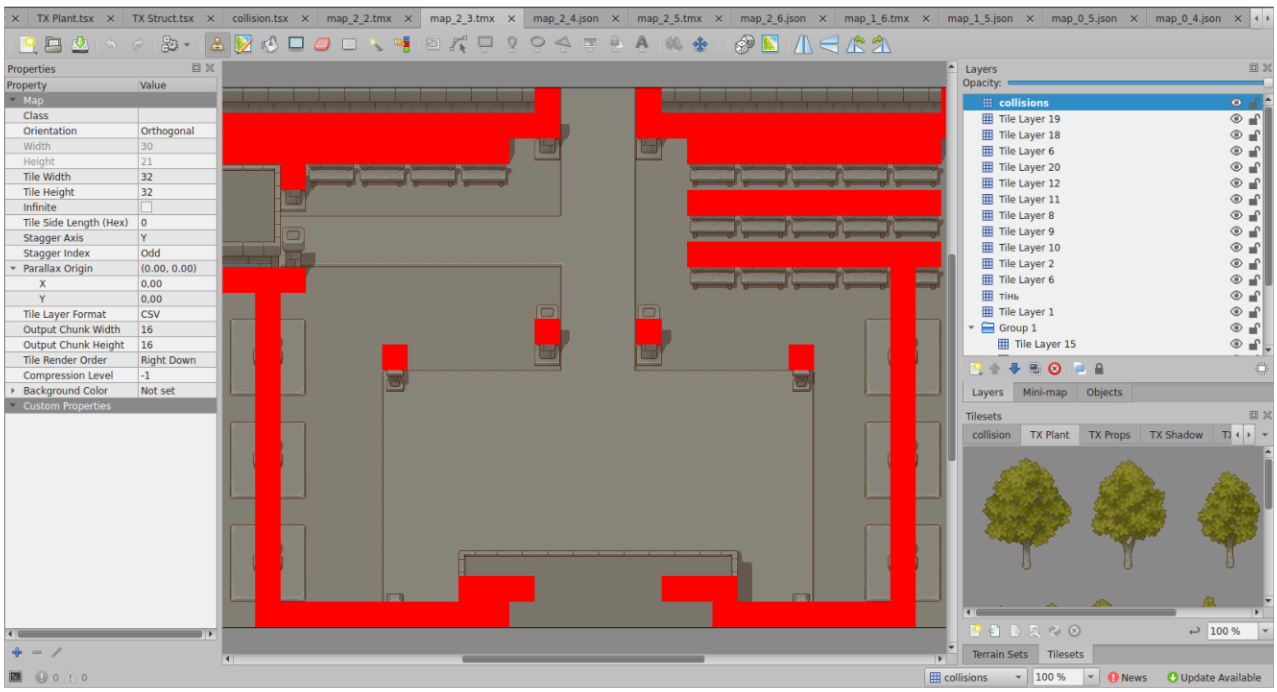

Рисунок 3.10 – Розміщення шару з колізіями на рівні

Колізія може бути встановлена на рівні тайлів (так званих «тайлових колізій») або на рівні окремих об'єктів. Наприклад, можна налаштувати колізію на об'єкті персонажа, щоб визначити, як він взаємодіє з оточенням: чи може він пересуватися через певні області, чи взаємодіє з іншими об'єктами, такими як стіни, платформи, перешкоди тощо [19].

Колізія використовується для створення реалістичного взаємодії об'єктів в грі, таких як рух персонажів, обробка зіткнень, розрахунок фізики та інші аспекти геймплею. Вона є важливим елементом процесу розробки ігор, допомагає забезпечити правильну взаємодію об'єктів на карті та досягти бажаних ефектів геймплею.

При виявленні колізії головного персонажа з противником, на окремому слої буде з'являтися екран поразки (рис. 3.11), а при колізії з точкою фінішу – екран перемоги (рис. 3.12).

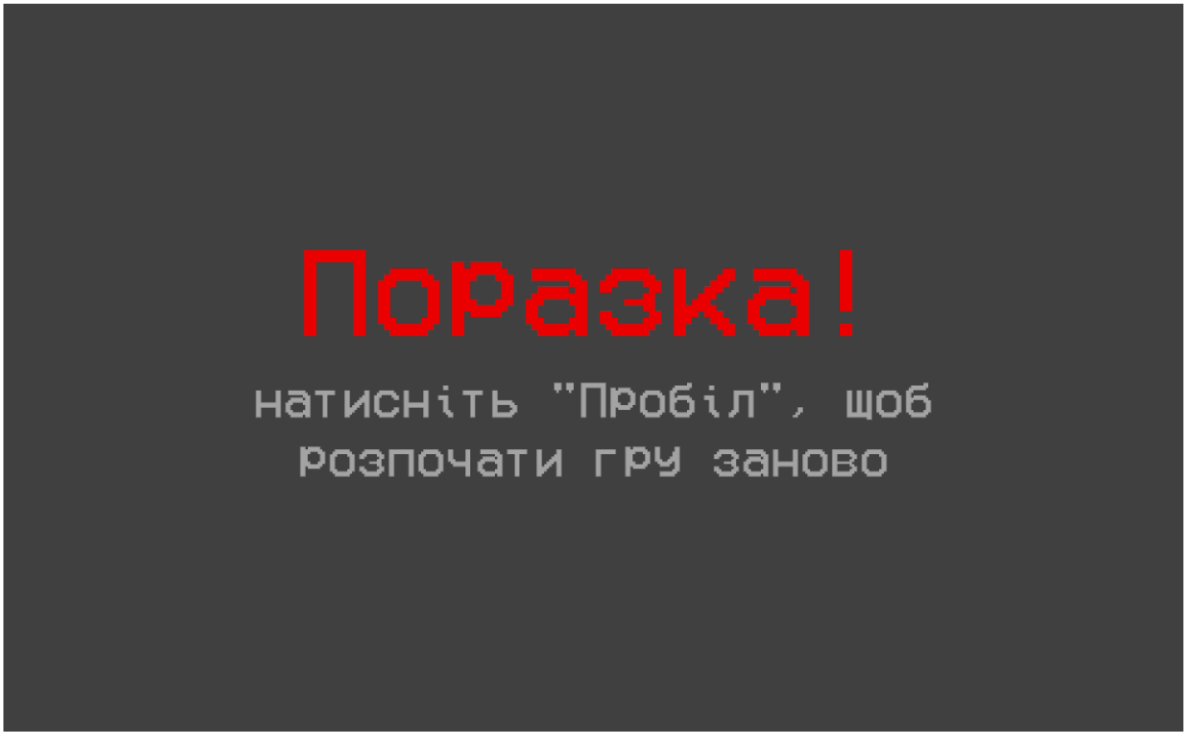

Рисунок 3.11 – Екран поразки

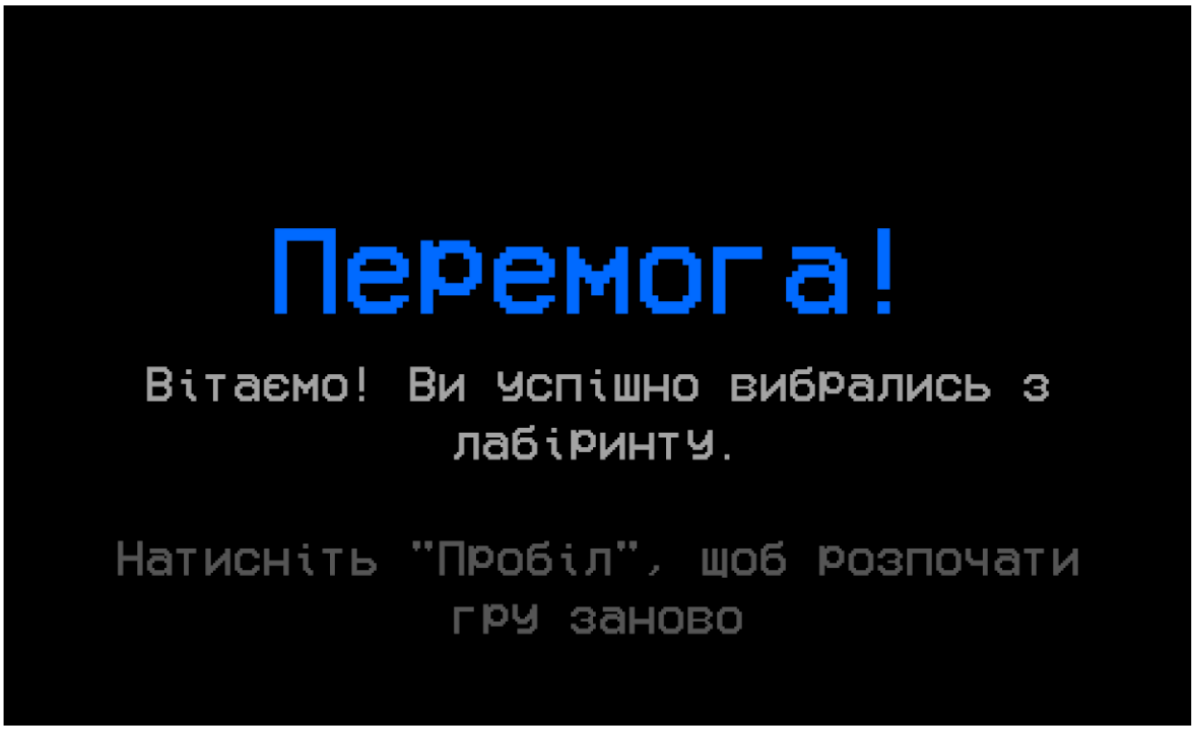

Рисунок 3.12 – Екран перемоги

Підказки для гравця було вирішено організувати у вигляді спливаючих фраз (монологів) головного героя (рис. 3.13, рис. 3.14).

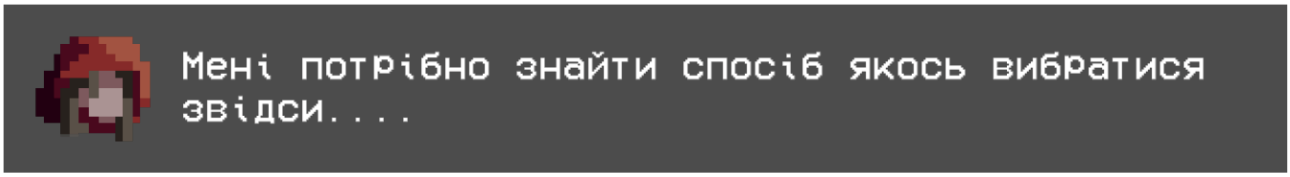

Рисунок 3.13 – «Стартова» фраза головного героя

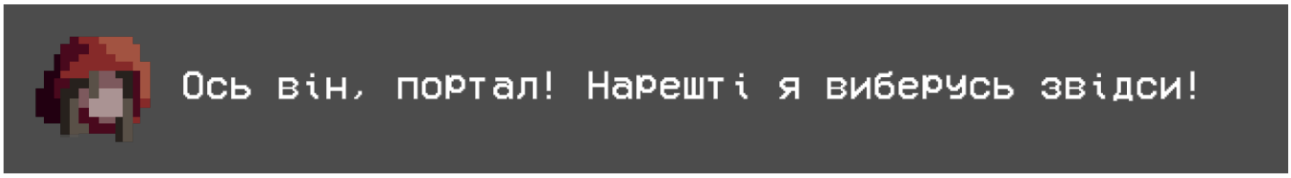

Рисунок 3.14 – Фраза головного героя біля порталу

3.2 Створення класів, об’єктів та обробників ігрових подій

Основні ігрові події будуть описані в файлі index.js (додаток Б). Він визначає різні об'єкти гри, такі як гравець, екрани початку, смерті та перемоги, портали, ворота, ключі та їх розміщення на екрані. Він також містить ряд функцій для взаємодії гравця з грою, таких як виявлення зіткнень, перевірка стану перемоги або поразки, перезавантаження гри, відтворення звуків та інші функції, необхідні для реалізації логіки гри [20].

Деякі змінні, такі як keyBlock, startScreenToggle, musicStart, runSound, deathSound, winSound, showWinPortal, portalSound, win, showGates, keyFound, та showKey, використовуються для збереження стану гри, таких як чи може гравець рухатися, чи відтворювати звуки, чи знайдено ключ тощо.

Код також містить визначення різних об'єктів гри, таких як player, playerShadow, deathScreen, startScreen, winScreen, winPortal, gates, та key, використовуючи клас Sprite, який має різні властивості, такі як позиція, зображення та інші атрибути, необхідні для відображення графічних елементів на екрані [21].

Код також визначає ряд функцій, таких як checkIfRunning (фукція, що відповідає за відтворення звуків кроків під час переміщення головного героя) (рис. 3.15), rectangularCollision (яка перевіряє колізію між будь–якими двома

об’єктами) (рис. 3.16), restart (перезапускає рівень) (рис. 3.17, рис. 3.18), deathDetect (після колізії з противником, відтворює звук поразки та показує екран з відповідним сповіщенням) (рис. 3.19), winDetect (робить все те саме що й функція deathDetect, але перевіряє колізію з фінішом і показує сповіщення про перемогу) (рис. 3.20) та інші.

```
const checkIfRunning = () => {
    if (!keyBlock) {
        let data = Object.entries(keys)
        let isRunning = false
        for (let i = 0; i < data.length - 1; i++) {
            if (data[i][1].pressed) {
                if (!runSound) {
                    audio.Run.play()
                    runSound = true
                    break
                }
                isRunning = true
            }
        }
        if (!isRunning) audio.Run.stop()
    }
}
```

Рисунок 3.15 – Функція checkIfRunning

```
function rectangularCollision({ rectangle1, rectangle2 }) {
    return (
        rectangle1.position.x + rectangle1.width >= rectangle2.position.x &&
        rectangle1.position.x <= rectangle2.position.x + rectangle2.width &&
        rectangle1.position.y <= rectangle2.position.y + rectangle2.height &&
        rectangle1.position.y + rectangle1.height >= rectangle2.position.y
    )
}
```

Рисунок 3.16 – Функція rectangularCollision

```
const restart = () => {
    gsap.to(overlay, {
        opacity: 1,
        onComplete: () => {
            dialogWindows.forEach(dialog => {
                dialog.show = false
                dialog.isShown = false
            })
            Object.keys(showDialogsData).forEach(key => {
                showDialogsData[key] = false
            })
```

Рисунок 3.17 – Функція restart (1/2)

```
            levels["0_2"].init({
                bgPosition: {
                    x: -56,
                    y: -174
                },
                enemiesOffset: {
                    x: 0,
                    y: 0
                }
            })
            player.position.x = 669
            player.position.y = 299
            playerShadow.position.x = player.position.x - 30
            playerShadow.position.y = player.position.y + 44
            offsetBuffer.x = 0
            offsetBuffer.y = 0
            lastDirection = "right"
            player.image = PlayerStandingRightImage
            player.frames.val = 0
            player.frames.elapsed = 0
            keyBlock = false
            player.dead = false
            win = false
            showWinPortal = false
            deathSound = false
            keyFound = false
            gsap.to(overlay, {
                opacity: 0
            })
        }
    })
}
```

Рисунок 3.18 – Функція restart (2/2)

```
const deathDetect = () => {
    audio.Run.stop()
    player.dead = true
    if (!deathSound) {
        audio.Death.play()
        deathSound = true
    }
    gsap.to(player.frames, {
        val: 6,
        onComplete: () => {
            keyBlock = true
        },
        duration: 0.5,
        modifiers: {
            val: function (x) {
                return parseInt(x);
            }
        }
    })
}
```

Рисунок 3.19 – Функція deathDetect

```
const winDetect = () => {
    audio.Run.stop()
    if (!winSound) {
        audio.Win.play()
        winSound = true
    }
    gsap.to(overlay, {
        opacity: 1,
        onComplete: () => {
            keyBlock = true
            win = true
            audio.Map.stop()
            musicStart = false
            gsap.to(overlay, {
                opacity: 0
            })
            portalSound = false
        },
        duration: 1.5
    })
}
```

Рисунок 3.20 – Функція winDetect

Також, у кінці файлу index.js знаходяться обробники подій keydown та keyup на вікні браузера. Вони відповідають за реагування на натискання та відпускання клавіш клавіатури, і встановлюють відповідні значення в об'єкті keys залежно від натиснутих клавіш. Основні дії, які відбуваються: при натисканні клавіші на клавіатурі (подія keydown) змінна startScreenToggle встановлюється в false, якщо musicStart не дорівнює true, то відтворюється звуковий файл audio.Map.play() та musicStart встановлюється в true, та в залежності від натиснутої клавіші, встановлюється значення true в відповідний об'єкт keys (наприклад, для клавіші 'w' або 'ц' встановлюється keys.w.pressed = true). При відпусканні клавіші на клавіатурі (подія keyup) змінна runSound встановлюється в false, в залежності від відпущеної клавіші, встановлюється значення false в відповідний об'єкт keys (наприклад, для клавіші 'w' або 'ц' встановлюється keys.w.pressed = false).

У файлі classes.js описані класи об’єктів, які використовуються у грі. Клас «Sprite» є базовим класом для різноманітних спрайтів у грі. Конструктор класу має багато параметрів, які використовуються для ініціалізації властивостей об'єкту спрайта [22].

Основні властивості об'єкту спрайта включають:

– «position»: об'єкт, що містить координати позиції спрайта на екрані.

– «image»: об'єкт «Image», який представляє зображення спрайта.

– «frames»: об'єкт, який містить інформацію про анімацію спрайта, зокрема максимальну кількість кадрів, поточний кадр та час, що пройшов з моменту останнього оновлення кадру.

– «moving»: логічне значення, яке вказує, чи рухається спрайт.

– «sprites»: об'єкт, який містить зображення для різних напрямків руху спрайта.

– «transitTo»: об'єкт, що містить інформацію про рівень, на який буде відбуватися перехід (властивість для об’єктів міжрівневого переходу).

– «offsetBuffer»: числове значення, яке використовується для розрахунку зсуву позиції спрайта під час руху.

– «bgPosition»: об'єкт, що містить координати позиції фону на екрані.

– «playerPosition»: об'єкт, що містить координати позиції гравця на екрані.

– «spriteType»: рядкове значення, яке вказує тип спрайта.

– «enemiesOffsetData»: об'єкт, який містить дані про зсуви спрайтів ворогів.

– «stepsAmount»: числове значення, яке визначає кількість кроків, які спрайт повинен зробити у визначеному напрямку перед зміною напрямку руху.

– «stepsCount»: числове значення, яке вказує поточну кількість зроблених кроків спрайта.

– «shadow»: логічне значення, яке вказує, чи відкидає об’єкт тінь.

– «enemyType»: рядкове значення, яке вказує вид противника.

– «dead»: логічне значення, яке вказує, чи помер об’єкт.

– «isPortal»: логічне значення, яке вказує, чи об’єкт це портал, чи ні.

Також існують класи Boundary (прозора межа карти, при колізії з якою, головний герой зупиняє рух) (рис. 3.21) та Hitbox (фактично, є рухомим варіантом об’єктів класу Boundary, з прив’язкою до позиції спрайтів противників, при колізії з яким, головний герой помирає) (рис. 3.22, рис. 3.23).

```
class Boundary {
    static height = 51.2
    static width = 51.2
    constructor({ position }) {
        this.position = position
        this.width = 51.2
        this.height = 51.2
    }

    draw() {
        c.fillStyle = 'rgba(255,0,0,0)'
        c.fillRect(this.position.x, this.position.y,
this.width, this.height)
    }
}
```

Рисунок 3.21 – Клас Boundary

```
class Hitbox {
    constructor({ position, steps, movingDirection }) {
        //bee - x+28 y+28
        //skeleton - x+10 y+60
        this.position = position
        this.width = 45
        this.height = 35
        this.movingDirection = movingDirection
        this.stepsAmount = steps
        this.stepsCount = 0
        this.moveForward = true
        this.moveForward = true
    }

    draw() {
        c.fillStyle = 'rgba(255,0,0,0)'
        c.fillRect(this.position.x, this.position.y,
this.width, this.height)

        if (this.moveForward) {
            if (this.stepsCount <= this.stepsAmount) {
```

Рисунок 3.22 – Клас Hitbox (1/2)

```
                if (this.movingDirection == "right") {
                    this.position.x += movingSpeed * mobSpeedMultiplier
                }
                if (this.movingDirection == "left") {
                    this.position.x -= movingSpeed * mobSpeedMultiplier

                }
                if (this.movingDirection == "up") this.position.y -=
movingSpeed * mobSpeedMultiplier
                if (this.movingDirection == "down") this.position.y +=
movingSpeed * mobSpeedMultiplier
                this.stepsCount++
            }
            if (this.stepsCount == this.stepsAmount) {
                this.moveForward = false
                this.stepsCount = 0
            }
        } else {
            if (this.stepsCount <= this.stepsAmount) {
                if (this.movingDirection == "right") {
                    this.position.x -= movingSpeed * mobSpeedMultiplier
                }
                if (this.movingDirection == "left") {
                    this.position.x += movingSpeed * mobSpeedMultiplier
                }
                if (this.movingDirection == "up") this.position.y +=
movingSpeed * mobSpeedMultiplier
                if (this.movingDirection == "down") this.position.y -=
movingSpeed * mobSpeedMultiplier
                this.stepsCount++
            }
            if (this.stepsCount == this.stepsAmount) {
                this.moveForward = true
                this.stepsCount = 0
            }
        }

    }
}
```

Рисунок 3.23 – Клас Hitbox (2/2)

У файлі levels.js описані налаштування рівнів, координати фону, розміщення об’єктів, противників, порталів, тобто код для ініціалізації рівнів гри і розміщення графічних елементів на канвасі. У коді рівні представлені у вигляді двомірного масиву. Кожен рівень представлений у вигляді JavaScript об’єкту (рис. 3.24).

```
"0_4": {
        init: ({ bgPosition, enemiesOffset }) => {
            movingSpeed = 2
            level = "0_4"
            collisionsMap = []
            boundaries = []
            enemies = []
            enemyHitboxes = []
            showWinPortal = true
            if (showDialogsData[3]) {
                dialogWindows[3].isShown = true
            }
            if (!showDialogsData[7]) {
                showDialogsData[7] = true
                dialogWindows[7].show = true
            }
            background = new Sprite({
                position: {
                    x: bgPosition.x,
                    y: bgPosition.y
                },
                imageSrc: "./img/maps/map_0_4.png"
            })
            for (let i = 0; i < collisions_0_4.length; i += 30) {
                collisionsMap.push(collisions_0_4.slice(i, 30 + i))
            }
            collisionsMap.forEach((row, i) => {
                row.forEach((symbol, j) => {
                    if (symbol === 1089) {
                        boundaries.push(new Boundary({
                            position: {
                                x: j * Boundary.width + bgPosition.x,
                                y: i * Boundary.height + bgPosition.y
                            }
                        }))
                    }
                })
            })
            foreground = new Sprite({
                position: {
                    x: bgPosition.x,
                    y: bgPosition.y
                },
                imageSrc: "./img/maps/foreground_0_4.png"
            })
            transitions = [
                new Sprite({
                    position: {
                        x: 0,
                        y: 850,
                    },
                    imageSrc: "./img/transition_horizontal_down.png",
                    transitTo: "0_5",
                    bgPosition: {
                        x: -136,
                        y: -4
                    },
                    offsetBuffer: {
                        x: -25,
                        y: 265
                    },
                    playerPosition: {
                        x: 694,
                        y: 34
                    },
                    enemiesOffsetData: {
                        x: -80,
                        y: 170
                    }
                }),
            ]
            movables = [background, ...boundaries, foreground,
...transitions, ...enemies, ...enemyHitboxes, winPortal]
        }
    },
```

Рисунок 3.24 – Об’єкт рівня

Додатково у файлі levels.js описані наступні змінні:

– canvas: елемент canvas на сторінці HTML, на якому відображається гра.

– c: контекст рендерингу 2D графіки на канвасі.

– background: об'єкт Sprite, що відповідає за відображення фонового зображення рівня гри.

– enemies: масив об'єктів Sprite, що представляють ворожих персонажів на рівні гри.

– collisionsMap: масив, який містить карту зіткнень на рівні.

– movables: масив, який містить всі рухомі об'єкти на рівні гри, такі як фон, перешкоди, вороги та їхні області взаємодії.

– boundaries: масив об'єктів Boundary, які представляють нерухомі межі на рівні.

– enemyHitboxes: масив об'єктів Hitbox, які представляють області взаємодії ворогів на рівні.

– foreground: об'єкт Sprite, що відповідає за відображення переднього плану рівня.

3.3 Тестування та відлагодження гри

Після старту гри, гравець з’являється на точці спавну (рис. 3.25).

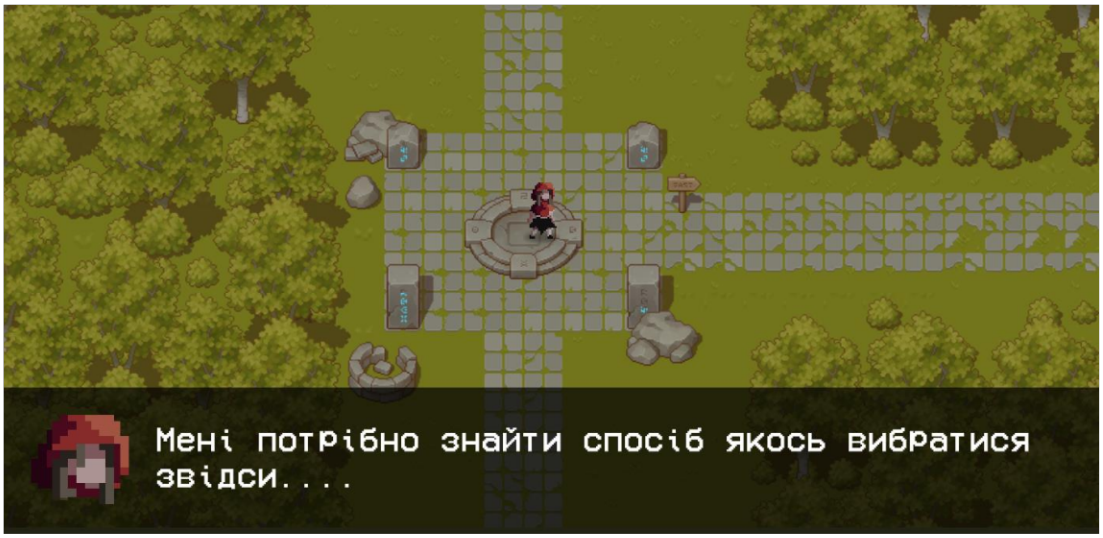

Рисунок 3.25 – Початок гри

Перевірка колізій та анімацій пройшла успішно. Усі персонажі рухаються як потрібно, а при зіткненні з противником з’являється екран поразки (рис. 3.26).

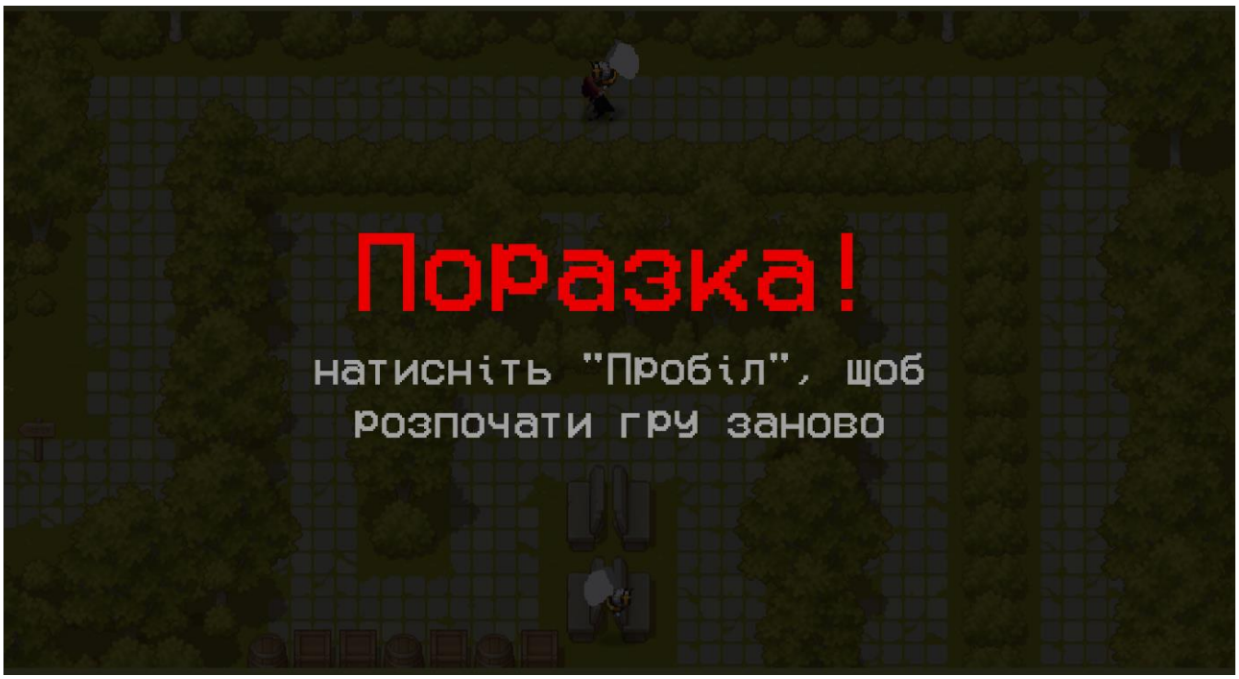

Рисунок 3.26 – Поразка після колізії з ворогом

Взаємодія з елементами оточення теж працює справно. Коли портал попадає в поле зору гравця, на екрані з’являється відповідна фраза (рис 3.27).

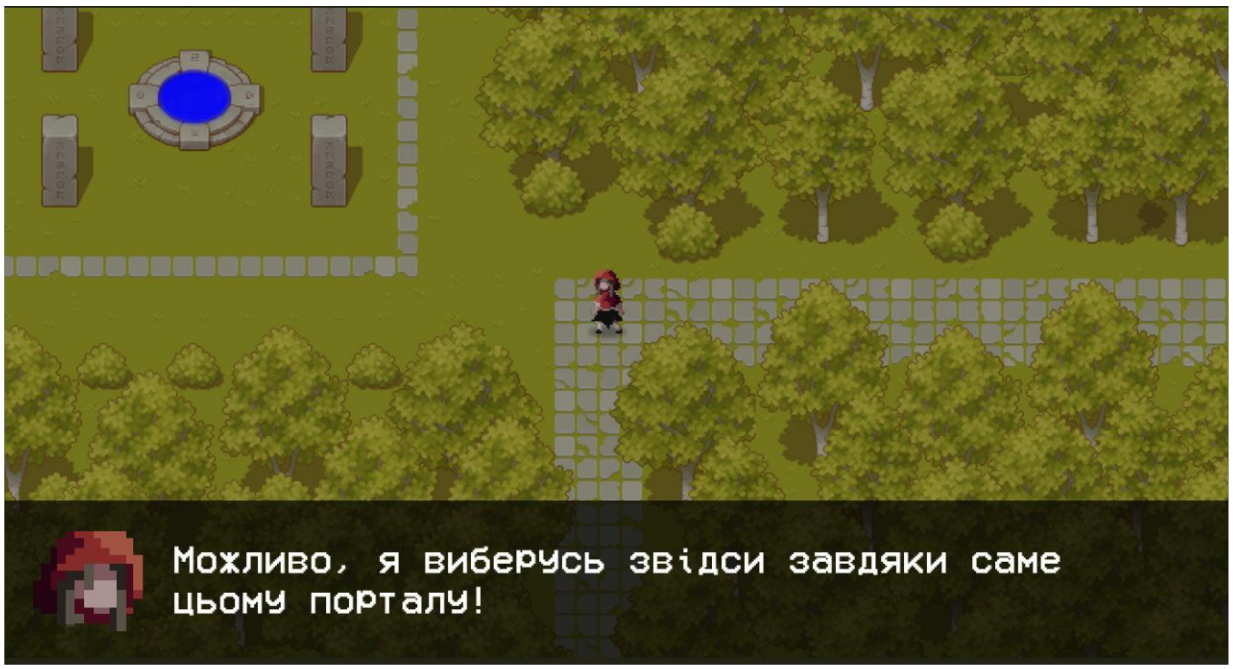

Рисунок 3.27 – Знаходження порталу

Переходи між рівнями теж працюють цілком справно. Після взаємодії з порталом–пасткою гравця перенесло на іншу частину карти (рис. 3.28).

Рисунок 3.28 – Перехід між рівнями за допомогою порталу

При успішному виході з лабіринту теж з’являється відповідне сповіщення (рис. 3.29).

Рисунок 3.29 – Екран перемоги

Під час тестування на іншому пристрої, було виявлено, що при роботі на моніторах з різною частотою оновлення екрану, гра веде себе по різному. Іншими словами, швидкість гри (руху персонажів, анімацій і т.д) напряму залежить від частоти оновлення монітору. Як виявилося, це особливість роботи HTML5 Canvas. Не відомо, чи це технічна помилка, чи було зроблено розробниками навмисно, але це викликало серйозну проблему.

Для вирішення цієї проблеми було вирішено буквально змінювати швидкість гри і залежності від частоти оновлення монітору користувача. Для цього було створено файл fpsMeter.js (рис. 3.30).

```
const times = [];
let fps;
let multiplier;

let playerSpeed
let playerFrameSpeed
let mobSpeedMultiplier

function refreshLoop() {
    window.requestAnimationFrame(() => {
        const now = performance.now();
        while (times.length > 0 && times[0] <= now - 1000) {
            times.shift();
        }
        times.push(now);
        fps = times.length;
        refreshLoop();
    });
    if (fps < 80) {
        playerSpeed = 5.5
        playerFrameSpeed = 5
        mobSpeedMultiplier = 1.1
    } else if (fps > 80 && fps < 100) {
        playerSpeed = 4
        playerFrameSpeed = 8
        mobSpeedMultiplier = 0.8
    } else if (fps > 100) {
        playerSpeed = 3
        playerFrameSpeed = 11
        mobSpeedMultiplier = 0.6
    }
}

refreshLoop();
```

Рисунок 3.30 – Вміст файлу fpsMeter.js

Функція refreshLoop() використовує window.requestAnimationFrame() для запуску циклу оновлення гри на кожному кадрі браузера. Вона також визначає кількість кадрів на секунду, вимірюючи час виконання одного кадру за допомогою performance.now() і зберігаючи ці значення в масиві times.

Потім функція перевіряє поточне значення FPS (у HTML5 Canvas, частота оновлення монітору пристрою фактично є максимальним можливим значенням FPS гри, а так як розроблена гра не вимагає потужних ресурсів комп'ютера, то практично весь час вона працює на максимально можливій кількості кадрів в секунду) і налаштовує різні параметри гри в залежності від цього значення. Наприклад, якщо FPS менше 80, то швидкість гравця, швидкість анімації гравця та множник швидкості ворогів будуть збільшені. Якщо FPS знаходиться в діапазоні від 80 до 100, то ці параметри будуть налаштовані на менші значення. А якщо FPS більше 100, то ці параметри будуть налаштовані на найменші значення.

Це дозволяє грі адаптуватися до продуктивності веб–браузера, забезпечуючи плавну геймплейну динаміку навіть на різних пристроях з різними продуктивністю.

Після реалізації даного рішення, та його імплементації у код класів об'єктів, гра почала працювати стабільно при будь–якій частоті оновлення монітору гравця.

Під час розробки та тестування, інших проблем та несправностей знайдено не було.

3.4 Завершення розробки та завантаження на хостинг

Після завершення розробки та налагодження гри, прийшов час завантажити її на окремий виділений сервер (хостинг). Як згадувалося раніше, була обрана платформа InfinityFree (рис. 3.31).

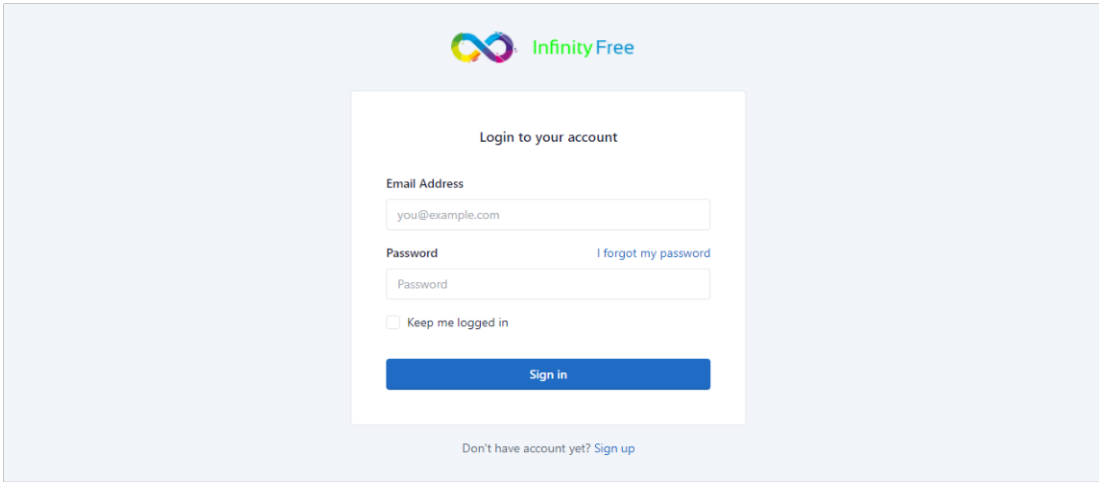

Рисунок 3.31 – Екран логіну до особистого кабінету InfinityFree

Після реєстрації на платформі, в особистому кабінеті необхідно натиснути кнопку «Create Account» (рис. 3.32), ввести назву домену та створити хостинг–аккаунт (рис. 3.33).

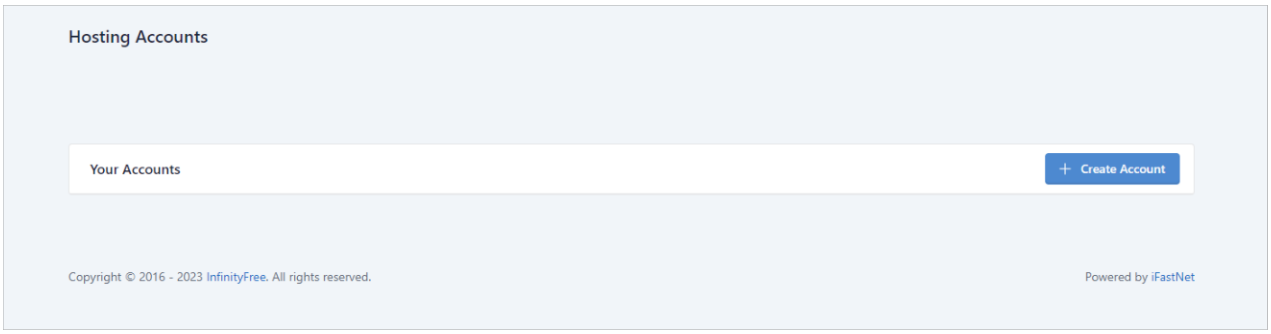

Рисунок 3.32 – Особистий кабінет

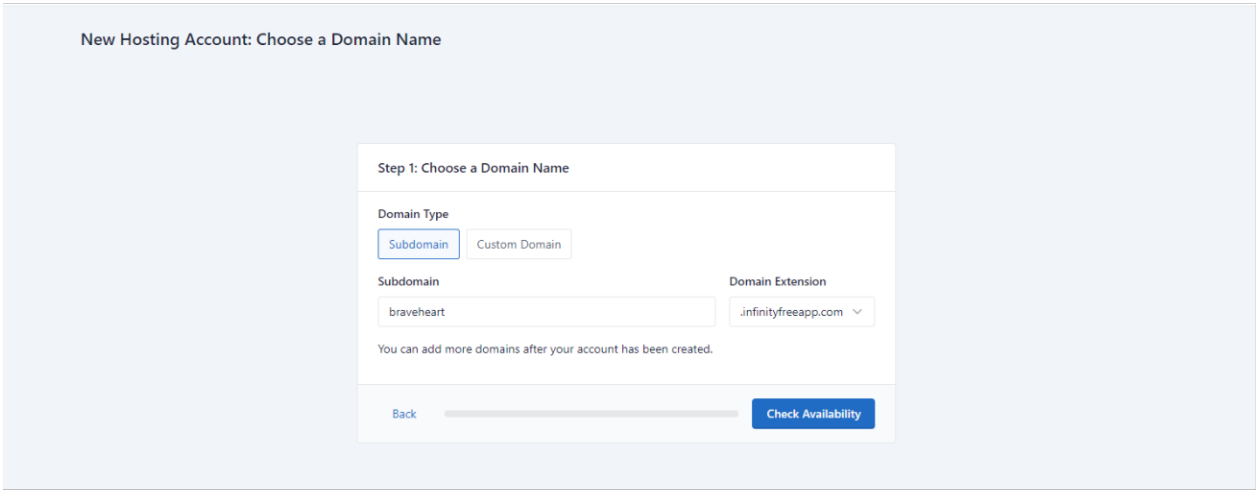

Рисунок 3.33 – Створення домену

Після створення домену, необхідно відкрити меню з інформацією про хостинг–аккаунт (рис. 3.34), та через нього перейти в панель керування (рис. 3.35). Саме в панелі керування ми побачимо дані для налаштування FTP–з'єднання.

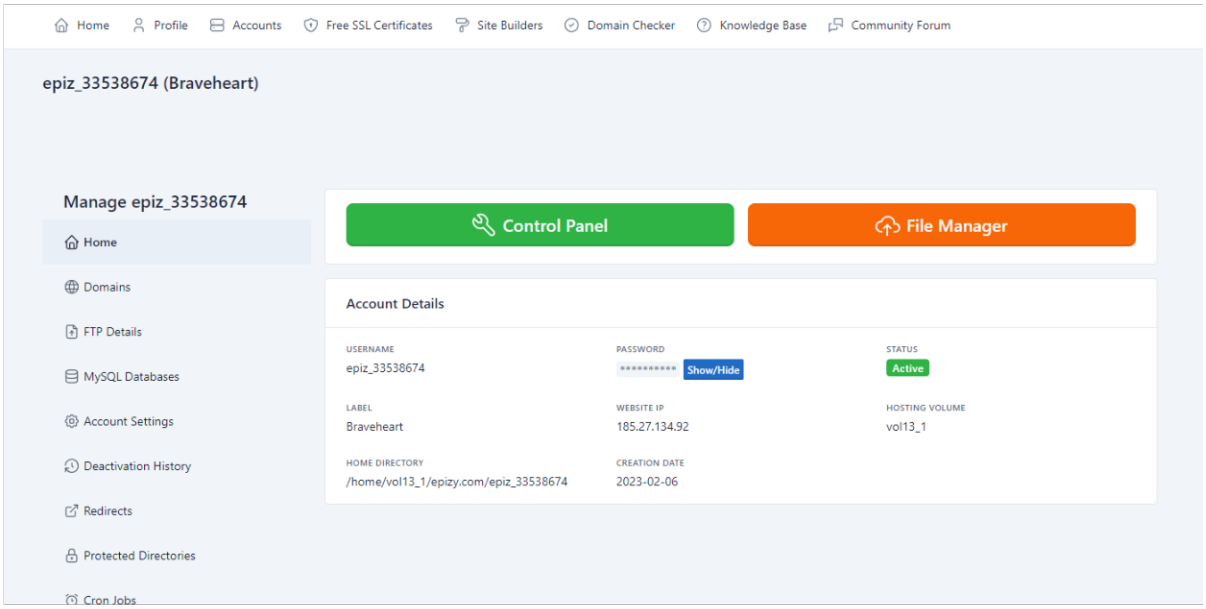

Рисунок 3.34 – Інформація про обліковий запис

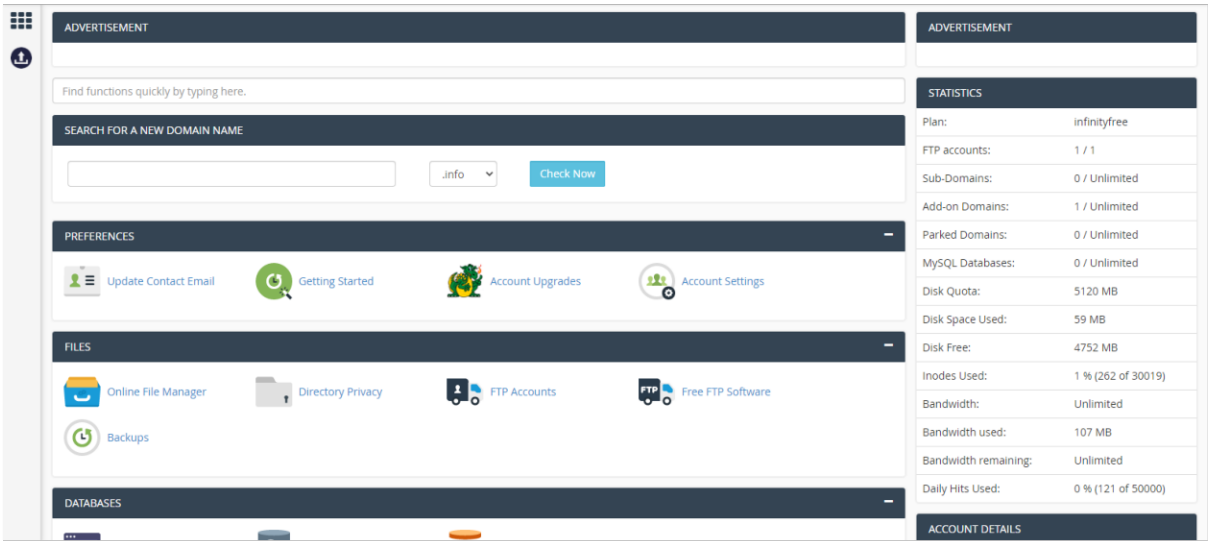

Рисунок 3.35 – Панель керування

Отримані в панелі керування дані для FTP з'єднання ми вставляємо в клієнт FileZilla (рис. 3.36) [23, 24]. Після сповіщення про успішне з'єднання, завантажуємо файли сайту на хостинг. (рис. 3.37)

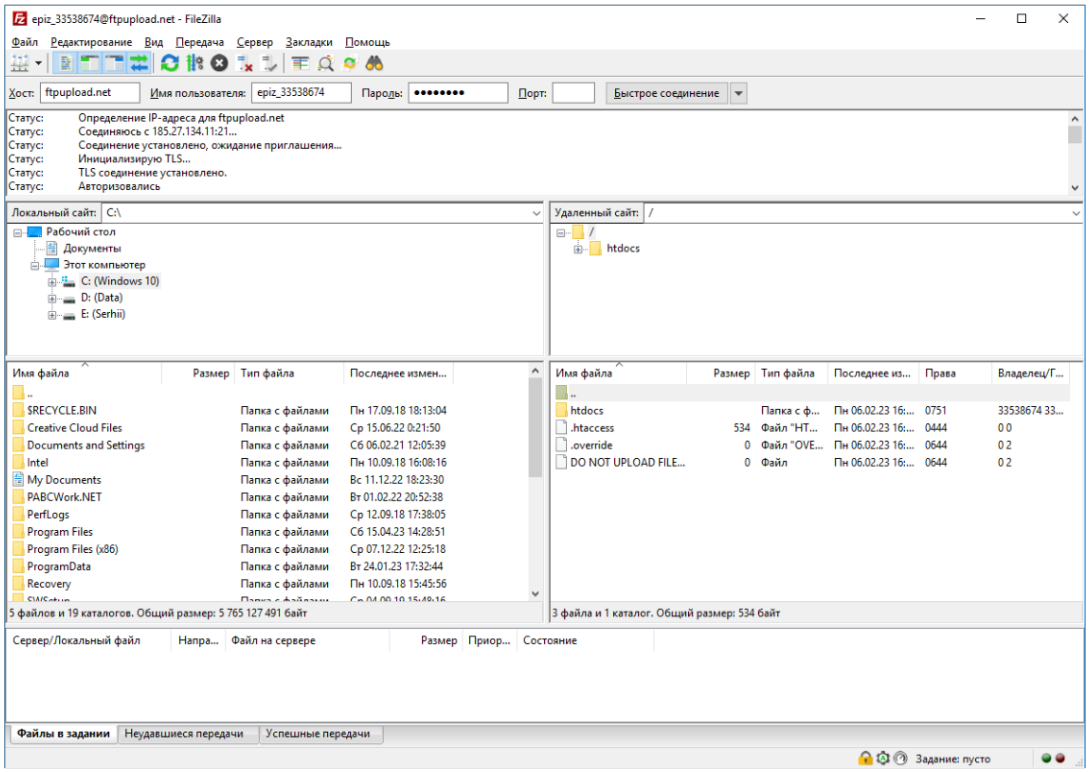

Рисунок 3.36 – Встановлення з’єднання з сервером

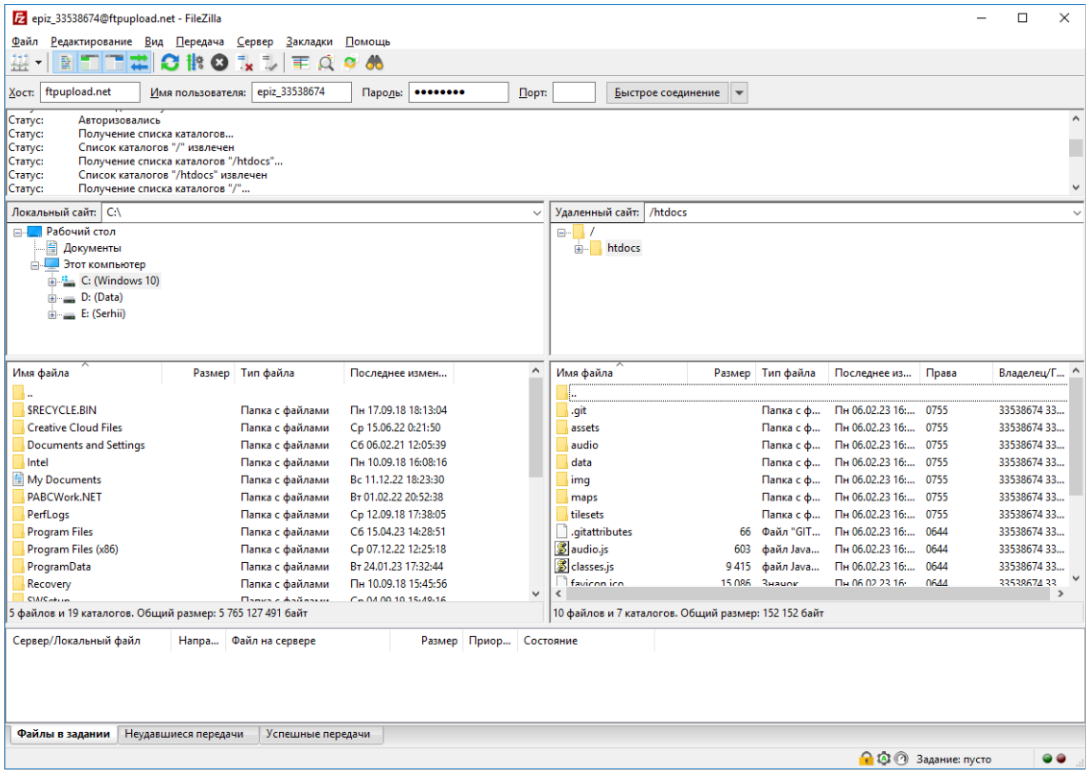

Рисунок 3.37 – Завантаження файлів на сервер

Ось і все, через кілька хвилин після завантаження файлів на сервер, гра стала доступна за посиланням http://breaveheart.infinityfreeapp.com/.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було проведено аналіз інформаційних джерел, пов’язаних зі сферою розробки ігор. Був здійснений порівняльний аналіз середовищ та інструментів для розробки, обґрунтований вибір необхідних засобів розробки, і спроектована структура об’єкту розробки. На основі цього аналізу була розроблена двомірна веб–гра за допомогою JavaScript – однієї з найпоширеніших мов програмування веб–розробки, яка володіє широким спектром можливостей для розробки ігор. Використання JavaScript під час розробки ігор дозволяє створювати веб–ігри, які доступні на різних платформах без необхідності встановлення додаткового програмного забезпечення.

Розробка гри включала підготовку візуальних елементів гри, створення класів, об'єктів та обробників ігрових подій. Також було проведено тестування та відлагодження гри, щоб забезпечити її коректну роботу. Після завершення розробки гра була успішно завантажена на хостинг для забезпечення віддаленого доступу.

Виконання даної роботи дало можливість вивчити особливості розробки ігор з використанням JavaScript, а також вдосконалити навички програмування і роботи з веб–технологіями. Дослідження технологій розробки веб–ігор виявило їх потенціал та можливості у створенні цікавих та привабливих ігрових додатків.

Отже, виконання даної роботи підтвердило актуальність теми розробки ігор за допомогою JavaScript і показало, що ця мова програмування є потужним інструментом для створення веб–ігор з високою якістю графіки та функціональністю. Результати роботи можуть бути використані як основа для подальшого розвитку та вдосконалення ігрових проектів, а також для навчання та ознайомлення зі сферою розробки веб–ігор з використанням JavaScript

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Музика, С. О. Основні етапи створення комп'ютерної гри з використанням WEB технологій. Житомир: ЖДУ ім. Івана Франка, 2022. С. 53–55

2. Phaser. Phaser. URL: https://phaser.io/ (дата звернення: 02.01.2023).

3. CreateJS. CreateJS. URL: https://createjs.com/ (дата звернення: 05.01.2023).

4. Unity Development Platform. Unity. URL: https://unity.com/ (дата звернення: 06.01.2023).

5. Pixi.js. Pixi.js. URL: https://pixijs.com/ (дата звернення: 19.01.2023).

6. Canvas API. MDN Web Docs. URL: https://developer.mozilla.org/en–US/docs/Web/API/Canvas_API (дата звернення: 24.01.2023).

7. Філіпов І. К. Аналіз програмних засобів для обробки зображень. Сучасні тенденції розвитку української науки: матеріали Всеукр. наук. конф.(21–22 липня 2018 р., м. Переяслав–Хмельницький). №5. Переяслав–Хмельницький, 2018. С. 28–31

8. LDtk – 2D Level Editor. LDtk. URL: https://ldtk.io/ (дата звернення: 26.01.2023).

9. Tiled | Flexible level editor. Tiled. URL: https://www.mapeditor.org/ (дата звернення:).

10. Максименюк М. Р., Арсенюк І. Р. Обгрунтування доцільності розробки програмного рушія для візуалізації інтерактивної графіки. Diss. ВНТУ, 2022. 3 с.

11. Новицька Є. О. Основні характеристики, інструменти програми GIMP. Математичні методи, моделі та інформаційні технології у науці, освіті, економіці, виробництві: збірник тез І Всеукраїнської науково–практичної Інтернет–конференції з проблем вищої освіти і науки, м. Маріуполь, 26 квітня 2019 р./Маріупольський державний університет; уклад. Шабельник ТВ, Дяченко ОФ, Морозова АО. Маріуполь: МДУ, 2019. 246 с.

12. Visual Studio Code. Visual Studio. URL: https://code.visualstudio.com/ (дата звернення: 05.02.2023).

13. Верницький І. Р. Розробка браузерної онлайн гри «Over world» на базі Java Script із використанням Node is та Phaser : дипломна робота магістра за спеціальністю «121 — інженерія програмного забезпечення». Тернопіль: ТНТУ, 2020. 57 с.

14. Недоля Д. А. Розробка web–додатку для проектування двовимірних форм : кваліфікаційна робота магістра спеціальності 121 "Інженерія програмного забезпечення". Запоріжжя : ЗНУ, 2020. 43 с.

15. Free Web Hosting. InfinityFree. URL: https://www.infinityfree.net/ (дата звернення:).

16. Filezilla – Free FTP Solution. Filezilla. URL: https://filezilla–project.org/ (дата звернення: 09.02.2023).

17. FTP. Вікіпедія. URL: https://uk.wikipedia.org/wiki/FTP (дата звернення: 12.02.2023).

18. Бербега, Володимир Олегович. Графічний рушій із підтримкою користувацького вводу. КПІ ім. Ігоря Сікорського, 2021. 78 с.

19. Пух Д. Веб–застосунок гра–головоломка. Інформаційно–комунікаційні технології в освіті, (10). 2023. С. 35–65.

20. Каневський М. В, Захарченко С. М. Особливості монетизації сучасних web–ігор. Збірник матеріалів XII Міжнародної науково–практичної конференції "Інтернет–освіта–наука" (ІОН–2020). Вінниця: ВНТУ, 2020. С. 183–185

21. Бодю К. О., Булгакова О. Ф. Аналіз реалізації принципів об’єктно–орієнтованого програмування у мові JavaScript в порівнянні з традиційними підходами. Тернопіль, 2019. С. 13–15

22. Швець Х. І., Горяшин А. С. Інформаційні технології класовий підхід в мові програмування JavaScript. Комп’ютерні технології обробки даних. Вінниця, 2022. С. 173–176.

23. Що таке FTP і для чого він потрібен? HOSTIQ. URL: https://hostiq.ua/wiki/ukr/ftp/ (дата звернення: 18.02.2023).

24. Що таке FTP Client і як ним користуватися? Ukraine. URL: https://www.ukraine.com.ua/uk/blog/hosting_ukraine/chto–takoe–ftp–client–i–kak–im–polzovatsya.html (дата звернення: 03.03.2023).

ДОДАТКИ

Додаток А

Лістинг коду файлу index.html

```
<title>Braveheart</title>
<link rel="shortcut icon" type="image/x-icon" href="/img/favicon/favicon.ico">
<style>
    body {
        background-color: rgb(35, 35, 20);
        display: flex;
        justify-content: center;
        align-items: center;
        margin: 0;
    }
</style>

<body>
    <canvas></canvas>
    <div style="display: flex; flex-direction: column;">
        <p style="color: white; font-size: large;" id="level"></p>
        <p style="color: white; font-size: large;" id="fpsMeter"></p>
        <p style="color: white; font-size: large;" id="playerSpeed"></p>
        <p style="color: white; font-size: large;" id="coordinates"></p>
        <p style="color: white; font-size: large;" id="offsetBuffer"></p>
        <p style="color: white; font-size: large;" id="bgPosition"></p>
    </div>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/howler/2.2.3/howler.min.js"
        integrity="sha512-
6+YN/9o9BWrk6wSfGxQGpt3EUK6XeHi6yeHV+TYD2GR0Sj/cggRpXr1BrAQf0as6XslxomMUxXp2vIl+
fv0QRA=="
        crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.11.4/gsap.min.js"
        integrity="sha512-
f8mwTB+Bs8a5c46DEm7HQLcJuHMBaH/UFlcgyetMqqkvTcYg4g5VXsYR71b3qC82lZytjNYvBj2pf0Ve
kA9/FQ=="
        crossorigin="anonymous" referrerpolicy="no-referrer"></script>
    <script src="fpsMeter.js"></script>
    <script src="audio.js"></script>
    <script src="./data/collisions.js"></script>
    <script src="levels.js"></script>
    <script src="classes.js"></script>
    <script src="index.js"></script>
</body>
```

Додаток Б

Лістинг коду файлу index.js

```
let keyBlock = false
let startScreenToggle = true
let musicStart = false
let runSound = false
let deathSound = false
let winSound = false
let showWinPortal = false
let portalSound = false
let win = false
let showGates = false
let keyFound = false
let showKey = false

const player = new Sprite({
    position: {
        x: canvas.width / 2 - 31,
        y: canvas.height / 2 - 71
    },
    frames: {
        max: 7
    },
    imageSrc: "./img/player/standing_right.png",
    sprites: {
        left: PlayerStandingLeftImage,
        right: PlayerStandingRightImage,
        runningLeft: PlayerRunningLeftImage,
        runningRight: PlayerRunningRightImage,
        deadLeft: PlayerDeadLeftImage,
        deadRight: PlayerDeadRightImage
    }
})

const playerShadow = new Sprite({
    position: {
        x: player.position.x - 30,
        y: player.position.y + 44
    },
    imageSrc: "./img/player/shadow.png",
})

const deathScreen = new Sprite({
    position: {
        x: 0,
        y: 0
    },
    imageSrc: "./img/deathScreen.png",
})

const startScreen = new Sprite({
    position: {
        x: 0,
        y: 0
    },
    imageSrc: "./img/startScreen.png",
})

const winScreen = new Sprite({
    position: {
```

```
        x: 0,
        y: 0
    },
    imageSrc: "./img/winScreen.png",
})

const winPortal = new Sprite({
    position: {
        x: 648,
        y: 113
    },
    imageSrc: "./img/portal_red.png",
})

const gates = new Sprite({
    position: {
        x: -57,
        y: -338
    },
    imageSrc: "./img/gates.png",
})

const key = new Sprite({
    position: {
        x: 670,
        y: 640
    },
    imageSrc: "./img/key.png",
    enemyType: "key",
    shadow: true
})

levels[level].init({
    bgPosition: {
        x: -56,
        y: -174
    },
    enemiesOffset: {
        x: 0,
        y: 0
    }
})

const keys = {
    w: { pressed: false },
    a: { pressed: false },
    s: { pressed: false },
    d: { pressed: false },
    space: { pressed: false },
}

const checkIfRunning = () => {
    if (!keyBlock) {
        let data = Object.entries(keys)
        let isRunning = false
        for (let i = 0; i < data.length - 1; i++) {
            if (data[i][1].pressed) {
                if (!runSound) {
                    audio.Run.play()
                    runSound = true
                    break
                }
                isRunning = true
            }
```

```
        }
        if (!isRunning) audio.Run.stop()
    }
}

const playerMovables = [player, playerShadow]

function rectangularCollision({ rectangle1, rectangle2 }) {
    return (
        rectangle1.position.x + rectangle1.width >= rectangle2.position.x &&
        rectangle1.position.x <= rectangle2.position.x + rectangle2.width &&
        rectangle1.position.y <= rectangle2.position.y + rectangle2.height &&
        rectangle1.position.y + rectangle1.height >= rectangle2.position.y
    )
}

const restart = () => {
    gsap.to(overlay, {
        opacity: 1,
        onComplete: () => {
            dialogWindows.forEach(dialog => {
                dialog.show = false
                dialog.isShown = false
            })
            Object.keys(showDialogsData).forEach(key => {
                showDialogsData[key] = false
            })
            levels["0_2"].init({
                bgPosition: {
                    x: -56,
                    y: -174
                },
                enemiesOffset: {
                    x: 0,
                    y: 0
                }
            })
            player.position.x = 669
            player.position.y = 299
            playerShadow.position.x = player.position.x - 30
            playerShadow.position.y = player.position.y + 44
            offsetBuffer.x = 0
            offsetBuffer.y = 0
            lastDirection = "right"
            player.image = PlayerStandingRightImage
            player.frames.val = 0
            player.frames.elapsed = 0
            keyBlock = false
            player.dead = false
            win = false
            showWinPortal = false
            deathSound = false
            keyFound = false
            gsap.to(overlay, {
                opacity: 0
            })
        }
    })
}

const deathDetect = () => {
    audio.Run.stop()
    player.dead = true
    if (!deathSound) {
```

```
        audio.Death.play()
        deathSound = true
    }
    gsap.to(player.frames, {
        val: 6,
        onComplete: () => {
            keyBlock = true
        },
        duration: 0.5,
        modifiers: {
            val: function (x) {
                return parseInt(x);
            }
        }
    })
}

const winDetect = () => {
    audio.Run.stop()
    if (!winSound) {
        audio.Win.play()
        winSound = true
    }
    gsap.to(overlay, {
        opacity: 1,
        onComplete: () => {
            keyBlock = true
            win = true
            audio.Map.stop()
            musicStart = false
            gsap.to(overlay, {
                opacity: 0
            })
            portalSound = false
        },
        duration: 1.5
    })
}

const changeLevel = (transition) => {
    if (transition.isPortal) {
        if (!portalSound) {
            audio.Portal.play()
            portalSound = true
        }
    }
    gsap.to(overlay, {
        opacity: 1,
        onComplete: () => {
            levels[transition.transitTo].init({
                bgPosition: {
                    x: transition.bgPosition.x,
                    y: transition.bgPosition.y
                },
                enemiesOffset: {
                    x: transition.enemiesOffsetData.x,
                    y: transition.enemiesOffsetData.y
                }
            })
            player.position.x = transition.playerPosition.x
            player.position.y = transition.playerPosition.y
            playerShadow.position.x = transition.playerPosition.x - 30
            playerShadow.position.y = transition.playerPosition.y + 44
            offsetBuffer.x = transition.offsetBuffer.x
```

```
            offsetBuffer.y = transition.offsetBuffer.y
            gsap.to(overlay, {
                opacity: 0
            })
            portalSound = false
        }
    })
}


let lastDirection = "right"
function animate() {
    window.requestAnimationFrame(animate)
    background.draw()
    boundaries.forEach(boundary => {
        boundary.draw()
    })
    transitions.forEach(transition => {
        transition.draw()
    })
    if (showWinPortal) winPortal.draw()
    if (showGates) gates.draw()
    if (showKey) key.draw()
    playerShadow.draw()
    player.draw()
    enemies.forEach(enemy => {
        enemy.draw()
        enemy.moving = true
    })
    enemyHitboxes.forEach(hitbox => {
        hitbox.draw()
    })
    foreground.draw()
    if (!startScreenToggle) {
        if (!keyBlock) {
            dialogWindows.forEach(dialog => {
                dialog.draw()
            })
        }
    }

    checkIfRunning()
    if (keyBlock) {
        if (!win) deathScreen.draw()
    }
    if (startScreenToggle) startScreen.draw()
    if (win) winScreen.draw()

    c.save()
    c.globalAlpha = overlay.opacity
    c.fillStyle = 'black'
    c.fillRect(0, 0, canvas.width, canvas.height)
    c.restore()

    let moving = true
    player.moving = false

    //Detect hitbox collision
    for (let i = 0; i < enemyHitboxes.length; i++) {
        const hitbox = enemyHitboxes[i]
        if (rectangularCollision({
            rectangle1: player,
            rectangle2: {
                ...hitbox, position: {
```

```
                    x: hitbox.position.x,
                    y: hitbox.position.y,
                }
            }
        })) {
            lastDirection == "right" ? player.image = player.sprites.deadRight :
player.image = player.sprites.deadLeft
            deathDetect()
        }
    }

    if (!keyBlock) {
        if (showWinPortal) {
            if (rectangularCollision({
                rectangle1: player,
                rectangle2: winPortal
            })) {
                winDetect()
            }
        }
        if (!keyFound) {
            if (level == "1_3") {
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: key
                })) {
                    audio.KeyFound.play()
                    keyFound = true
                    showKey = false
                }
            }
        }
        if (keys.w.pressed) {
            player.moving = true
            if (!player.dead) lastDirection == "right" ? player.image =
player.sprites.runningRight : player.image = player.sprites.runningLeft
            for (let i = 0; i < boundaries.length; i++) {
                const boundary = boundaries[i]
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: {
                        ...boundary, position: {
                            x: boundary.position.x,
                            y: boundary.position.y + playerSpeed,
                        }
                    }
                })) {
                    moving = false
                    break
                }
            }
            for (let i = 0; i < transitions.length; i++) {
                const transition = transitions[i]
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: {
                        ...transition, position: {
                            x: transition.position.x,
                            y: transition.position.y,
                        }
                    }
                })) {
                    changeLevel(transition)
                    break
```

```
            }
        }
        if (moving) {
            if (background.position.y > -playerSpeed) {
                moving = false
                playerMovables.forEach(playerMovable => {
                    playerMovable.position.y -= playerSpeed
                })
                offsetBuffer.y += playerSpeed
            } else {
                if (offsetBuffer.y < -10 || offsetBuffer.y > 10) {
                    playerMovables.forEach(playerMovable => {
                        playerMovable.position.y -= playerSpeed
                    })
                    offsetBuffer.y += playerSpeed
                } else {
                    movables.forEach(movable => {
                        movable.position.y += playerSpeed
                    })
                }
            }
        }
    }
    if (keys.a.pressed) {
        player.moving = true
        if (!player.dead) player.image = player.sprites.runningLeft
        lastDirection = "left"
        for (let i = 0; i < boundaries.length; i++) {
            const boundary = boundaries[i]
            if (rectangularCollision({
                rectangle1: player,
                rectangle2: {
                    ...boundary, position: {
                        x: boundary.position.x + playerSpeed,
                        y: boundary.position.y,
                    }
                }
            })) {
                moving = false
                break
            }
        }
        for (let i = 0; i < transitions.length; i++) {
            const transition = transitions[i]
            if (rectangularCollision({
                rectangle1: player,
                rectangle2: {
                    ...transition, position: {
                        x: transition.position.x,
                        y: transition.position.y,
                    }
                }
            })) {
                changeLevel(transition)
                break
            }
        }
        if (moving) {
            if (background.position.x > -playerSpeed) {
                moving = false
                playerMovables.forEach(playerMovable => {
                    playerMovable.position.x -= playerSpeed
                })
                offsetBuffer.x += playerSpeed
```

```
                } else {
                    if (offsetBuffer.x < -10 || offsetBuffer.x > 10) {
                        playerMovables.forEach(playerMovable => {
                            playerMovable.position.x -= playerSpeed
                        })
                        offsetBuffer.x += playerSpeed
                    } else {
                        movables.forEach(movable => {
                            movable.position.x += playerSpeed
                        })
                    }
                }
            }
        }
        if (keys.s.pressed) {
            player.moving = true
            if (!player.dead) lastDirection == "right" ? player.image =
player.sprites.runningRight : player.image = player.sprites.runningLeft
            for (let i = 0; i < boundaries.length; i++) {
                const boundary = boundaries[i]
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: {
                        ...boundary, position: {
                            x: boundary.position.x,
                            y: boundary.position.y - playerSpeed,
                        }
                    }
                })) {
                    moving = false
                    break
                }
            }
            for (let i = 0; i < transitions.length; i++) {
                const transition = transitions[i]
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: {
                        ...transition, position: {
                            x: transition.position.x,
                            y: transition.position.y,
                        }
                    }
                })) {
                    changeLevel(transition)
                    break
                }
            }
            if (moving) {
                if (background.position.y < (canvas.height - background.height +
playerSpeed)) {
                    moving = false
                    playerMovables.forEach(playerMovable => {
                        playerMovable.position.y += playerSpeed
                    })
                    offsetBuffer.y -= playerSpeed
                } else {
                    if (offsetBuffer.y < -10 || offsetBuffer.y > 10) {
                        playerMovables.forEach(playerMovable => {
                            playerMovable.position.y += playerSpeed
                        })
                        offsetBuffer.y -= playerSpeed
                    } else {
                        movables.forEach(movable => {
```

```
                            movable.position.y -= playerSpeed
                        })
                    }
                }
            }
        }
        if (keys.d.pressed) {
            player.moving = true
            if (!player.dead) player.image = player.sprites.runningRight
            lastDirection = "right"
            for (let i = 0; i < boundaries.length; i++) {
                const boundary = boundaries[i]
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: {
                        ...boundary, position: {
                            x: boundary.position.x - playerSpeed,
                            y: boundary.position.y,
                        }
                    }
                })) {
                    moving = false
                    break
                }
            }
            for (let i = 0; i < transitions.length; i++) {
                const transition = transitions[i]
                if (rectangularCollision({
                    rectangle1: player,
                    rectangle2: {
                        ...transition, position: {
                            x: transition.position.x,
                            y: transition.position.y,
                        }
                    }
                })) {
                    changeLevel(transition)
                    break
                }
            }
            if (moving) {
                if (background.position.x < (canvas.width - background.width +
playerSpeed)) {
                    moving = false
                    playerMovables.forEach(playerMovable => {
                        playerMovable.position.x += playerSpeed
                    })
                    offsetBuffer.x -= playerSpeed
                } else {
                    if (offsetBuffer.x < -10 || offsetBuffer.x > 10) {
                        playerMovables.forEach(playerMovable => {
                            playerMovable.position.x += playerSpeed
                        })
                        offsetBuffer.x -= playerSpeed
                    } else {
                        movables.forEach(movable => {
                            movable.position.x -= playerSpeed
                        })
                    }
                }
            }
        }
    } else {
        if (keys.space.pressed) {
```

```
            restart()
        }
    }

    if (!player.dead) {
        if (!player.moving) {
            if (lastDirection == "left") {
                player.image = player.sprites.left
            } else if (lastDirection == "right") {
                player.image = player.sprites.right
            }
        }
    }
    //document.getElementById('level').innerHTML = `${level}`;
    //document.getElementById('playerSpeed').innerHTML = `${playerSpeed}`;
    //document.getElementById('fpsMeter').innerHTML = `FPS: ${fps}`;
    //document.getElementById('coordinates').innerHTML = `PLAYER:
${player.position.x}, ${player.position.y}`;
    //document.getElementById('offsetBuffer').innerHTML = `offsetBUFFER:
${offsetBuffer.x}, ${offsetBuffer.y}`;
    //document.getElementById('bgPosition').innerHTML = `BG:
${background.position.x}, ${background.position.y}`;
}

animate()

window.addEventListener('keydown', (e) => {
    startScreenToggle = false
    if (!musicStart) {
        audio.Map.play()
        musicStart = true
    }
    switch (e.key) {
        case 'w':
            keys.w.pressed = true
            break;
        case 'ц':
            keys.w.pressed = true
            break;
        case 'a':
            keys.a.pressed = true
            break;
        case 'ф':
            keys.a.pressed = true
            break;
        case 's':
            keys.s.pressed = true
            break;
        case 'i':
            keys.s.pressed = true
            break;
        case 'd':
            keys.d.pressed = true
            break;
        case 'в':
            keys.d.pressed = true
            break;
        case 'ArrowUp':
            keys.w.pressed = true
            break;
        case 'ArrowLeft':
            keys.a.pressed = true
            break;
        case 'ArrowDown':
```

```
            keys.s.pressed = true
            break;
        case 'ArrowRight':
            keys.d.pressed = true
            break;
        case ' ':
            keys.space.pressed = true
            break;
    }
})

window.addEventListener('keyup', (e) => {
    runSound = false
    switch (e.key) {
        case 'w':
            keys.w.pressed = false
            break;
        case 'ц':
            keys.w.pressed = false
            break;
        case 'a':
            keys.a.pressed = false
            break;
        case 'ф':
            keys.a.pressed = false
            break;
        case 's':
            keys.s.pressed = false
            break;
        case 'i':
            keys.s.pressed = false
            break;
        case 'd':
            keys.d.pressed = false
            break;
        case 'в':
            keys.d.pressed = false
            break;
        case 'ArrowUp':
            keys.w.pressed = false
            break;
        case 'ArrowLeft':
            keys.a.pressed = false
            break;
        case 'ArrowDown':
            keys.s.pressed = false
            break;
        case 'ArrowRight':
            keys.d.pressed = false
            break;
        case ' ':
            keys.space.pressed = false
            break;
    }
})
```