

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

«Розробка 2D платформера»

Виконав студент групи ЗПЗ-21-1

ПАВЛИК Ілля Валентинович

Керівник роботи:

ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:

ВАКАЛЮК Тетяна Анатоліївна

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ПАВЛИК Ілля Валентинович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«Розробка 2D платформера»**,

затверджена Наказом закладу вищої освіти від «14» лютого 2025 р., №62/с

Вихідні дані для роботи: Об'єктом дослідження процес розробки програмного продукту в цілому. Предметом дослідження є методи та засоби проектування й реалізації 2D ігрових механік за допомогою сучасних фреймворків. Дослідження спрямоване на створення повнофункціонального 2D платформера з типовою ігровою механікою.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М	02.03.2025р.	02.03.2025р.
2	Лобанчикова Н.М	04.04.2025р.	04.04.2025р.
3	Лобанчикова Н.М	05.05.2025р.	05.05.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-застосунку 2D-платформера «Endless Jumper» та пояснювальної записки. Пояснювальна записка містить 82 сторінки, 6 ілюстрацій і 4 таблиці.

Метою роботи є розробка кросплатформенного веб-додатку з реалізацією базових ігрових механік (плавне керування персонажем, фізично коректні стрибки, стрільба, система бонусів) Методи, використані під час створення проекту, включають програмування на JavaScript із використанням HTML5 Canvas і CSS3, застосування архітектурного патерну MVC, організацію даних через Firebase (Firestore, Authentication), хостинг і CI/CD на Microsoft Azure за допомогою Docker, а також поєднання автоматизованого тестування за допомогою Jest і мануальних тестових сценаріїв.

Результати розробки полягають у створенні інтерактивного, адаптивного та масштабованого прототипу гри з динамічною відображенням таблиці лідерів у реальному часі, високою продуктивністю на різних пристроях і налагодженою інфраструктурою розгортання.

КЛЮЧОВІ СЛОВА: JavaScript, HTML5 Canvas, CSS3, MVC, Firebase, Azure, Docker, Jest, 2D-платформер, CI/CD.

					ІПЗ.КР.Б-121-25-ІПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Розробка 2D платформера Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		Павлик І.В						
Керівник		Лобанчикова Н.М					3	76
Рецензен.		Вакалюк Т.А				Житомирська політехніка, група ЗІПЗ-21-1		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's thesis qualification work consists of a web application of a 2D platformer "Endless Jumper" and an explanatory note. The explanatory note contains 82 pages, 6 illustrations and 4 tables.

The work method is the development of a cross-platform web application with the implementation of basic game mechanisms (smooth character control, physically correct jumping, shooting, bonus system)

The methods used during the creation of the project include JavaScript programming using HTML5 Canvas and CSS3, application of the MVC architectural pattern, data organization via Firebase (Firestore, Authentication), hosting and CI/CD on Microsoft Azure using Docker, as well as connecting automated testing using Jest and manual test scripts.

The development results consist in creating an interactive, adaptive and scalable game prototype with dynamic display of real-time leaderboards, high performance on different devices and a well-established deployment infrastructure.

KEYWORDS: JavaScript, HTML5 Canvas, CSS3, MVC, Firebase, Azure, Docker, Jest, 2D platformer, CI/CD.

					ИП3.КР.Б-121-25-ИЗ	
						4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ТА ТЕХНОЛОГІЇ РОЗРОБКИ 2D ПЛАТФОРМЕРІВ.....	9
1.1. Аналіз аналогів програмного продукту.....	9
1.2. Аналіз підходів та технологій для реалізації 2D платформерів.....	12
1.3. Обґрунтування вибору засобів реалізації.....	14
1.4. Постановка задач розробки	18
Висновки до першого розділу	19
РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ 2D ПЛАТФОРМЕРА	21
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	21
2.2. Створення персонажів та сценаріїв руху: моделі, методи, алгоритми.....	25
2.3. Алгоритми створення платформи та середовища	26
2.4 Створення бази даних системи та функціоналу 2D платформера.....	29
Висновки до другого розділу	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ОСОБЛИВОСТІ НАЛАШТУВАННЯ (РОЗГОРТАННЯ, РОБОТИ) ПЛАТФОРМЕРА	49
3.1 Порядок встановлення та налаштування параметрів 2D платформера.....	49
3.2 Інтерфейс та особливості використання.....	51
3.3 Тестування роботи 2D платформера.....	57
Висновки до третього розділу	62
ВИСНОВКИ	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	65
ДОДАТКИ	68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

IDE – Integrated Development Enviroment

JSON – JavaScript Object Notation

MVVM – Model View ViewModel

SDK – Software Development Kit – набір для розробки програмного забезпечення

SSD – Solid State Drive

БД – база даних

ПЗ – програмне забезпечення

					ІПЗ.КР.Б-121-25-ПЗ	
						6

ВСТУП

В даній роботі обрано тему «Розробка 2D платформера» як актуальну сферу сучасної індустрії програмного забезпечення, що поєднує в собі художню та технічну складові та має широкий попит як серед геймерів, так і серед освітніх проєктів. Попит на доступні, проте привабливі ігрові рішення зростає з розвитком мобільних та веб-платформ, а простота 2D-графіки дозволяє швидко реалізувати прототипи і тестувати ігрову механіку. Крім того, засвоєння методів розробки двовимірних платформерів слугує надійною базою для подальшого опанування більш складних тривимірних середовищ.

Метою даної роботи є створення повнофункціонального 2D платформера з типовою ігровою механікою (переміщення, стрибки, збори предметів, взаємодія з перешкодами), а також обґрунтування вибору архітектури, інструментальних засобів і алгоритмів. В межах цієї мети розглядаються такі завдання: аналіз аналогів і вимог користувачів, проєктування структури системи, розробка бази даних та реалізація ключових модулів ігрового рушія, оформлення інтерфейсу та проведення тестування.

Об'єктом дослідження виступає процес розробки програмного продукту в цілому, а **предметом** – методи та засоби проєктування й реалізації 2D ігрових механік за допомогою сучасних фреймворків.

Для досягнення поставленої мети необхідним є вирішення наступних інженерно-технічних задач:

- провести аналіз аналогічних програмних продуктів для виявлення перваг та недоліків;
- провести аналіз підходів та технологій для реалізації 2 D платформера;
- провести моделювання та проєктування програмного забезпечення;
- провести програмну реалізацію 2D платформера.

При виконанні поставлених завдань слід застосовуються методи системного та порівняльного аналізу, UML-моделювання, а також

експериментальна перевірка на основі тестових сценаріїв і автоматизованих засобів відлагодження коду.

Практичне значення результатів роботи полягає у створенні готового програмного рішення, яке може бути використано як освітній інструмент для вивчення основ геймдеву, а також як база для подальшого розширення функціоналу або портування на інші платформи. Інформаційною базою слугували офіційні документації обраного рушія, наукові публікації з питань розробки ігор і досвід розробників-практиків, викладений у фахових статтях та технічних блогах.

Структура роботи включає вступ, три основні розділи (аналіз вимог, проєктування, інтерфейс і тестування), висновки, перелік використаних джерел і додатки, що містять діаграми, ілюстративні матеріали та фрагменти коду. Пояснювальна записка містить 76 сторінок, 6 ілюстрацій і 4 таблиці.

РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ТА ТЕХНОЛОГІЙ РОЗРОБКИ 2D ПЛАТФОРМЕРІВ

В даному розділі буде здійснено аналіз ключових аспектів розробки та впровадження 2D-платформера. Спочатку буде визначено завдання та окреслено функціональні і нефункціональні вимоги до системи, після чого проведено порівняльний огляд існуючих аналогів на ринку. Далі обґрунтовано вибір архітектурного патерну та розглянуто сильні й слабкі сторони альтернативних підходів, а також підібрано оптимальний набір технологічних засобів і визначено мінімальні апаратні вимоги. Такий комплексний підхід дозволяє закласти теоретичну основу для подальшої розробки, тестування й впровадження прототипу гри.

1.1. Аналіз аналогів програмного продукту

Жанр нескінченних вертикальних платформерів здобув значну популярність насамперед завдяки простій, але водночас захоплюючій механіці: гравець постійно рухається вгору, долаючи нові перепони. Ця ідея виявилася особливо вдалою для мобільних платформ: інтуїтивне керування через нахил пристрою або прості дотики стимулює повторні спроби, адже геймплей швидкий і дає відчуття прогресу навіть під час нетривалих ігрових сесій [1,2]. У межах кваліфікаційної роботи важливо не лише описати механіку, але й обґрунтувати вибір реалізації елементів керування, бонусів та дизайну, спираючись на успішні приклади.

Першим і найвідомішим представником цього жанру є «Doodle Jump», випущена у 2009 році [3]. У «Doodle Jump» персонаж постійно стрибає вгору, а гравець, нахиляючи пристрій вліво чи вправо, керує переміщенням і може стріляти у ворогів через тап по екрану. Гра вирізняється багатством типів платформ: звичайні, рухомі, що зникають, вибухові, а також наявністю різних ворогів і бонусів (пружини, реактивні ранці тощо), які з'являються несподіван. Візуально «Doodle Jump» оформлена як рукописний малюнок на зошиті в

клітинку, що створює впізнаваний стиль. Для «Endless Jumper» запозичується сама суть вертикального руху та ідея уникнення або знищення ворогів за допомогою стрільби, але при цьому реалізується керування через екранні кнопки або клавіатуру. Такий підхід робить гру доступною як на мобільних пристроях, так і на настільних системах, позбуваючись залежності від акселерометра, яка є невід’ємною в «Doodle Jump», але може бути менш зручною для ПК-версії та на деяких мобільних пристроях.

Наступним популярним прикладом є «Mega Jump», яка суттєво зміщує акцент із точного позиціювання на платформі до збору монет та бонусів із метою отримати прискорення «Mega Jump!», що дозволяє стрімко пролетіти вгору, іноді знищуючи перешкоди на шляху [4]. У «Mega Jump» ключова увага приділена саме системі бонусів і накопиченню ресурсів. У «Endless Jumper» також передбачено бонусну систему (наприклад, «Super Jump» або «Rapid Fire»), але ці елементи виконують радше тактичну функцію для подолання певних складних ділянок або специфічних комбінацій ворогів і платформ. Якщо в «Mega Jump» бонуси є центром ігрового процесу та стимулом для постійного накопичення й використання з метою максимального стрибка, то в «Endless Jumper» вони мають допоміжний характер: гравець вирішує, коли доцільно витратити накопичений бонус, щоби пройти конкретну ділянку, зберігаючи баланс між ризиком і винагородою. Такий підхід додає глибини стратегії та підсилює відчуття контролю над ситуацією, натомість у «Mega Jump» іноді може виникати відчуття хаотичності через сильний акцент на прискоренні.

Хоча «Helix Jump» не є класичним вертикальним платформером, його механіка падіння через обертову вежу має спільні риси з жанром із точки зору вимог до реакції гравця й простоти управління [5]. У «Helix Jump» гравець керує не самим рухом вгору, а обертанням структури так, щоб кулька падала вниз, уникаючи заборонених секторів. Гра відзначається мінімалістичним 3D-дизайном і тим, що замість прямого переміщення персонажа змінюється середовище.

Порівняння характеристик наведено в таблиці 1.1

Таблиця 1.1

Порівняння аналогів з розробленим 2D платформером

Характеристика	Endless Jumper	Doodle Jump	Mega Jump	Helix Jump
Геймплей	Нескінченний підйом вгору по платформах.	Нескінченний підйом вгору по платформах.	Нескінченний підйом, акцент на зборі монет для "мега-стрибка".	Нескінченне падіння вниз через обертів платформи.
Підтримувані операційні системи	Веб (ПК та мобільні пристрої через браузер).	Переважно мобільні платформи (iOS, Android).	Переважно мобільні платформи (iOS, Android).	Переважно мобільні платформи (iOS, Android).
Управління	Клавіатура (ПК) та екранні кнопки (мобільні).	Акселерометр (нахил пристрою) та дотик до екрана (стрільба).	Акселерометр (нахил пристрою).	Дотик та перетягування (обертання вежі).
Бойова система	Присутня. Гравець може стріляти для знищення ворогів.	Присутня. Гравець може стріляти для знищення ворогів.	Відсутня. Ворогів/перешкод переважно уникають або знищують бонусом.	Відсутня. Ігровий процес побудований на уникненні перешкод.
Система бонусів	Присутня (прискорення стрільби, посилений стрибок).	Присутня (пружини, реактивні ранці, батути).	Центральна механіка (активація "мега-стрибка" та інші покращення).	Відсутня або мінімальна.
Візуальний стиль	Мінімалістичний 2D, векторна графіка (SVG).	Унікальний стиль "малюнків на папері в клітинку".	Яскравий та барвистий мультяшний 2D стиль.	Мінімалістичний 3D стиль.
Технологічний стек	HTML5 Canvas, JavaScript, Firebase.	Нативна розробка (Objective-C/Swift, Java/Kotlin).	Нативна розробка.	Ігровий рушій (ймовірно, Unity).
Змагальний елемент	Онлайн таблиця лідерів.	Глобальна та локальна таблиці лідерів.	Таблиці лідерів, розблокування персонажів.	Система рівнів, таблиці лідерів.

Як видно з таблиці 1.1, "Endless Jumper" є більше класичним представником жанру, поєднуючи в собі механіки, випробувані в "Doodle Jump", зокрема вертикальний геймплей та елементи стрільби. Водночас проєкт має ключову відмінність та перевагу – він реалізований як веб-додаток, що робить його універсально доступним на будь-якому пристрої з браузером, на відміну від суто мобільних аналогів. Схема управління адаптована як для ПК, так і для мобільних пристроїв, що розширює потенційну аудиторію. Технологічний стек на основі JavaScript та Firebase є сучасним та гнучким рішенням для веб-ігор, що дозволяє легко інтегрувати соціальні функції, такі як таблиця лідерів.

1.2. Аналіз підходів та технологій для реалізації 2D платформерів

Правильний вибір архітектури є фундаментальним рішенням при розробці будь-якого програмного продукту, а для ігрових додатків він набуває особливої ваги [6]. Архітектура визначає внутрішню структуру коду, способи взаємодії його компонентів та закладає основи для майбутньої масштабованості, гнучкості та можливості підтримки проєкту. Для 2D-платформера, де одночасно відбувається обробка вводу користувача, розрахунок фізики, оновлення стану ігрових об'єктів та їх візуалізація, структурований підхід є критично важливим. Розглянемо декілька провідних архітектурних патернів, щоб оцінити їхні переваги та недоліки в контексті нашого завдання [7].

В сучасній розробці, особливо в геймдеві, домінує кілька ключових підходів. Entity-Component-System (ECS) є одним із найпотужніших та найпродуктивніших патернів [8]. Його головна ідея полягає у відмові від глибокої ієрархії успадкування на користь композиції. Ігровий об'єкт, або Сутність (Entity), є лише унікальним ідентифікатором. Усі його характеристики та дані зберігаються в незалежних блоках, що називаються Компонентами (Components), як-от PositionComponent чи PhysicsComponent. Уся логіка зосереджена в Системах (Systems), які оперують лише тими сутностями, що мають необхідний набір компонентів. Це забезпечує неймовірну гнучкість і

високу продуктивність завдяки оптимізованому доступу до даних, але вимагає більшої дисципліни та може бути складним для освоєння на початковому етапі.

Класичні патерни, що походять із розробки інтерфейсів користувача, також знайшли широке застосування в іграх. Серед них виділяється тріада MVC, MVP та MVVM.

Model-View-Presenter (MVP) є еволюцією MVC, спрямованою на ще більше розділення відповідальності [9]. В цьому патерні Вигляд (View) стає максимально пасивним. Він лише відображає дані та перенаправляє абсолютно всі події користувача (натискання клавіш, кліки) до Презентера (Presenter). Саме Презентер обробляє цей ввід, запитує дані з Моделі (Model), форматує їх і передає назад у вигляд для відображення. Таким чином, Вигляд нічого не знає про Модель, а вся логіка взаємодії зосереджена в Презентері. Це робить систему надзвичайно зручною для тестування, оскільки Вигляд можна легко замінити заглушкою.

Model-View-ViewModel (MVVM) є ще одним кроком у розвитку цієї ідеї, особливо популярним у сучасних вебфреймворках. Його ключовою особливістю є механізм прив'язки даних (data binding) [10]. Вигляд (View) напряму "підписаний" на Модель Вигляду (ViewModel). ViewModel – це спеціальний клас, що отримує дані з Моделі (Model) і представляє їх у зручному для Вигляду форматі. Коли дані у ViewModel змінюються, Вигляд оновлюється автоматично без додаткового коду. І навпаки, дії користувача у Вигляді можуть викликати команди у ViewModel, яка вже змінює Модель. Це значно зменшує кількість коду для синхронізації, але реалізація механізму прив'язки даних у чистому JavaScript може бути невиправдано складною, а неявний потік даних іноді ускладнює налагодження.

Після аналізу різних підходів для розробки 2D-платформера на базі JavaScript та HTML було обрано архітектуру Model-View-Controller (MVC). Цей патерн пропонує оптимальний баланс між структурованістю, простотою

реалізації та контролем над потоком даних, що є ідеальним для проєкту такого типу та масштабу.

В контексті нашого проєкту компоненти MVC реалізовані наступним чином. Модель (Model) є ядром гри, що містить всю логіку та дані: фізику світу, стан гравця та ворогів, структуру рівнів. Вона повністю незалежна від інших частин системи. Вигляд (View) – це візуальна репрезентація Моделі, представлена елементом canvas. Його єдина функція – отримувати дані з Моделі та відображати їх на екрані в кожному кадрі. Контролер (Controller) виступає сполучною ланкою, що слухає події вводу від користувача (натискання клавіш) і передає відповідні команди Моделі для оновлення її стану.

Патерни MVP та MVVM, хоч і пропонують кращі можливості для тестування та роботи зі складними інтерфейсами, створюють додаткові рівні абстракції [11]. Для гри на canvas, де Вигляд по суті є єдиним об'єктом, що оновлюється в кожному кадрі, прямий та чіткий потік даних "Ввід -> Контролер -> Модель -> Вигляд" є більш прозорим і легким для налагодження, ніж опосередкована взаємодія через Презентер чи автоматична прив'язка даних. MVC забезпечує достатнє розділення відповідальності для чистоти коду, не обтяжуючи проєкт зайвою складністю.

Таким чином, архітектура MVC надала нашому проєкту міцний фундамент, забезпечивши організованість коду, легкість тестування та зрозумілий потік даних, що є критично важливим для успішної розробки 2D-платформера.

1.3. Обґрунтування вибору засобів реалізації

JavaScript – це високорівнева та динамічна мова програмування, що є фундаментальною технологією [12] Всесвітньої павутини та використовується для створення інтерактивних вебсайтів. У нашому проєкті JavaScript виступає як основний інструмент для реалізації всієї клієнтської логіки гри. Завдяки його гнучкості та потужним можливостям, зокрема стандарту ES6+, були реалізовані

ігровий цикл, фізична модель, механіка взаємодії об'єктів та управління станом додатку [13]. Використання нативного JavaScript без сторонніх фреймворків дозволило досягти максимальної продуктивності та мінімального часу завантаження гри, що є критично важливим для забезпечення позитивного користувацького досвіду.

HTML5 та CSS3 є основними технологіями для створення структури та візуального оформлення вебдодатків. HTML5 надає семантичну розмітку для всього інтерфейсу, а його ключовий елемент, ``<canvas>``, став полотном для рендерингу всієї ігрової сцени. Це дозволило створити власний графічний рушій, повністю контролюючи процес візуалізації [14]. CSS3, у свою чергу, використовується для стилізації всіх неігрових елементів інтерфейсу, таких як меню, кнопки, інформаційні панелі та модальні вікна, забезпечуючи їх адаптивність та сучасний вигляд на різних пристроях та розмірах екранів [15].

Firebase – це платформа від Google, що надає набір інструментів для швидкої розробки потужних бекенд-сервісів (Backend-as-a-Service) [16]. У проєкті були використані два ключових сервіси Firebase. Firestore – це гнучка NoSQL база даних, яка застосовується для реалізації таблиці лідерів у реальному часі. Firebase Authentication забезпечує просту та безпечну автентифікацію користувачів, зокрема анонімний вхід, що дозволяє гравцям зберігати свої рекорди без необхідності створювати повноцінний обліковий запис [17]. Використання Firebase значно прискорило розробку, дозволивши зосередитись на клієнтській логіці гри, делегувавши складні бекенд-задачі надійній хмарній інфраструктурі.

Microsoft Azure – це глобальна хмарна платформа, обрана для розгортання та хостингу ігрового додатку [18]. Завдяки сервісу Azure Static Web Apps, розгортання статичного сайту, що складається з HTML, CSS та JavaScript файлів, стає простим, безпечним та масштабованим процесом. Azure забезпечує високу доступність та швидкість завантаження для користувачів по всьому світу завдяки інтегрованій мережі доставки контенту (CDN). Платформа також надає потужні

інструменти для моніторингу та управління додатком, що гарантує його стабільну роботу.

Для розробки проєкту використовувалося інтегроване середовище розробки (IDE), адаптоване для вебтехнологій. Хоча розробка на JavaScript не вимагає специфічної IDE, як Xcode для Swift, робота велася у Visual Studio Code – сучасному редакторі коду від Microsoft. VS Code є надзвичайно популярним у веб-спільноті завдяки своїй легкості, гнучкості та величезній екосистемі розширень. Інтегрований термінал, підтримка Git, інструменти для налагодження та статичного аналізу коду значно спрощують та прискорюють процес розробки.

Оскільки проєкт розроблено на базі вебтехнологій, вимоги до апаратного забезпечення для кінцевих користувачів є мінімальними. Додаток може працювати на будь-якому персональному комп'ютері, ноутбукі чи мобільному пристрої під управлінням сучасної операційної системи. Єдиною вимогою є наявність оновленого веббраузера з підтримкою стандартів HTML5 та JavaScript ES6. Усі вимоги до серверної інфраструктури повністю покриваються хмарними платформами Firebase та Microsoft Azure.

Використання JavaScript, HTML5, CSS3, Firebase та Azure створює потужний, гнучкий та економічно ефективний технологічний стек. Цей набір інструментів дозволив розробити кросплатформенну гру, доступну широкому колу користувачів без необхідності встановлення, та забезпечив надійну інфраструктуру для її функціонування та подальшого розвитку.

Сучасні підходи до створення 2D ігор можна умовно розділити на дві основні категорії: використання спеціалізованих ігрових рушіїв та розробка з використанням веб-технологій або бібліотек. Кожен із цих підходів має свої унікальні переваги, недоліки та сфери застосування [19].

Першим і найбільш поширеним підходом є використання готових ігрових рушіїв. Це комплексні програмні середовища, що надають розробникам готовий набір інструментів, значно прискорюючи процес створення гри. Вони містять

вбудовані системи для рендерингу графіки, обробки фізики, відтворення звуку та управління анімаціями.

Серед найвідоміших рушіїв виділяється Unity [20]. Це універсальне середовище для розробки як 2D, так і 3D ігор, що використовує мову програмування C#. Для 2D платформерів Unity пропонує потужні інструменти, такі як 2D Physics Engine, редактор анімацій та система Tilemap для створення рівнів. Його величезна спільнота та великий магазин ассетів роблять його чудовим вибором для проєктів будь-якої складності.

Альтернативою є Godot Engine, безкоштовний рушій з відкритим вихідним кодом. Він має інтуїтивну архітектуру, засновану на концепції сцен та вузлів, та підтримує власну, схожу на Python, мову GDScript, а також C#. Його легкість та відсутність ліцензійних відрахувань роблять Godot привабливим для незалежних розробників [21].

Для тих, хто фокусується виключно на 2D, існує GameMaker Studio. Цей рушій історично орієнтований на двовимірні ігри, що робить його простішим для освоєння. Він використовує власну скриптову мову GML (GameMaker Language), яка дозволяє швидко створювати прототипи та повноцінні ігри [22].

Другий підхід полягає у розробці з використанням веб-технологій. З розвитком стандартів HTML5 та JavaScript веб-браузери перетворилися на повноцінну ігрову платформу, що забезпечує максимальну кросплатформеність, адже гра може бути запущена на будь-якому пристрої без попереднього встановлення [23].

Цей шлях можна пройти, створюючи гру "з нуля" за допомогою нативного JavaScript та HTML5 Canvas. Елемент canvas служить "полотном" для рендерингу 2D графіки, а на JavaScript реалізується вся ігрова логіка: ігровий цикл, фізика, анімації тощо. Такий підхід дає розробнику повний контроль над кодом, але вимагає значно більше часу та глибоких знань. Саме цей підхід був обраний для даного дипломного проєкту, що дозволило продемонструвати глибоке розуміння фундаментальних принципів розробки ігор.

Щоб спростити розробку на веб-технологіях, існують спеціалізовані бібліотеки та фреймворки. Phaser.js є одним із найпопулярніших фреймворків, що надає готові рішення для роботи з графікою, фізикою та анімаціями [24]. Іншим інструментом є Pixi.js – надзвичайно швидка 2D рендерингова бібліотека, яка використовує апаратне прискорення WebGL, але залишає реалізацію ігрової логіки на розсуд розробника [25].

Вибір конкретного підходу та технології залежить від цілей проєкту, досвіду команди та вимог до продуктивності. Для даної роботи був обраний шлях розробки на нативному JavaScript, що дозволило максимально глибоко зануритись у всі технічні аспекти створення 2D платформи.

1.4. Постановка задач розробки

У сучасному середовищі програмної інженерії створення 2D-платформи є надзвичайно актуальною задачею з огляду на стрімке зростання попиту на інтерактивні розважальні та освітні продукти [1]. Жанр двовимірних платформерів поєднує в собі високу сприйнятливості аудиторією і одночасно відносну простоту реалізації базових механік: руху персонажа, стрибків, гравітації та взаємодії з об'єктами середовища. Це дозволяє у відносно короткі терміни створювати діючі прототипи, перевіряти ітеративно концепції та швидко адаптувати гру під різні платформи – від десктопних OS до мобільних пристроїв і веб-браузерів[2]. Водночас така доступність не знімає необхідності глибокого обґрунтування вибору архітектури системи, інструментальних засобів та підходів до організації коду, адже якісний продукт має бути масштабованим, ефективним з точки зору продуктивності та легко підтримуваним у майбутньому.

Головна проблема, яку покликана вирішити ця робота, полягає в комплексному проєктуванні та побудові 2D-платформи з типовим набором ігрових механік – плавне управління персонажем, фізично коректні стрибки, система збору предметів і обробка зіткнень із перешкодами – при забезпеченні

модульності, читабельності коду та можливості швидкого прототипування. Для цього необхідно розробити єдиний підхід до опису даних гри (рівнів, властивостей об'єктів, параметрів фізики й налаштувань візуальних та звукових ресурсів), визначити критерії вибору рушія чи фреймворка (наприклад, Unity 2D, Godot Engine і Phaser), а також закласти основу для впровадження тестів ключових сценаріїв взаємодії гравця з оточенням. Важливо забезпечити відокремленість ігрової логіки від презентаційного шару, інтегрувати патерни проєктування (MVC, ECS, фабрики ресурсів тощо) та визначити способи управління пам'яттю й оптимізації продуктивності на різних платформах.

У рамках постановки задачі передбачається виконати низку досліджень і аналітичних процедур для формалізації функціональних і нефункціональних вимог: вивчити існуючі аналоги на ринку й оцінити їх сильні й слабкі сторони, скласти технічне завдання з чітко визначеними критеріями успішності (FPS, час відповіді на введення, плавність анімації), а також окреслити межі дослідження – виключити з проєкту мережеві режими, тривимірну візуалізацію та складні системи штучного інтелекту. Така деталізація дасть змогу чітко уявити обсяг майбутньої реалізації, визначити необхідні інструменти та ресурси (апаратні вимоги, версії бібліотек, система контролю версій тощо) і сформулювати план тестування й валідації готового прототипу відповідно до загальних вимог дипломної роботи.

Висновки до першого розділу

У підсумку першого розділу було чітко сформульовано ціль створення 2D-платформера: від опису базових ігрових механік і модульної організації коду до визначення функціональних і нефункціональних вимог із контрольованими межами проєкту. Проведений аналіз аналогів – від класичних «Doodle Jump» і «Mega Jump» до експериментальних «Helix Jump» – дав змогу виокремити сильні сторони та потенційні недоліки кожного підходу й інтегрувати в «Endless

Jumpreg» найдоцільніші елементи: гібридну механіку стрибків і стрільби, тактичні бонуси та кросплатформені інтерфейсні рішення.

Обґрунтування вибору MVC-архітектури підтвердило необхідність зрозумілої й легко тестованої структури, а порівняльний аналіз ECS, MVP та MVVM дозволив уникнути зайвої складності й зосередитися на прозорому потоці даних «Ввід → Контролер → Модель → Вигляд». Вибір стеку JavaScript/HTML5/Canvas, підтриманого Firebase для бекенду та Azure для хостингу, забезпечить гнучкість, високу продуктивність і швидке розгортання прототипу. Такий методичний підхід закладає надійну основу для реалізації, тестування та подальшого масштабування гри відповідно до вимог дипломної роботи.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ 2D ПЛАТФОРМЕРА

У цьому розділі ми переходимо від теоретичного аналізу до практичної реалізації, детально розглядаючи процес проектування 2D платформи "Endless Jumper". Тут буде описано ключові рішення, структура основних ігрових компонентів, таких як гравець, платформи та вороги, а також проектування бази даних для зберігання таблиці рекордів. Цей розділ слугує технічним кресленням, що визначає логіку та взаємодію всіх елементів гри перед їхньою безпосередньою програмною реалізацією.

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Аналіз вимог включає визначення та деталізацію функціональних і нефункціональних вимог до програмного забезпечення, оцінку часу реакції системи на дії користувачів, специфікацію можливих сценаріїв використання, а також розробку загальної схеми елементів системи з описом їхньої структури, взаємодії та методів [25]. Цей процес дозволяє чітко зрозуміти потреби користувачів та технічні характеристики, необхідні для створення ефективного та зручного програмного продукту [26].

Для розробленого 2D платформи було визначено наступні функціональні вимоги:

Гравець повинен мати можливість рухатись вліво та вправо, а також стріляти для знищення ворогів. Також гравець повинен стрибати вгору по платформах, що постійно генеруються.

Система повинна відстежувати та відображати поточний рахунок гравця, який збільшується з висотою підйому та за знищення ворогів, а також здоров'я, яке зменшується при контакті з ворогами.

Додаток повинен мати чітко визначені стани: головне меню, ігровий процес, екран завершення гри (Game Over) та екран таблиці лідерів. У грі повинні генеруватися вороги, контактування з якими завдає шкоди гравцеві. Гравець може підбирати бонуси, що тимчасово покращують його характеристики (наприклад, швидкість стрільби або сила стрибка). Система повинна дозволяти гравцеві після завершення гри ввести своє ім'я та зберегти результат у загальну таблицю лідерів. Інші гравці повинні мати можливість переглянути 10 найкращих результатів. Для збереження результатів не потрібна повна реєстрація, оскільки, система використовує анонімний вхід через сервіс Firebase.

Також для розробленого додатку було визначено наступні нефункціональні вимоги:

Додаток повинен коректно працювати в усіх сучасних веб-браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) на настільних та мобільних операційних системах, а інтерфейс користувача має бути англійською мовою.

Ігровий процес повинен підтримувати стабільну частоту кадрів (близько 60 FPS) на середньостатистичному обладнанні. Час завантаження гри та переходу між екранами не повинен перевищувати 3 секунд.

Веб-додаток, розміщений на Microsoft Azure, повинен мати час безвідмовної роботи не менше 99.9%, що забезпечується інфраструктурою провайдера. А також усе спілкування з серверами Firebase має відбуватися через захищений протокол HTTPS. Дані, що вводяться користувачем для таблиці лідерів, повинні проходити базову санітизацію для запобігання XSS-атак. Бекенд на Firebase повинен автоматично масштабуватися для обробки зростаючої кількості гравців та записів у таблиці лідерів. Хостинг на Azure має витримувати сплески відвідуваності.

Схема варіантів використання системи представлена на рис.2.1.

					ІПЗ.КР.Б-121-25-ПЗ	
						22

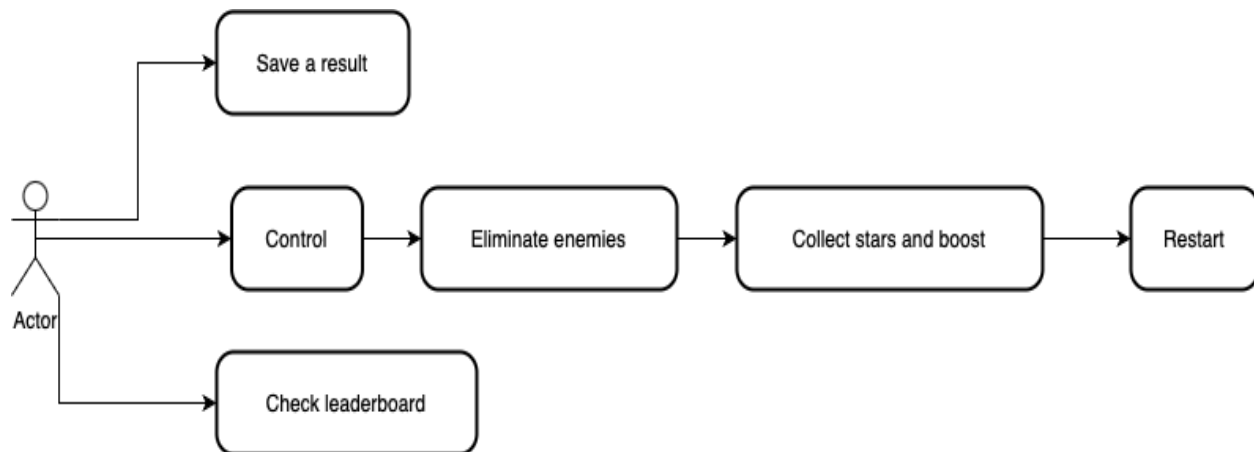


Рис. 2.1 – Схема варіантів використання стистеми

Таблиця 2.1

Варіанти використання

№	Варіант використання	Опис	Актор	Передумови	Післяумови
1	2	3	4	5	6
1	Управління (Control)	Приймає введення гравця (рух $\leftarrow/\rightarrow$, стрибок, стрільба) і передає команди ігровій логіці	Гравець	Гра запущена, активна сцена	Стан персонажа оновлено згідно з введенням
2	Знищення ворогів (Eliminate enemies)	Виконує стрільбу по ворожих об'єктах і обробляє їх видалення, оновлює рахунок	Гравець	Виконано «Управління»; на сцені присутні вороги	Ворог знищений, рахунок збільшено

Продовження таблиці 2.1

3	Збір зірок та бустів (Collect stars and boost)	Підбирає зірочки для прискорення стрільби та буст-пакети для підкиду вгору, застосовує ефекти бонусів	Гравець	Виконано попередні дії; на сцені є бонуси	Бонус активовано, оновлено індикатори швидкості або заряду буста
4	Перезапуск гри (Restart)	Ініціює новий раунд після смерті або завершення попереднього, скидає всі змінні гри до початкових	Гравець	Персонаж загинув або досягнуто умов завершення гри	Ігровий процес перезапущено, стан гри ініціалізовано заново
5	Збереження результату (Save a result)	Відправляє фінальний рахунок на сервер та зберігає його в таблиці лідерів	Гравець	Гру завершено; встановлено з'єднання з бекендом	Результат успішно збережено, відображено повідомлення про успіх
6	Перегляд таблиці лідерів (Check leaderboard)	Відображає актуальний список найвищих результатів із ранжуванням за позицією, ніком, рахунком, датою	Гравець	Користувач вибрав пункт «High Scores» у головному меню	Відображено прокручування у таблицю лідерів із підсвіченими топ-3 позиціями

Гра повинна підтримувати управління як з клавіатури на ПК (клавіші A/D або стрілки для руху, пробіл для стрільби), так і за допомогою екранних сенсорних кнопок на мобільних пристроях. Код повинен бути структурований відповідно до архітектури MVC для спрощення подальших модифікацій, виправлення помилок та додавання нового функціоналу. Для відстеження помилок на стороні клієнта використовуються стандартні інструменти розробника в браузері. Моніторинг роботи серверної частини здійснюється через панелі управління Firebase та Microsoft Azure.

2.2. Створення персонажів та сценаріїв руху: моделі, методи, алгоритми

В основі динаміки будь-якої гри лежить поведінка її персонажів та об'єктів. У цьому розділі детально розглядаються моделі даних, методи та алгоритми, що визначають функціонування та сценарії руху всіх динамічних сутностей у грі "Endless Jumper", включаючи гравця, ворогів, платформи та інші ігрові елементи.

Модель та алгоритми руху гравця. Гравець є центральним об'єктом системи, і його поведінка є найскладнішою. Модель гравця визначена через об'єкт-прототип `playerProto`, що слугує кресленням для створення екземпляра гравця. Ця модель інкапсулює всі ключові властивості: фізичні розміри (`width`, `height`), параметри руху (`speed`, `jumpForce`), поточну швидкість по осях (`dx`, `dy`), а також змінні стану, такі як `onGround` (чи знаходиться гравець на платформі), `powerupTimer` (таймер активного бонусу) та `hasBoost` (наявність бонусу для стрибка).

Сценарій руху гравця реалізований через його метод `update` та логіку обробки зіткнень у головному ігровому циклі. Горизонтальний рух напряму залежить від дій користувача: метод зчитує стан об'єкта `keys` і відповідно змінює горизонтальну швидкість `dx`. Вертикальний рух є поєднанням двох сил: імпульсу від стрибка та постійного впливу гравітації. На кожному кадрі до вертикальної швидкості `dy` додається значення `gravity`, що імітує прискорення вільного падіння. Ключовим моментом є алгоритм зіткнення з платформами. Коли гравець падає (`dy > 0`) і його координати перетинаються з координатами платформи, його вертикальна швидкість `dy` різко змінюється на негативне значення `-player.jumpForce`, що імітує відштовхування та стрибок. Це створює основну ігрову механіку нескінченного підйому.

Модель та сценарій руху ворогів. Модель ворога створюється за допомогою функції-фабрики `createEnemy`, яка генерує об'єкт з базовими властивостями, такими як розмір, швидкість та напрямок руху. Важливою особливістю моделі є те, що кожен ворог при своєму створенні отримує

посилання на платформу, на якій він з'явився. Це посилання є основою для його сценарію руху.

Алгоритм поведінки ворога є простим, але ефективним патрулюванням. У методі `update` ворога його горизонтальна координата `x` постійно змінюється відповідно до його швидкості та напрямку. Головна логіка полягає у перевірці меж. Як тільки ворог досягає лівого чи правого краю платформи, до якої він прив'язаний, його напрямок руху (`direction`) інвертується. Це створює передбачувану, але ефективну перешкоду для гравця, яка рухається в обмеженому просторі, не ризикуючи впасти з платформи.

Сценарії руху динамічних елементів середовища. Окрім гравця та ворогів, ігровий світ наповнений іншими динамічними елементами, що роблять його більш живим та складним. Рухомі платформи створюються з імовірністю 25% функцією `generatePlatform`. Їхня модель, окрім стандартних властивостей, містить параметри `moveSpeed`, `moveDirection` та `moveRange`, що визначають швидкість, напрямок та амплітуду руху. Сценарій їхнього руху реалізований в їхньому власному методі `update` і є схожим на логіку ворога: платформа рухається горизонтально до досягнення певної відстані від своєї початкової точки, після чого змінює напрямок. Це створює додатковий виклик для гравця, вимагаючи точності при стрибках.

Снаряди, що випускаються гравцем, є найпростішими динамічними об'єктами. Їхня модель – це простий об'єкт з координатами та фіксованою вертикальною швидкістю `dy`. На відміну від інших об'єктів, їхній рух не інкапсульований у власний метод `update`. Натомість, їхні координати оновлюються централізовано в головному ігровому циклі, який просто ітерує по масиву `bullets` і на кожному кадрі змінює їхню позицію. Такий підхід є ефективним для великої кількості простих, однотипних об'єктів.

2.3. Алгоритми створення платформи та середовища

Однією з ключових особливостей платформера "Endless Jumper" є нескінченний ігровий світ, який створюється на льоту. Це досягається за допомогою алгоритмів процедурної генерації контенту (PCG). Замість того, щоб заздалегідь створювати фіксовані рівні, система генерує платформи, ворогів та бонуси безпосередньо під час гри, базуючись на наборі визначених правил та елементів випадковості.

Процес генерації починається у функції `init`, яка викликається на старті кожної нової гри. Спочатку повністю очищуються всі масиви, що містять ігрові об'єкти. Потім створюється найперша платформа – вона є статичною, займає всю ширину екрана і слугує безпечною стартовою зоною для гравця. Відразу після цього запускається цикл, який викликає функцію `generatePlatform` для створення початкового набору платформ, що заповнюють екран по висоті. Це створює стартову "вежу" платформ, по якій гравець починає свій підйом.

```
function generatePlatform(yPos) {
    const width = Math.random() (120 - 60) + 60;
    const x = Math.random() (canvasWidth - width);
    const type = Math.random() < 0.25 ? 'moving' : 'static';
    const newPlatform = createPlatform(x, yPos, width, type);

    if (type === 'moving') {
        newPlatform.startX = Math.max(newPlatform.moveRange / 2, Math.min(x, canvasWidth - width -
newPlatform.moveRange / 2));
        newPlatform.x = newPlatform.startX;
    }

    platforms.push(newPlatform);

    const hasEnemy = Math.random() < 0.2 && type === 'static';
    const hasWeapon = Math.random() < 0.1;
    const hasBoost = Math.random() < 0.08; // NEW: Chance to spawn boost

    if (hasEnemy) {
        enemies.push(createEnemy(x, yPos, newPlatform));
    } else if (hasWeapon) {
        weaponPowerups.push(createWeaponPowerup(x + width/2 - 10, yPos - 25));
    }
}
```

```

    } else if (hasBoost) {
        boostPowerups.push(createBoostPowerup(x + width/2 - 10, yPos - 25));
    }
}

```

Функція `generatePlatform` є серцем усієї системи процедурної генерації. Вона приймає як аргумент вертикальну позицію (`yPos`), на якій потрібно створити нову платформу. Спочатку за допомогою `Math.random()` визначаються випадкові параметри платформи: її ширина (у діапазоні від 60 до 120 пікселів) та горизонтальне положення. Далі, з імовірністю 25%, платформа може стати рухомою (`moving`). Після створення самої платформи та додавання її до масиву `platforms`, запускається логіка генерації додаткових об'єктів. Ця логіка побудована на серії імовірнісних перевірок. З імовірністю 20% на платформі може з'явитися ворог, але за умови, що платформа є статичною – це зроблене для уникнення надто складних для гравця ситуацій. Якщо ворог не був створений, система перевіряє можливість створення бонусів: з імовірністю 10% може з'явитися бонус для зброї ("Rapid Fire"), а з імовірністю 8% – бонус для стрибка ("Super Jump"). Структура `if...else if` гарантує, що на одній платформі може з'явитися лише один об'єкт (або ворог, або один із бонусів), що запобігає їх накладанню та зберігає баланс гри. Відповідно до результату перевірки викликається відповідна функція-фабрика (`createEnemy`, `createWeaponPowerup`, `createBoostPowerup`), і новий об'єкт додається до свого масиву.

Механізм, що робить гру нескінченною, знаходиться всередині головної функції оновлення `update`. На кожному кадрі система перевіряє положення найвищої з існуючих платформ (`topPlatformY`). Якщо ця платформа, просуваючись вниз через рух камери, наближається до верхньої межі видимої зони (`cameraY - 100`), система викликає функцію `generatePlatform`. Нова платформа створюється на 80 пікселів вище поточної найвищої платформи. Цей процес безперервно повторюється, доки гравець піднімається вгору, створюючи ілюзію нескінченного ігрового світу та гарантуючи, що у гравця завжди буде шлях для подальшого просування.

2.4 Створення бази даних системи та функціоналу 2D платформи

Для реалізації функціоналу таблиці лідерів та забезпечення збереження ігрових результатів, у проєкті було спроектовано та інтегровано базу даних. З огляду на веб-орієнтовану архітектуру додатку та вимоги до гнучкості й масштабованості, було ухвалено рішення використати хмарну NoSQL базу даних Google Firestore, що є частиною платформи Firebase [27].

Вибір Firestore обґрунтований кількома ключовими перевагами. По-перше, це документо-орієнтована база даних, що дозволяє зберігати дані у вигляді JSON-подібних документів. Така структура є надзвичайно гнучкою і ідеально підходить для зберігання простих об'єктів, таких як ігрові результати, без необхідності заздалегідь визначати жорстку схему, як у реляційних базах даних [28]. По-друге, Firestore є безсерверним (serverless) рішенням, яке повністю управляється Google. Це знімає з розробника завдання з налаштування, підтримки та масштабування серверної інфраструктури, дозволяючи зосередитись безпосередньо на логіці додатку [29].

Спроектowana структура бази даних є максимально простою та ефективною для вирішення поставленої задачі. Вона складається з єдиної колекції під назвою scores. Колекція – це аналог таблиці в реляційних базах даних, що містить набір документів.

Кожен документ у колекції scores представляє один ігровий результат, збережений користувачем, і має наступну структуру:

- name (String): ім'я, яке гравець вводить після завершення гри;
- score (Number): кінцевий рахунок, набраний гравцем у даній ігровій сесії;
- createdAt (Timestamp): часова мітка, що автоматично генерується в момент створення запису; це поле є корисним для аудиту та може бути використане в майбутньому для реалізації сортування за датою або для створення тижневих чи щоденних таблиць лідерів.

Взаємодія додатку з базою даних чітко визначена двома операціями: записом та читанням. Запис даних відбувається після завершення гри, коли

гравець заповнює форму та відправляє свій результат. Функція `addDoc` з бібліотеки `Firebase` асинхронно створює новий документ у колекції `scores`, записуючи в нього ім'я, рахунок та поточну дату [30].

Читання даних відбувається при перегляді таблиці лідерів. Для цього формується спеціальний запит до `Firestore`, який виглядає наступним чином: `query(collection(db, "scores"), orderBy("score", "desc"), limit(10))`. Цей запит ефективно вибирає документи з колекції `scores`, сортує їх за полем `score` у порядку спадання (від найбільшого до найменшого) і обмежує результат першими десяти записами. Такий підхід гарантує, що користувач завжди бачить актуальну десятку найкращих гравців, а запит до бази даних залишається швидким та оптимізованим.

Таблиця 2.2

Опис таблиці Users

Назва	Тип	Опис
<code>createdAt</code>	UUID	Унікальний ідентифікатор
<code>name</code>	String	Ім'я гравця
<code>score</code>	Int	Рахунок гравця

Реалізація функціоналу 2D платформера

Перейдемо до огляду реалізації функціоналу 2D платформера

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-app.js";
import { getFirestore, collection, addDoc, query, orderBy, limit, getDocs } from
"https://www.gstatic.com/firebasejs/11.6.1/firebase-firestore.js";
import { getAuth, signInAnonymously, onAuthStateChanged } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-
auth.js";

document.addEventListener('DOMContentLoaded', () => {
  // --- Firebase Setup ---
  let db, auth;
  try {
    const firebaseConfig = {
```

```

apiKey: "AlzaSyCeELYG4yMBxe9GXNkD-GSzig3ly71Xr7s",
authDomain: "diploma.firebaseio.com",
projectId: "diploma",
storageBucket: "diploma.firebaseio.com",
messagingSenderId: "933282700060",
appId: "1:933282700060:web:6ce84c3daa30985eb2b8d9"
};

const app = initializeApp(firebaseConfig);
db = getFirestore(app);
auth = getAuth(app);

onAuthStateChanged(auth, (user) => {
  if (!user) {
    signInAnonymously(auth);
  }
});

} catch (e) {
  console.error("Error initializing Firebase. Leaderboard features will be disabled.", e);
  document.getElementById('leaderboard-btn').style.display = 'none';
  document.getElementById('leaderboard-btn-gameover').style.display = 'none';
}

const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');

// --- UI and Screen Elements ---
const startScreen = document.getElementById('start-screen');
});

```

На самому початку скрипту виконується імпорт необхідних модулів з бібліотеки Firebase. Це включає `initializeApp` для ініціалізації додатку, функції для роботи з базою даних `Firestore` (`getFirestore`, `collection`, `addDoc` та інші) і сервісом `Authentication` (`getAuth`, `signInAnonymously`). Весь код обгорнутий у слухач події `DOMContentLoaded`, що гарантує його виконання тільки після повного завантаження та парсингу HTML-документа.

Далі відбувається налаштування та ініціалізація `Firebase`. Об'єкт `firebaseConfig` містить унікальні ключі для підключення до хмарного проєкту.

Функція `initializeApp` створює екземпляр додатку, після чого ініціалізуються сервіси `Firestore` та `Authentication`. Важливою є реалізація анонімної автентифікації: `onAuthStateChanged` перевіряє, чи є активний користувач, і якщо немає, `signInAnonymously` автоматично створює анонімний сеанс. Це дозволяє зберігати результати гравців без необхідності повноцінної реєстрації. Блок `try...catch` обробляє можливі помилки підключення до `Firebase`, деактивуючи функціонал таблиці лідерів у разі збою.

Після налаштування `Firebase` скрипт отримує доступ до ключових елементів `DOM`, таких як ігрове поле (`canvas`), екрани меню та елементи інтерфейсу. Це створює основу для подальшої маніпуляції відображенням та станом гри.

```
const playerProto = {
  width: 30, height: 40, speed: 4, jumpForce: 12, dx: 0, dy: 0, onGround: false,
  shootCooldown: 0, baseShootCooldown: 20, powerupTimer: 0, hasBoost: false, // NEW: hasBoost property

  draw() {
    if (gameAssets.player) {
      ctx.drawImage(gameAssets.player, this.x, this.y - cameraY, this.width, this.height);
    }
  },
  update() {
    this.dx = 0;
    if (keys.right) this.dx = this.speed;
    if (keys.left) this.dx = -this.speed;
    this.x += this.dx;

    this.dy += gravity;
    this.y += this.dy;

    if (this.x < 0) this.x = 0;
    if (this.x + this.width > canvasWidth) this.x = canvasWidth - this.width;

    const currentCooldown = this.powerupTimer > 0 ? this.baseShootCooldown / 2 : this.baseShootCooldown;
    if (this.shootCooldown > 0) this.shootCooldown--;
    if (keys.shoot && this.shootCooldown === 0) {
      this.shoot();
    }
  }
};
```

```

        this.shootCooldown = currentCooldown;
    }
    if (this.powerupTimer > 0) this.powerupTimer--;
},
shoot() {
    bullets.push({ x: this.x + this.width / 2 - 2.5, y: this.y + 15, width: 5, height: 10, dy: -10 });
},
activatePowerup() {
    this.powerupTimer = 300; // 5 seconds at 60fps
}
};

```

playerProto – це шаблон об’єкта гравця, який об’єднує в собі як властивості фізики і руху, так і методи для відображення та взаємодії. Властивості width та height задають розміри персонажа, speed і jumpForce керують швидкістю горизонтального руху і силою стрибка, а dx і dy відповідають за поточний вектор руху по осях. Логіка падіння реалізована через додавання значення gravity до dy, після чого нове значення враховується в координаті y. Коли гравець доходить до країв полотна (canvas), його координати автоматично коригуються, щоб він не виходив за межі екрану. Механізм стрільби працює за затворним принципом: shootCooldown зменшується з кожним кадром, і коли він дорівнює нулю та натиснуто клавішу стрільби, викликається метод shoot(), який додає об’єкт кулі до масиву bullets. При наявності активного бонусу (powerupTimer > 0) інтервал між пострілами зменшується вдвічі. Метод activatePowerup() встановлює таймер бонусу на 300 кадрів, що еквівалентно п’яти секундам гри при 60 кадрах на секунду.

```

function createEnemy(x, y, platform) {
    return {
        x: x + platform.width / 2 - 15, y: y - 30, width: 30, height: 30, speed: 1, direction: 1, platform,
        draw() {
            if (gameAssets.enemy) {
                ctx.drawImage(gameAssets.enemy, this.x, this.y - cameraY, this.width, this.height);
            }
        },
        update() {

```

```

        this.x += this.speed * this.direction;
        if (this.x < this.platform.x || this.x + this.width > this.platform.x + this.platform.width) {
            this.direction = -1;
        }
    }
};
}

```

Функція `createPlatform(x, y, width, type = 'static')` повертає об'єкт платформи з фіксованою висотою 20 та вказаною шириною. Якщо тип платформи – 'moving', їй автоматично призначається випадкова швидкість руху (`moveSpeed`), напрямок (`moveDirection`) і радіус коливань (`moveRange`), а початкова позиція зберігається в `startX`. Метод `draw()` малює платформу з округленими кутами за допомогою ланцюжка викликів `lineTo` і `quadraticCurveTo`, забарвлюючи її в синій для рухомих або в сірий для статичних блоків. Метод `update()` зміщує платформу горизонтально і змінює напрямок руху, коли вона досягає межі своєї допустимої амплітуди.

```

function createWeaponPowerup(x, y) {
    return {
        x, y, width: 20, height: 20,
        draw() {
            ctx.save();
            ctx.translate(this.x + this.width / 2, (this.y - cameraY) + this.height / 2);
            ctx.rotate(gameTime * 0.1);
            ctx.fillStyle = 'f6e05e';
            ctx.beginPath();
            for (let i = 0; i < 5; i++) {
                ctx.lineTo(Math.cos(i * 2 * Math.PI / 5 * 2) * (this.width/2), Math.sin(i * 2 * Math.PI / 5 * 2) * (this.width/2));
                ctx.lineTo(Math.cos((i * 2 + 1) * Math.PI / 5 * 2) * (this.width/4), Math.sin((i * 2 + 1) * Math.PI / 5 * 2) * (this.width/4));
            }
            ctx.closePath();
            ctx.fill();
            ctx.restore();
        }
    }
}

```

Функція `createEnemy(x, y, platform)` створює об'єкт ворога, розташованого над даною платформою по центру. Властивості `width`, `height` і `speed` задають розміри та швидкість переміщення, а `direction` відповідає за поточний напрям руху. Метод `update()` переміщує ворога вздовж платформи, а при досягненні краю блоку (`platform.x` або `platform.x + platform.width`) змінює напрямок на протилежний. Малювання здійснюється через `ctx.drawImage()` за наявності завантаженого ресурсу `gameAssets.enemy`.

```
function createBoostPowerup(x, y) {
    return {
        x, y, width: 20, height: 20,
        draw() {
            ctx.fillStyle = '48bb78'; // Green color for boost
            ctx.beginPath();
            ctx.arc(this.x + this.width / 2, (this.y - cameraY) + this.height / 2, this.width / 2, 0, Math.PI * 2);
            ctx.fill();
            // Draw an upward arrow
            ctx.fillStyle = 'white';
            ctx.beginPath();
            ctx.moveTo(this.x + this.width / 2, this.y - cameraY + 4);
            ctx.lineTo(this.x + this.width - 4, this.y - cameraY + this.height / 2 + 2);
            ctx.lineTo(this.x + 4, this.y - cameraY + this.height / 2 + 2);
            ctx.closePath();
            ctx.fill();
        }
    }
}
```

Функція `createWeaponPowerup(x, y)` повертає об'єкт бонуса типу «зброя», який позиціонується у вказаних координатах і обертається навколо центру зі швидкістю, пропорційною глобальній змінній `gameTime`. У методі `draw()` викликається послідовність побудови правильного п'ятикутника зі змінними радіусами зовнішніх та внутрішніх вершин, що надає формі вигляду зірочки. Вона забарвлюється у жовтий колір, а завдяки трансляції та повороту контексту обертається відповідно до часу гри.

```
function createParticle(x, y, color) {
    const size = Math.random() * 5 + 2;
```

```

return {
  x, y, size, color, life: 60,
  dx: (Math.random() - 0.5) * 4, dy: (Math.random() - 0.5) * 4,
  draw() {
    ctx.fillStyle = this.color;
    ctx.globalAlpha = this.life / 60;
    ctx.fillRect(this.x, this.y - cameraY, this.size, this.size);
    ctx.globalAlpha = 1.0;
  },
  update() {
    this.x += this.dx;
    this.y += this.dy;
    this.life--;
  }
}
}

```

Функція `createBoostPowerup(x, y)` генерує об'єкт бонуса, який збільшує здібності гравця, і відображається як зелений круг із вкладеною білою стрілкою, спрямованою вгору. У методі `draw()` спочатку малюється коло через `ctx.arc()`, потім всередині нього будується простий трикутник, що символізує прискорення або підсилення стрибка. Забарвлення та побудова стрілки виконуються двома послідовностями команд `beginPath()` і `fill()`.

```

function init() {
  isGameOver = false;
  gameOverScreen.style.display = 'none';
  submitScoreForm.style.display = 'block';
  submitScoreForm.innerHTML = `<input type="text" id="player-name-input" placeholder="Enter Your
Name"><button type="submit" class="btn">Submit Score</button>`;

  player = Object.create(playerProto);
  player.x = canvasWidth / 2 - player.width / 2;
  player.y = canvasHeight - 100;
  player.powerupTimer = 0;
  player.hasBoost = false; // Reset boost on init

  cameraY = 0;
  score = 0;
  health = 100;
}

```

```

platforms = []; bullets = []; enemies = []; particles = []; weaponPowerups = []; boostPowerups = []; // Init boost
array

platforms.push(createPlatform(0, canvasHeight - 50, canvasWidth, 'static'));
for (let i = canvasHeight - 200; i > -200; i -= 80) {
    generatePlatform(i);
}

updateUI();

if (animationFrameId) cancelAnimationFrame(animationFrameId);
gameLoop();
}

```

init() – ця функція скидає стан гри до початкового: вона встановлює прапорець isGameOver у false, ховає екран завершення гри та показує форму для введення імені гравця, динамічно вставляючи в неї HTML-поле та кнопку. Далі створюється новий об'єкт гравця через Object.create(playerProto) із початковими координатами по центру екрану та нульовими лічильниками бонусів і охолодження стрільби. Камера по вертикалі встановлюється в нуль, показники рахунку та здоров'я скидаються до базових значень, а масиви платформ, куль, ворогів, частинок і обох типів бонусів ініціалізуються порожніми. Першою в масив платформ додається статична підлога на самому низу полотна, після чого у циклі генеруються піднесені платформи з інтервалом у 80 пікселів, починаючи на 200 пікселів вище низу й піднімаючись до -200 пікселів від верхньої межі екрану. Наприкінці виклику цієї функції оновлюється інтерфейс і запускається основний цикл гри через gameLoop(), попередньо скасувавши можливий попередній запит кадру.

```

function gameLoop() {
    if (isGameOver) return;
    update();
    draw();
    animationFrameId = requestAnimationFrame(gameLoop);
}

```

`gameLoop()` –рекурсивний цикл анімації: він перевіряє, чи гра ще не завершена, викликає логіку оновлення стану гри `update()`, потім рендеринг поточного кадру `draw()`, і зберігає ідентифікатор запиту наступного кадру в `animationFrameId`, щоб за допомогою `requestAnimationFrame` викликати себе знову.

```
function draw() {
    ctx.clearRect(0, 0, canvasWidth, canvasHeight);
    ctx.fillStyle = 'rgba(255, 255, 255, 0.5)';
    for (let i=0; i<100; i++) {
        const x = (i / 37) % canvasWidth;
        let y = ((i / 89) - cameraY / 0.1) % canvasHeight;
        if(y < 0) y += canvasHeight;
        ctx.fillRect(x, y, 2, 2);
    }
    platforms.forEach(p => p.draw());
    enemies.forEach(e => e.draw());
    weaponPowerups.forEach(wp => wp.draw());
    boostPowerups.forEach(bp => bp.draw()); // NEW: Draw boost power-ups
    bullets.forEach(b => {
        if (gameAssets.bullet) {
            ctx.drawImage(gameAssets.bullet, b.x, b.y - cameraY, b.width / 1.5, b.height / 1.5);
        }
    });
    particles.forEach(p => p.draw());
    player.draw();
}
```

`draw()` – відповідає за малювання всіх елементів на канвасі: спочатку очищається попередній кадр, потім напівпрозорими крапками створюється динамічний фоновий ефект зірок, де `cameraY` злегка зсуває рисунок. Після цього виконується послідовне малювання платформ, ворогів і всіх двох видів бонусів через їхні власні методи `draw()`. Кулі відтворюються за допомогою ресурсу `gameAssets.bullet`, а кулі відображаються через їхній метод `draw()`. На завершення викликається метод `draw()` гравця, який малює його спрайт у відповідній позиції з урахуванням зміщення камери.

```
if (isGameOver) return;
```

```

gameTime++;
player.update();
platforms.forEach(p => p.update());

```

Якщо змінна `isGameOver` істинна, одразу ж виходимо з функції (`return`). Інакше збільшуємо загальний лічильник часу гри `gameTime` і викликаємо методи `update()` для гравця та всіх платформ, щоб кожен об'єкт оновив свій внутрішній стан за поточний кадр.

```

player.onGround = false;

```

Перед перевіркою всіх колізій скидаємо прапорець `onGround` у `false`. Далі, при виявленні дотику до будь-якої платформи, ми його знову встановимо в `true`, що гарантує коректне виявлення приземлення лише один раз за кадр.

```

platforms.forEach(p => {
  if (player.dy > 0 &&
    player.x < p.x + p.width && player.x + player.width > p.x &&
    player.y + player.height > p.y && player.y + player.height < p.y + p.height + player.dy) {

    let jumpMultiplier = 1;
    if (player.hasBoost) {
      jumpMultiplier = 2;
      player.hasBoost = false;
      for (let i = 0; i < 20; i++) {
        particles.push(createParticle(
          player.x + player.width / 2,
          player.y + player.height,
          '48bb78'
        ));
      }
    }

    player.dy = -player.jumpForce * jumpMultiplier;
    player.y = p.y - player.height;
    player.onGround = true;

    if (p.type === 'moving') {
      player.x += p.moveSpeed * p.moveDirection;
    }
  }
}

```

```
}  
});
```

Перевірка зіткнення гравця з кожною платформою під час падіння ($dy > 0$).
Якщо колізія підтвердилася:

Якщо є активний буст (hasBoost), подвоюємо силу стрибка, відключаємо буст і генеруємо 20 зелених частинок. Задаємо нову швидкість по Y (dy) і позицію у так, щоб гравець опинився на краю платформи. Встановлюємо `onGround = true`. Якщо платформа рухома, зсуваємо гравця разом із нею.

```
if (player.y - cameraY < canvasHeight / 3) {  
    cameraY = player.y - canvasHeight / 3;  
}  
const currentScore = Math.floor(-cameraY / 10);  
if (currentScore > score) {  
    score = currentScore;  
}
```

Якщо гравець піднявся вище верхньої третини екрану, підтягуємо камеру (`cameraY`) за ним, щоб тримати його в цій зоні. Після цього обчислюємо поточний рахунок за висотою ($-cameraY/10$) і оновлюємо, тільки якщо досягли нового максимуму.

```
const topPlatformY = platforms[platforms.length - 1].y;  
if (topPlatformY > cameraY - 100) {  
    generatePlatform(topPlatformY - 80);  
}  
  
platforms = platforms.filter(p => p.y < cameraY + canvasHeight + 50);  
enemies = enemies.filter(e => e.platform.y < cameraY + canvasHeight + 50);  
weaponPowerups = weaponPowerups.filter(wp => wp.y < cameraY + canvasHeight + 50);  
boostPowerups = boostPowerups.filter(bp => bp.y < cameraY + canvasHeight + 50);
```

Коли найвище розташована платформа доходить близько до верхньої межі екрану, створюємо нову трохи вище. Далі фільтруємо всі масиви, залишаючи лише ті об'єкти (платформи, вороги, пауер-апи), які ще знаходяться в межах екрану плюс невеликий запас знизу.

```

bullets.forEach((b, i) => {
    b.y += b.dy;
    if (b.y < cameraY) bullets.splice(i, 1);
});

```

Кожна куля рухається вгору на величину `b.dy`. Якщо куля вилітає за верхню межу камери (`y < cameraY`), видаляємо її з масиву, щоб не обробляти зайві об'єкти.

```

enemies.forEach(e => e.update());

```

```

bullets.forEach((bullet, bIndex) => {
    enemies.forEach((enemy, eIndex) => {
        if (bullet.x < enemy.x + enemy.width && bullet.x + bullet.width > enemy.x &&
            bullet.y < enemy.y + enemy.height && bullet.y + bullet.height > enemy.y) {

            for (let i = 0; i < 15; i++) {
                particles.push(createParticle(
                    enemy.x + enemy.width / 2,
                    enemy.y + enemy.height / 2,
                    'f56565'
                ));
            }

            enemies.splice(eIndex, 1);
            bullets.splice(bIndex, 1);
            score += 50;
        }
    });
});

```

Спочатку оновлюємо стан кожного ворога (`e.update()`). Потім у вкладених циклах перевіряємо зіткнення кожної кулі з кожним ворогом. При влучанні генеруємо червоні кулі, видаляємо ворога й кулю та додаємо 50 очок до рахунку.

```

enemies.forEach(enemy => {
    if (player.x < enemy.x + enemy.width && player.x + player.width > enemy.x &&
        player.y < enemy.y + enemy.height && player.y + player.height > enemy.y) {

        health--;
    }
});

```

```

    for (let i = 0; i < 5; i++) {
        particles.push(createParticle(
            player.x + player.width / 2,
            player.y + player.height / 2,
            'f6e05e'
        ));
    }
    if (health <= 0) endGame();
}
});

```

Якщо прямокутники гравця й ворога перетинаються, зменшуємо health, генеруємо кілька жовтих частинок навколо гравця і, коли здоров'я досягає нуля, закінчуємо гру викликом endGame().

```

weaponPowerups.forEach((wp, index) => {
    if (player.x < wp.x + wp.width && player.x + player.width > wp.x &&
        player.y < wp.y + wp.height && player.y + player.height > wp.y) {

        player.activatePowerup();
        weaponPowerups.splice(index, 1);
        for (let i = 0; i < 20; i++) {
            particles.push(createParticle(
                wp.x + wp.width / 2,
                wp.y + wp.height / 2,
                'f6e05e'
            ));
        }
    }
});

```

Колізія з пауер-апом зброї викликає метод activatePowerup(), видаляє цей пауер-ап з масиву і створює навколо нього серію жовтих частинок.

```

boostPowerups.forEach((bp, index) => {
    if (player.x < bp.x + bp.width && player.x + player.width > bp.x &&
        player.y < bp.y + bp.height && player.y + player.height > bp.y) {

        player.hasBoost = true;
        boostPowerups.splice(index, 1);
        for (let i = 0; i < 20; i++) {

```

```

particles.push(createParticle(
    bp.x + bp.width / 2,
    bp.y + bp.height / 2,
    '48bb78'
));
}
}
});

```

Аналогічно до зброї, але тут при підборі пауер-апу бусту ми встановлюємо `hasBoost = true` і генеруємо зелені кулі.

Якщо гравець опустився нижче відображуваної області (`player.y > cameraY + canvasHeight`), викликаємо `endGame()`. Після цього в будь-якому разі оновлюємо елементи інтерфейсу викликом `updateUI()`.

`update()` – центральна функція ігрової логіки, яка також одразу повертає керування, якщо встановлено `isGameOver`. Вона інкрементує глобальний лічильник часу гри `gameTime`, викликає оновлення властивостей гравця та всіх платформ, після чого перевіряє зіткнення гравця з тими платформами, на яких він падає зверху, і, в разі попадання, обчислює відскок із врахуванням тимчасового збільшення сили стрибка при активації буста, застосовує горизонтальне зміщення разом із рухомою платформою, створює кулі при приземленні та відновлює прапорець `onGround`. Далі камера по вертикалі стежить за гравцем, рахунок перераховується відносно підйому камери, а за потреби з'являються нові платформи над полем зору. Старі об'єкти платформ, ворогів і бонусів видаляються, якщо вони випадають за межі екрану, а кожна куля рухається вгору і знищується при виході з поля зору. Логіка збивання ворогів кулями викликає вибух частинок та нараховує додаткові очки, зіткнення ворогів з гравцем зменшує здоров'я і запускає завершення гри при досягненні нуля, а підбирання зброї чи буста активує відповідний метод у прототипі гравця та створює ефект частинок. В фіналі кожної ітерації обробляються кулі та їхнє життєве коло, перевіряється падіння гравця за межі екрану для виклику `endGame()`, і оновлюється інтерфейс користувача.

```
function updateUI() {
    scoreEl.textContent = `Score: ${score}`;
    healthEl.textContent = `Health: ${Math.max(0, health)}`;
    if (player && player.powerupTimer > 0) {
        powerupTimerEl.textContent = `Rapid Fire: ${((player.powerupTimer / 60).toFixed(1))s}`;
        powerupTimerEl.style.display = 'block';
    } else {
        powerupTimerEl.style.display = 'none';
    }
    // --- NEW: Update boost status UI ---
    if (player && player.hasBoost) {
        boostStatusEl.style.display = 'block';
    } else {
        boostStatusEl.style.display = 'none';
    }
}
```

Система управління реалізована через глобальні слухачі подій клавіатури (keydown та keyup). Вони змінюють стан об'єкта keys, який зберігає інформацію про те, які клавіші натиснуті в даний момент. Цей об'єкт потім використовується в методі player.update() для визначення напрямку руху. Для мобільних пристроїв реалізована підтримка сенсорного управління за допомогою екранних кнопок, які реагують на події touchstart та touchend.

Взаємодія з елементами інтерфейсу, такими як кнопки "Start Game", "Restart" та "High Scores", здійснюється через слухачі подій click. Наприклад, натискання на кнопку старту запускає процес завантаження ассетів і викликає функцію init.

Форма відправки результату (submitScoreForm) обробляє подію submit. Вона асинхронно відправляє дані (ім'я гравця та рахунок) до бази даних Firestore за допомогою раніше імпортованої функції addDoc.

Функція updateUI відповідає за синхронізацію ігрових даних (рахунок, здоров'я, статус бонусів) з відповідними елементами HTML, забезпечуючи актуальне відображення інформації для гравця.

```
async function showLeaderboard() {
    startScreen.style.display = 'none';
```

```

gameOverScreen.style.display = 'none';
leaderboardScreen.style.display = 'flex';
leaderboardList.innerHTML = '<li>Loading...</li>';

if (!db) {
  leaderboardList.innerHTML = '<li>Leaderboard is unavailable.</li>';
  return;
}

try {
  const q = query(collection(db, "scores"), orderBy("score", "desc"), limit(10));
  const querySnapshot = await getDocs(q);
  leaderboardList.innerHTML = "";
  if (querySnapshot.empty) {
    leaderboardList.innerHTML = '<li>No scores yet. Be the first!</li>';
    return;
  }
  let rank = 1;
  querySnapshot.forEach((doc) => {
    const data = doc.data();
    const li = document.createElement("li");
    li.innerHTML = `<span>${rank}. </span> ${data.name}</span><span>${data.score}</span>`;
    leaderboardList.appendChild(li);
    rank++;
  });
} catch (error) {
  console.error("Error getting leaderboard: ", error);
  leaderboardList.innerHTML = '<li>Could not load scores.</li>';
}
}

```

Даний код відповідає за показ екрану з таблицею лідерів. Перш за все приховуються інші екрани і відображає контейнер `leaderboardScreen`. Далі виводиться тимчасове повідомлення «Loading...». Якщо з'єднання з базою даних (db) відсутнє, замість завантаження списку показується повідомлення, що таблиця недоступна. Якщо база є, формується запит до колекції «scores» у Firestore: сортування за полем `score` у спадному порядку та обмеження до 10 записів. Після отримання даних попередній вміст списку очищається. Якщо

записів немає, виводиться повідомлення, що ще немає жодного результату. Інакше ітерується кожен документ, витягуються поля name та score, створюється елемент з рангом, ім'ям гравця і балами, і додається до списку. У разі помилки під час запиту вона логуються в консолі й відображається повідомлення про неможливість завантажити результати.

```
window.addEventListener('resize', resizeCanvas);
window.addEventListener('keydown', e => {
  if (e.code === 'ArrowRight' || e.code === 'KeyD') keys.right = true;
  if (e.code === 'ArrowLeft' || e.code === 'KeyA') keys.left = true;
  if (e.code === 'Space') keys.shoot = true;
});
window.addEventListener('keyup', e => {
  if (e.code === 'ArrowRight' || e.code === 'KeyD') keys.right = false;
  if (e.code === 'ArrowLeft' || e.code === 'KeyA') keys.left = false;
  if (e.code === 'Space') keys.shoot = false;
});
```

В даному фрагменті налаштовані глобальні обробники подій на вікні. Подія resize викликає функцію resizeCanvas, яка підлаштовує розміри ігрового полотна під поточні розміри вікна. Далі дві події для натискання та відпускання клавіш: при keydown фіксується, якщо натиснуто праву стрілку або клавішу D — встановлюється прапорець keys.right = true; аналогічно для лівої стрілки або A — keys.left = true; пробіл відповідає стрільбі (keys.shoot = true). При keyup відповідні прапорці скидаються у false. Це дозволяє в ігровому циклі читати стан keys і переміщувати або виконувати дію.

```
restartButton.addEventListener('click', init);
leaderboardBtn.addEventListener('click', showLeaderboard);
backToMenuBtn.addEventListener('click', () => {
  leaderboardScreen.style.display = 'none';
  startScreen.style.display = 'flex';
});
document.getElementById('leaderboard-btn-gameover').addEventListener('click', showLeaderboard);
```

Ця частина коду прив'язує обробники до елементів інтерфейсу. При кліку на кнопку restartButton викликається функція init, що, скидає стан гри та починає

заново. Кнопка `leaderboardBtn` показує екран таблиці лідерів через виклик `showLeaderboard`. Кнопка `backToMenuBtn` приховує екран лідерів і повертає до стартового екрана, а в її колбецці змінюється CSS-властивість `display` відповідних контейнерів. Також є окремий елемент з `id 'leaderboard-btn-gameover'`, який на екрані кінця гри повторно відкриває таблицю лідерів.

```
submitScoreForm.addEventListener('submit', async (e) => {
  e.preventDefault();
  if (!db) {
    alert('Cannot submit score, database not connected.');
```

```
    return;
  }
  const playerNameEl = document.getElementById('player-name-input');
  const playerName = playerNameEl.value || 'Anonymous';
  const submitBtn = submitScoreForm.querySelector('button');
  submitBtn.disabled = true;
  submitBtn.textContent = 'Submitting...';

  try {
    await addDoc(collection(db, "scores"), { name: playerName, score: score, createdAt: new Date() });
    submitScoreForm.innerHTML = '<p>Score submitted!</p>';
  } catch (error) {
    console.error("Error adding document: ", error);
    submitBtn.textContent = 'Try Again';
    submitBtn.disabled = false;
  }
});
```

Функціонал таблиці лідерів реалізований в асинхронній функції `showLeaderboard`. При її виклику приховуються інші екрани та відображається екран таблиці лідерів.

Функція створює запит до колекції `"scores"` у `Firestore` за допомогою функції `query`. Запит включає сортування результатів за полем `score` у порядку спадання (`orderBy("score", "desc")`) та обмежує вибірку до 10 найкращих записів (`limit(10)`).

`await getDocs(q)` асинхронно отримує дані з сервера. Після успішного отримання, список на сторінці очищується, і для кожного документа з

querySnapshot створюється новий елемент , який містить ранг, ім'я гравця та його рахунок. Потім цей елемент додається до списку лідерів на сторінці. Блок try...catch обробляє можливі помилки під час запиту до бази даних, інформуючи користувача про проблему.

Висновки до другого розділу

У другому розділі було показано, як із загального аналізу вимог переходять до конкретного технічного проєктування «Endless Jumper». Спочатку визначено ключові функціональні та нефункціональні потреби гри, зокрема механіку керування персонажем, обробку станів програми, динамічне створення ворогів і бонусів, а також вимоги до продуктивності й безпеки. Це дозволило чітко окреслити роль кожного елементу в системі та встановити загальну архітектуру взаємодії між користувачем і додатком.

Далі запропоновано об'єктно-орієнтовану модель, де прототип гравця інкапсулює властивості та методи для руху, стрибків і стрільби, а фабричні функції створюють платформи, ворогів, бонуси та кулі. Окрему увагу приділено проєктуванню бази даних на Firestore: колекція результатів із полем часу дозволяє зберігати й отримувати топ-10 гравців, забезпечуючи гнучкість і масштабованість. Алгоритми ініціалізації, основного циклу оновлення та рендерингу продемонстровано на прикладах коду, що забезпечують безперебійну роботу гри й оновлення інтерфейсу.

Особливої ваги набув механізм процедурної генерації контенту, який гарантує нескінченне наповнення світу платформи та підтримує баланс складності через ймовірнісні правила появи ворогів і бонусів. Ретельно описані алгоритми руху динамічних об'єктів – як гравця під дією гравітації, так і ворогів та рухомих платформ – створюють передбачувану, але водночас динамічну поведінку елементів. У результаті другий розділ сформував однозначну технічну основу для подальшої реалізації та інтеграції всіх компонентів гри.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ОСОБЛИВОСТІ НАЛАШТУВАННЯ (РОЗГОРТАННЯ, РОБОТИ) ПЛАТФОРМЕРА

У третьому розділі детально розглянуто безпосередній інтерфейс веб-застосунку «Endless Jumper» та послідовність його використання кінцевим користувачем. Спочатку описано особливості розгортання гри в середовищі Azure за допомогою Docker-контейнера, що дозволяє забезпечити простий і відтворюваний процес налаштування без необхідності локальної інсталяції. Далі проаналізовано ключові елементи головного меню та ігрової сцени: від розташування логотипу та блоків керування до адаптивних графічних ефектів, які створюють атмосферу безперервного підйому. Окрему увагу приділено сторінці «Таблиця лідерів» із реального часу оновлюваною інформацією про найкращі результати спільноти.

Також розділ описує комплекс процедур тестування, що складається з ручного проходження сценаріїв взаємодії користувача із застосунком та узагальнення отриманих результатів у вигляді таблиці. Зміст третього розділу покликаний продемонструвати, як реалізовані дизайнерські рішення та функціональні вимоги забезпечують інтуїтивність інтерфейсу та надійну роботу гри в різних середовищах.

3.1 Порядок встановлення та налаштування параметрів 2D платформера

Оскільки розроблений платформер є веб застосунком, він не потребує встановлення, натомість ми розмістимо його на Azure за допомогою Docker. Для цього спочатку створимо Dockerfile з наступним вмістом:

```
# Use the official lightweight Nginx image
FROM nginx:alpine

# Remove default Nginx website
RUN rm -rf /usr/share/nginx/html/

# Copy your static site into the Nginx html directory
```

```
COPY . /usr/share/nginx/html/
```

```
# Expose port 80 to the Docker host  
EXPOSE 80
```

```
# Start Nginx in the foreground (so the container doesn't exit)  
CMD ["nginx", "-g", "daemon off;"]
```

Далі за допомогою команд зберемо image контейнера та запустимо його на Dockerhub

```
# Build the image (tag it "endless-jumper" for example)  
docker build -t endless-jumper .
```

```
# Run the container, mapping host port 8080 → container port 80  
docker run -d -p 8080:80 endless-jumper
```

Після того, як ми зібрали Docker-образ і запустили його в DockerHub (під тегом /endless-jumper:latest) можемо розгорнути його в Azure одним із двох основних способів – через Azure Container Instances(ACI) або через Azure Web App for Containers.

Далі створимо ресурсну групу та план App Service:

```
az group create \  
  --name rg-endlessjumper \  
  --location eastus  
  
az appservice plan create \  
  --name plan-endlessjumper \  
  --resource-group rg-endlessjumper \  
  --sku F1 \  
  --is-linux
```

Після цього розгорнемо Web App, вказавши Docker-образ із DockerHub:

```
az webapp create \  
  --resource-group rg-endlessjumper \  
  --plan plan-endlessjumper \  
  --name endless-jumper-app \  
  --deployment-container-image-name username/endless-jumper:latest
```

При успішному виконанні цієї команди Azure створить веб-додаток– за кілька хвилин ми зможемо відкрити його в браузері й побачити запущений 2D-платформер.

Якщо необхідно внести виправлення в код, додаємо їх і пушимо новий тег образу `endless-jumper:v1.1`) оновити розгортання можна так:

```
az webapp config container set \  
  --name endless-jumper-app \  
  --resource-group rg-endlessjumper \  
  --docker-custom-image-name username/endless-jumper:v1.1  
az webapp restart \  
  --name endless-jumper-app \  
  --resource-group rg-endlessjumper
```

Цей підхід дозволяє автоматизувати CI/CD.

3.2 Інтерфейс та особливості використання

Після завантаження сторінки користувач одразу бачить продумане та лаконічне головне меню. У його верхній частині розташовано логотип гри та назву «Endless Jumper. Під назвою розміщено короткий блок інструкцій з керування:

Стрілки вліво/вправо або A/D для руху персонажа;

Пробіл або Кнопка “Стрибок” на сенсорних пристроях для виконання стрибка;

ЛКМ або Touch для стрільби по ворогах.

Ці підказки винесено окремим текстовим блоком у центрі екрану, щоб новий гравець міг швидко зорієнтуватися в механіці.

Нижче інструкцій розташована велика кнопка «Почати гру». Вона оформлена яскравим фоном і ефектом наведення, що привертає увагу та підштовхує до початку геймплею. Поруч із нею – кнопка «Таблиця лідерів», яка переводить на окрему сторінку з переліком найкращих результатів. Ця кнопка виконана в

іншому, але сумісному стилі, щоб користувач міг одразу зрозуміти: тут відкривається статистика, а не безпосередній запуск гри.

Весь інтерфейс адаптивний: елементи автоматично підлаштовуються під розміри екрану мобільного пристрою або десктопної панелі браузера. Фон головного меню – динамічна анімована картина, що відтворює безкінечне піднімання: рухомі платформи та силуети гравця на тлі – лиш ненав’язливо створюють атмосферу гри, але не відволікають від вибору дії. Завдяки такому дизайну інтерфейс залишається зрозумілим.

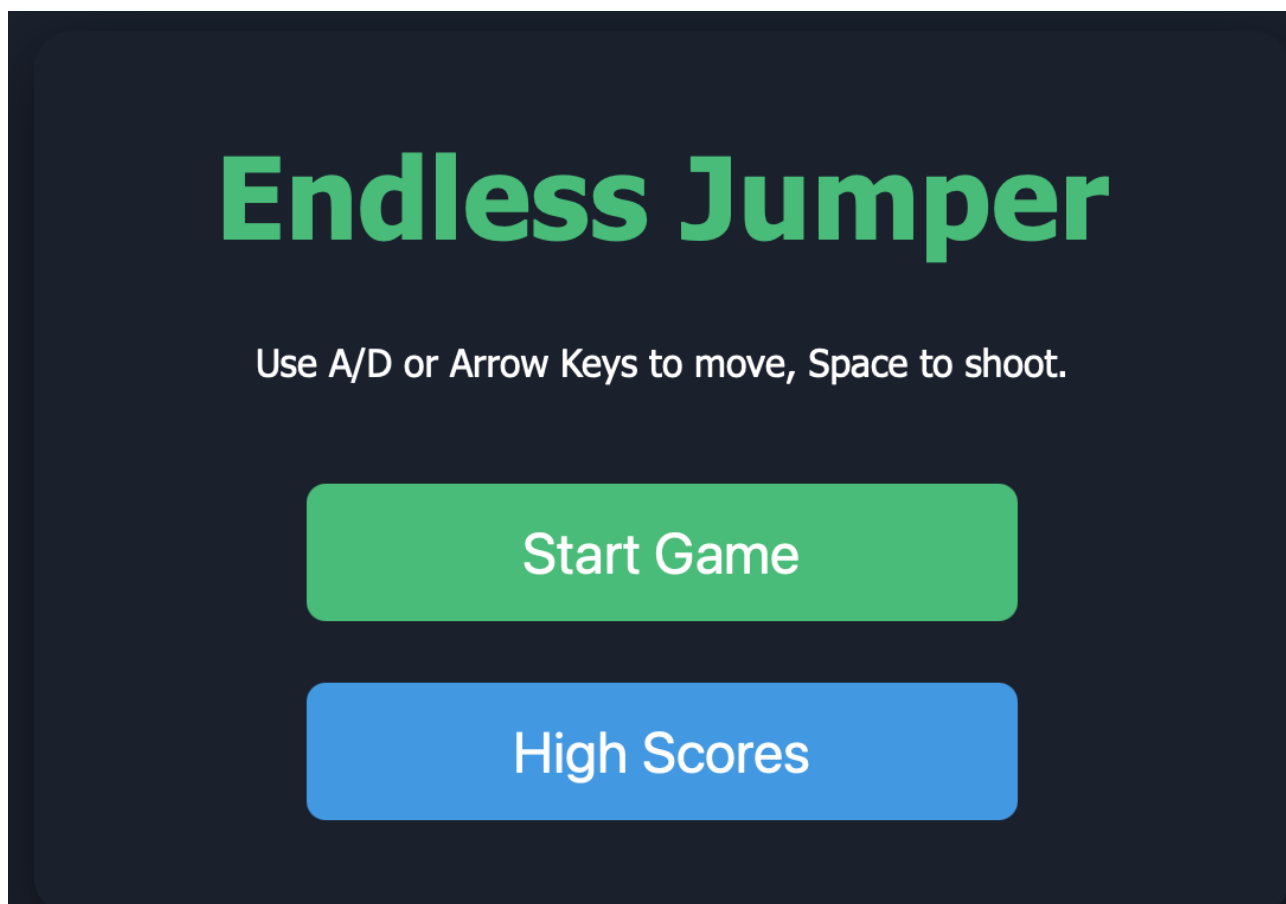


Рис 3.1. Зовнішній вигляд початкового екрану

Після натискання кнопки «High Scores» відкривається окрема сторінка з таблицею найкращих результатів. У верхній частині екрану розміщено заголовок «Таблиця лідерів», а під ним – коротка підказка: «Оновлюється в реальному часі». Основною частиною сторінки є вертикально прокручуваний список записів у такому форматі:

- позиція (№);

					ІПЗ.КР.Б-121-25-ІЗ	
						52

- нікнейм гравця;
- його рахунок.

Кожен рядок таблиці оформлений чіткими розділювачами, щоб легко відрізнити позиції. У нижній частині сторінки розташована кнопка «Повернутися», що повертає користувача до головного меню

Інтерфейс адаптивний: на мобільних пристроях таблиця підлаштовується під ширину екрана, показуючи тільки найголовніше Фон сторінки витриманий у тій же стилістиці з динамічними платформами на задньому плані, але з дещо приглушеними кольорами, щоб не відволікати від контенту. Таким чином користувач отримує зручний і зрозумілий перегляд найвищих досягнень спільноти.

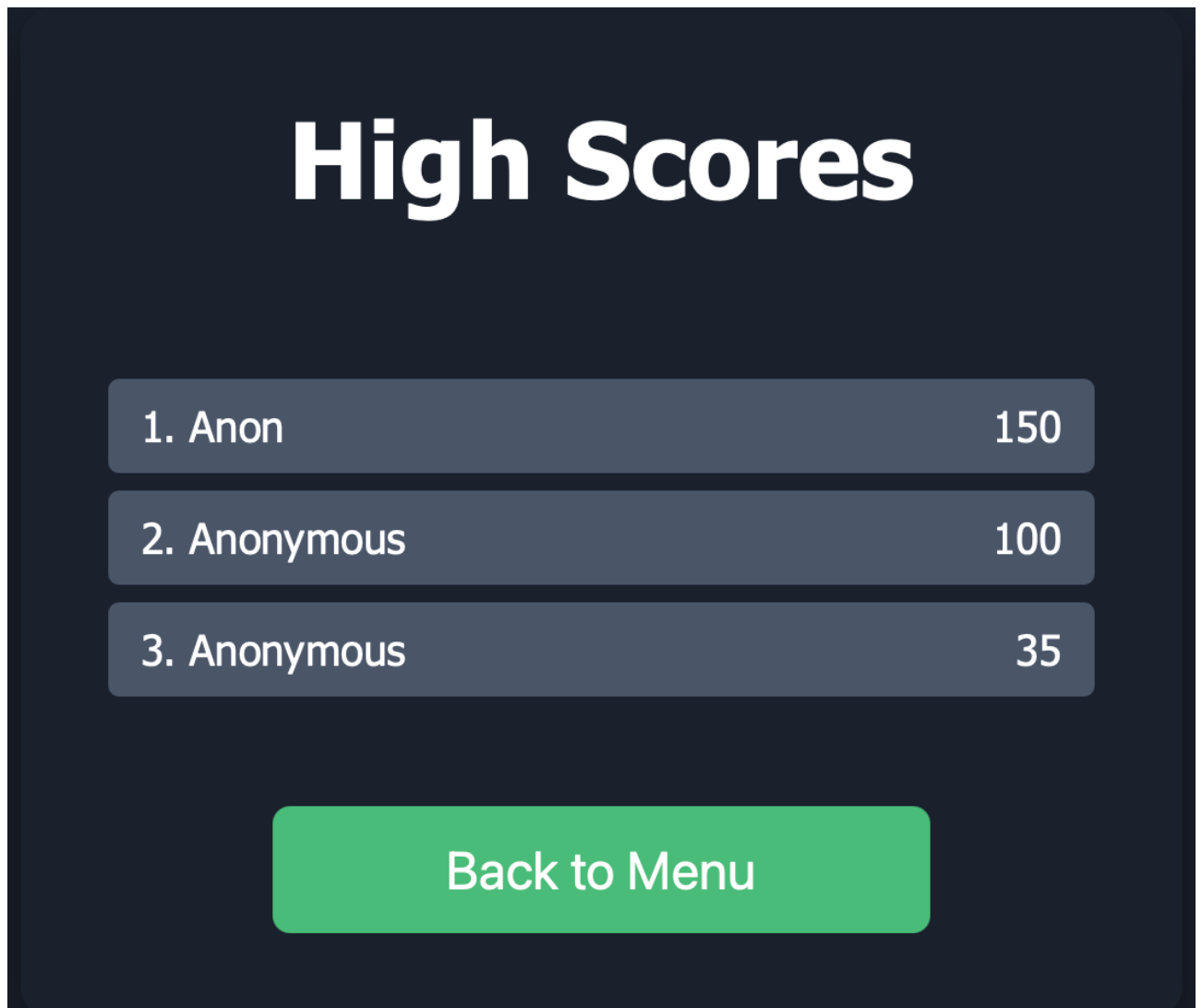


Рис 3.2. Зовнішній вигляд таблиці лідерів

Після натискання «Почати гру» на екрані одразу з’являється ігрова сцена:

Гравець опиняється на нижній платформі, а перед ним на різних висотах розташовані вороги червоного кольору – вони переміщуються взад-вперед або стрибають, намагаючись зачепити персонажа. По дорозі трапляються зоряні бонуси у вигляді жовтих зірочок: збираючи їх, ви тимчасово підвищуєте швидкість стрільби, що дає можливість швидше розчищати шлях від ворогів. Також серед платформ можна знайти буст-пакети (підкиди) – яскраві кулясті іконки, які при збиранні миттєво викидають вас вгору на кілька рівнів, допомагаючи долати великі відстані та уникати небезпечних пасток.

Верхня частина екрану відведена під індикатор рахунку, лічильник зібраних бонусів та показчик поточного рівня заряду буста, щоб ви завжди бачили, скільки ресурсів залишилося. Фон сцени анімований – рухливі хмари й силуети вдалечині підтримують відчуття безперервного підйому. Таким чином, геймплей відразу занурює гравця в поєднання стрибків, стрільби та тактичного збору бонусів.

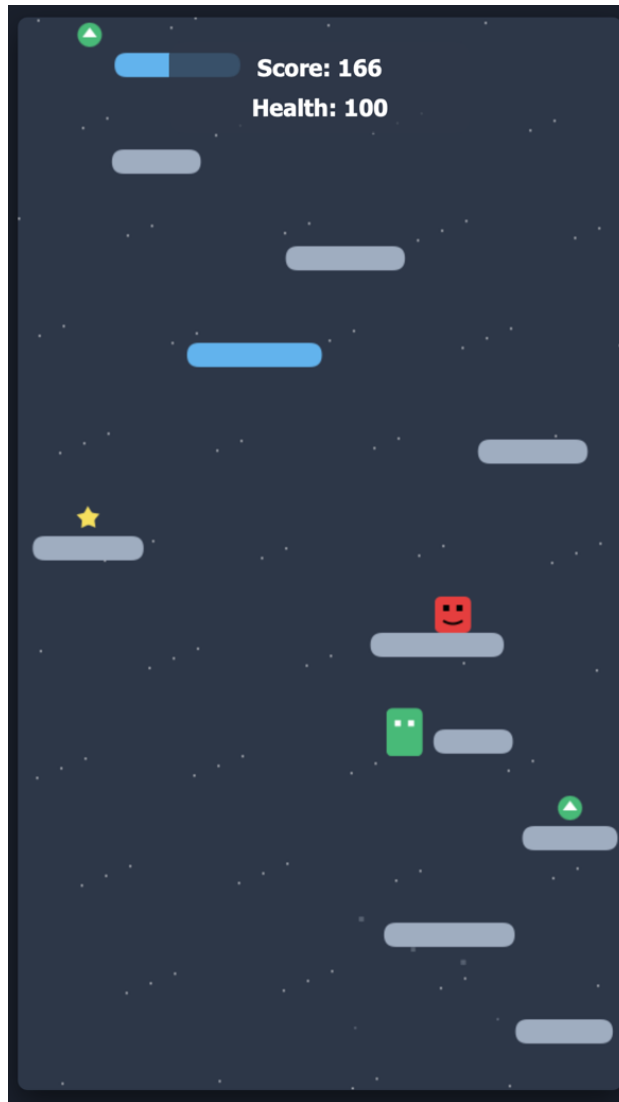


Рис 3.3. Зовнішній вигляд ігрового процесу

При натисканні клавіші Пробіл персонаж миттєво вистрілює серією куль вгору. Ці кулі летять у напрямку ворогів, і при влучанні викликають анімацію вибуху з коротким спалахом, зникнення ворога та миттєве оновлення рахунку. Якщо куля не влучає у ціль, вона продовжує політ до меж екрану

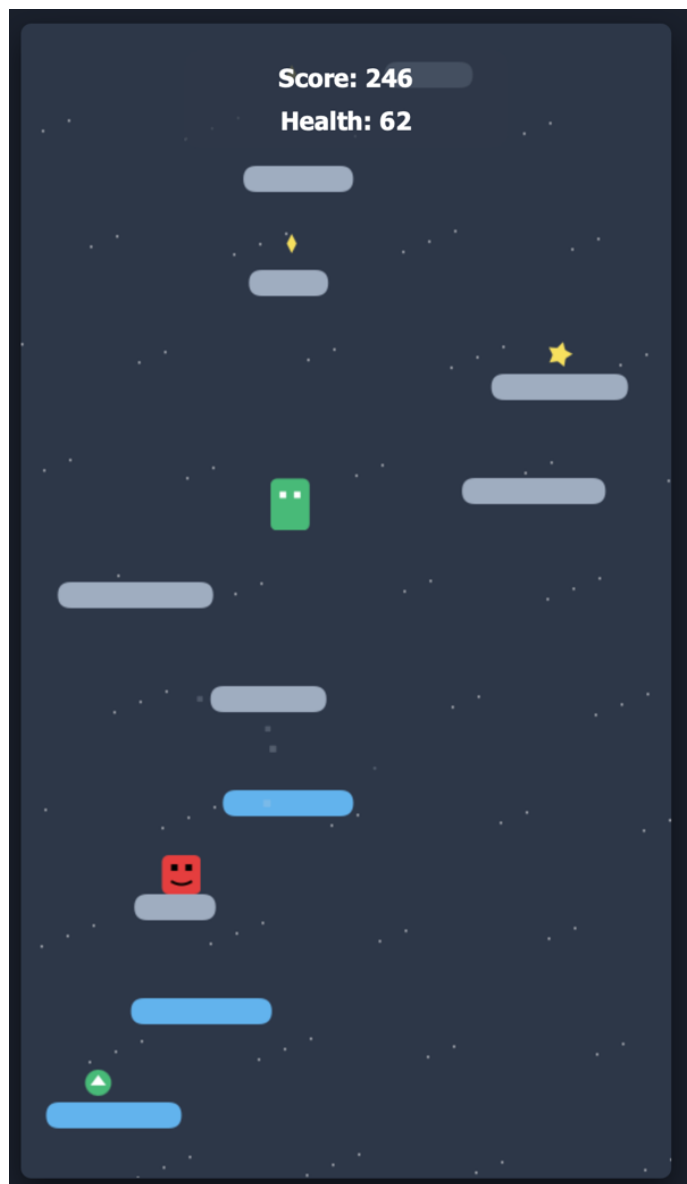


Рис 3.4. Вигляд куль якими стріляє гравець

При падінні персонажа, чи коли в гравця закінчується здоров'я, користувач бачить перед собою екран "Game Over"¹. Цей екран інформує гравця про завершення поточної ігрової сесії та надає можливість взаємодіяти з кількома ключовими елементами. На екрані відображається фінальний рахунок гравця, отриманий за час гри². Також є поле для введення імені, що дозволяє гравцеві зберегти свій результат у таблиці лідерів.

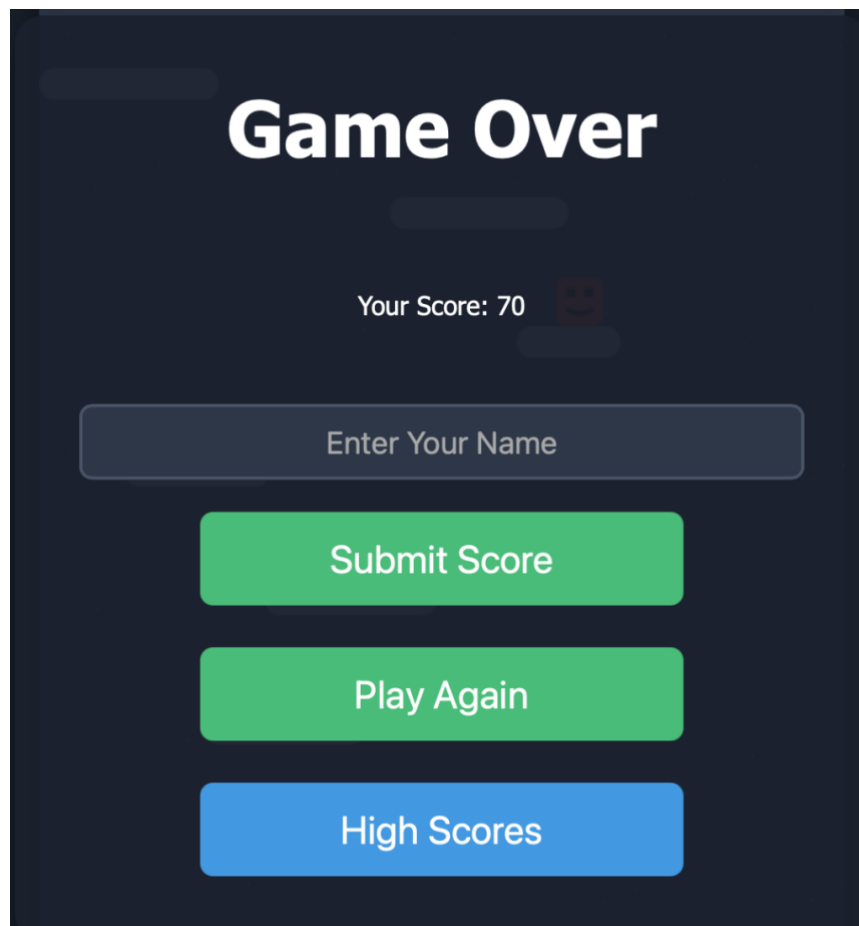


Рис 3.5 – Екран завершення гри

Під полем для введення імені розташовані три кнопки: "Submit Score" (Відправити рахунок), "Play Again" (Грати знову) та "High Scores" (Таблиця лідерів). Це дає користувачеві вибір: або зафіксувати свій результат, або одразу розпочати нову гру, або переглянути найкращі досягнення інших гравців, (рис3.5).

3.3 Тестування роботи 2D платформера

Тест-кейс 1

У цьому тест-кейсі я запускаю додаток і, опинившись у головному меню, інтуїтивно перевіряю наявність назви «Endless Jumper», блоку з інструкціями з керування та двох кнопок: «Почати гру» і «High Scores». Після наведення курсору на кожну кнопку я очікую побачити візуальний ефект hover, а натискання має бути можливим та приводити до відповідної дії без затримок.

					ІПЗ.КР.Б-121-25-ПЗ	
						57

Тест-кейс 2

У наступному тест-кейсі я в головному меню натискаю на кнопку «High Scores» і спостерігаю, як відкривається окрема сторінка з заголовком «Таблиця лідерів». Я очікую, що одразу з'явиться прокручуваний список ігрових записів із зазначенням позиції, нікнейму, рахунку та дати/часу досягнення.

Тест-кейс 3

У третьому тест-кейсі, перебуваючи в головному меню, я натискаю «Почати гру» та очікую побачити ігрову сцену з нижньою платформою й персонажем у центрі екрану. Одночасно мають відобразитися індикатори рахунку, кількості бонусів та заряду буста, а на різних висотах сцени – вороги, зірочки та буст-пакети. Важливо, щоб усе завантажувалося плавно, без артефактів або зависань.

Тест-кейс 4

Потім я перевіряю механіку стрільби: у грі я натискаю пробіл (або відповідну кнопку на екрані) і стежу, як персонаж вистрілює серією куль, які летять у напрямку ворогів. Очікую побачити ефект пострілу, а при влучанні – анімацію вибуху з коротким спалахом, зникнення ворога й миттєве оновлення рахунку на +10 або +20. Якщо куля не влучає, вона повинна продовжити політ до меж екрану.

Тест-кейс 5

Нарешті, у п'ятому тест-кейсі я рухаюся до зірочки на платформі і очікую, що вона автоматично зникне в момент контакту, а індикатор швидкості стрільби на панелі оновиться, показавши підвищену частоту пострілів на певний час. Далі я дістаюся до буст-пакета, підбираю його та бачу, як персонаж миттєво підкидається вгору на кілька рівнів, а індикатор заряду буста змінюється відповідно. Ці ефекти мають працювати коректно та без затримок.

В таблиці 3.1 наведено результати мануального тестування

					ІПЗ.КР.Б-121-25-ПЗ	
						58

Тест-кейс 1

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Запуск додатку та перевірка головного меню	Наявність назви "Endless Jumper", інструкцій, кнопок "Почати гру" та "High Scores"; візуальний ефект hover при наведенні курсору; можливість натискання кнопок з відповідною дією без затримок.	Як і очікувалось	Виконано
2	В головному меню натиснути кнопку "High Scores"	Відкриття сторінки "Таблиця лідерів" з прокручуваним списком ігрових записів (позиція, нікнейм, рахунок, дата/час).	Як і очікувалось	Виконано
3	В головному меню натиснути кнопку "Почати гру"	Поява ігрової сцени з нижньою платформою та персонажем у центрі; відображення індикаторів рахунку, бонусів, заряду буста; наявність ворогів, зірочок, буст-пакетів; плавне завантаження без артефактів.	Як і очікувалось	Виконано
4	Натиснути пробіл (стрільба)	Персонаж вистрілює серією куль; ефект пострілу; при влучанні - анімація вибуху, зникнення ворога, оновлення рахунку (+10 або +20); куля продовжує політ до меж екрану, якщо не влучає.	Як і очікувалось	Виконано
5	Рух до зірочки та буст-пакета	Зірочка зникає при контакті, індикатор швидкості стрільби оновлюється (підвищена частота пострілів); буст-пакет підбирається, персонаж підкидається вгору, індикатор заряду буста змінюється; коректна робота ефектів без затримок.	Як і очікувалось	Виконано

Автоматичне тестування. Задля перевірки правильності логіки роботи застосунку, та логіки генерації платформ, ворогів та бустів було додано автоматичні тести.

```
import { createPlatform } from '../gameLogic';

test('static platform does not move on update()', () => {
  const p = createPlatform(10, 20, 100, 'static');
  const xBefore = p.x;
  p.update();
  expect(p.x).toBe(xBefore);
});
```

У цьому тесті ми перевіряємо поведінку платформи з типом `static`, тобто такої, що за задумом ніколи не повинна змінювати своєї позиції. Спочатку викликаємо `createPlatform(10, 20, 100, 'static')`, тобто створюємо платформу з координатами `x=10`, `y=20` та шириною 100 пікселів. Поле `p.x` після створення зберігає значення 10, яке ми фіксуємо в змінній `xBefore`. Коли виконуємо `p.update()`, внутрішня логіка обробки руху для статичної платформи не повинна нічого змінювати – жоден рядок коду не змінює `p.x`. Нарешті, у твердженні `expect(p.x).toBe(xBefore)` перевіряємо, що позиція залишилася незмінною, і таким чином підтверджуємо, що метод `update()` не вносить небажаних зсувів для статичних об'єктів.

```
import { createPlatform } from '../gameLogic';

test('moving platform reverses direction when exceeding range', () => {
  const p = createPlatform(0, 0, 50, 'moving');
  p.startX = 0;
  p.moveRange = 10;
  p.moveSpeed = 5;
  p.moveDirection = 1;

  p.update();
  expect(p.x).toBe(5);

  p.update();
```

```

expect(p.moveDirection).toBe(-1);
expect(p.x).toBe(10);

p.update();
expect(p.x).toBe(5);
});

```

В даному тесті ми імітуємо платформу з типом `moving`, яка рухається взад-перед у межах заданого діапазону. Виклик `createPlatform(0, 0, 50, 'moving')` встановлює початкову позицію `startX = 0` і ширину 50. Далі вручну задаємо властивість `moveRange = 10`, щоб діапазон руху був малим, та `moveSpeed = 5`, щоб у два кроки вийти за межі. Перший виклик `p.update()` додає до `p.x` значення `moveSpeed * moveDirection`, тобто `+5`, що й перевіряємо `expect(p.x).toBe(5)`. Другий виклик зсуває платформу ще на `+5`, досягає повної амплітуди (10), і в цей момент логіка повинна інвертувати `moveDirection` в `-1`, аби наступного разу рухатися у зворотному напрямку. Тому перевірка `expect(p.moveDirection).toBe(-1)` підтверджує коректну зміну напрямку, а `expect(p.x).toBe(10)` – що позиція доходить до межі. Третій виклик `update()` вже враховує новий напрямок і зменшує `p.x` на 5, повертаючи його до 5, як і очікується при віддзеркаленні руху.

```

import { createEnemy } from '../gameLogic';

test('spawn position is centered on platform', () => {
  const dummyPlatform = { x: 0, width: 100 };
  const e = createEnemy(0, 50, dummyPlatform);
  expect(e.x).toBe(dummyPlatform.width / 2 - 15);
  expect(e.y).toBe(50 - 30);
});

```

У цьому тесті ми фокусуємося на функції створення ворога `createEnemy`, яка приймає координату платформи `y = 50` і об'єкт платформи з шириною 100. Відповідно до реалізації, ворог повинен з'явитися строго на її поверхні і по центру: `e.x` обчислюється як половина ширини платформи (50) мінус половина ширини спрайта ворога (15), тобто 35. Поле `e.y` встановлюється на рівні `platformY - enemyHeight`, тобто `50 - 30 = 20`, щоб «притягнути» його зверху

платформи. Тести `expect(e.x).toBe(35)` та `expect(e.y).toBe(20)` гарантують, що позиціювання враховує розміри та координати платформи коректно.

```
import { createEnemy } from '../gameLogic';

test('reverses direction when out of bounds', () => {
  const e = createEnemy(0, 0, { x: 0, width: 20 });
  e.speed = 10;
  e.direction = 1;

  e.update();
  expect(e.direction).toBe(-1);

  const prevX = e.x;
  e.update();
  expect(e.x).toBe(prevX - 10);
});
```

Цей тест перевіряє стратегію ворога при виході за межі платформи. Створюємо платформу шириною лише 20 пікселів, після чого створюємо ворога і призначаємо йому значну швидкість `speed = 10` та напрямок вправо `direction = 1`. Перший виклик `e.update()` додає `speed` до `e.x`, що виводить ворога за межі платформи, – згідно з логікою `update()` він повинен змінити напрямок на зворотний (`direction = -1`). Підтверджуємо це через `expect(e.direction).toBe(-1)`. Потім зберігаємо поточне значення `e.x` у `prevX` і викликаємо `e.update()` вдруге: тепер ворог рухається ліворуч на ту саму відстань, тобто `e.x` має стати `prevX - 10`. Перевірка `expect(e.x).toBe(prevX - 10)` свідчить про коректну обробку інверсії руху та обмежень по краях платформи.

Висновки до третього розділу

У результаті проведеного аналізу інтерфейсу та порядку роботи з веб-застосунком «Endless Jumper» встановлено, що використання Docker-контейнера та розгортання на Azure забезпечують простоту підтримки та масштабування проєкту без додаткових вимог до кінцевого користувача. Завдяки чіткій структурі головного меню з інтуїтивно зрозумілими підказками керування,

адаптивним оформленням і привабливим анімаційним фоном вдається створити комфортне середовище для входження в гру як новачків, так і досвідчених геймерів.

Ручне та автоматичне тестування засвідчило коректну роботу основних сценаріїв взаємодії: від відображення елементів меню до генерації та обробки бонусів у процесі геймплею. Усі перевірені функції відповідали очікуваним результатам, що підтверджує стабільність реалізованих механік.

					ІІЗ.КР.Б-121-25-ІІЗ	
						63

ВИСНОВКИ

В даній дипломній роботі представлено розробку 2D- платформера «Endless Jumper» з використанням веб-технологій.

1. Дослідження включало аналіз існуючих ігор у цьому жанрі, що допомогло визначити ключові особливості та конкурентні переваги «Endless Jumper», такі як його кросплатформна веб-доступність і адаптована схема керування. Архітектурний шаблон Model-View-Controller (MVC) був обраний через його баланс між структурою, простотою та контролем потоку даних, який виявився ідеальним для обсягу проекту.

2. Технічна реалізація використовувала JavaScript, HTML5 Canvas і CSS3 для основної логіки гри та рендерингу, тоді як Firebase надавав надійні серверні служби для таблиць лідерів у реальному часі та анонімної автентифікації, а Microsoft Azure сприяв масштабованому хостингу. Процедурні алгоритми генерації вмісту відіграли важливу роль у створенні нескінченного ігрового світу з динамічними платформами, ворогами та бонусами, забезпечуючи повторність і збалансовану складність.

3. Детальну увагу було приділено моделям пересування та сценаріям взаємодії для всіх ігрових об'єктів, включаючи гравця, ворогів і різні елементи середовища. Інтерфейс користувача був інтуїтивно зрозумілим та адаптивним, із чіткою навігацією, інструкціями в грі та таблицею лідерів у реальному часі. Комплексне ручне та автоматизоване тестування підтвердило стабільність і правильність реалізованої механіки та функціональних вимог, підтвердивши загальну продуктивність гри та досвід користувача.

Зрештою, проект успішно представив готове до розгортання програмне рішення, яке може слугувати навчальним інструментом для вивчення основ розробки ігор і основою для майбутніх розширень або портування на інші платформи.

					ІПЗ.КР.Б-121-25-ПЗ	
						64

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abstracts: Computer games and multimedia as an innovative approach to electronic communication [Електронний ресурс] // ONTU. – 2022. – Режим доступу: <https://ontu.edu.ua/download/konfi/2022/Abstracts-Computer-games-and-multimedia-as-an-innovative-approach-to-electronic-communication-22.pdf>.
2. Iterative prototyping: Secret weapon in game development [Електронний ресурс] // Wayline. – 2023. – Режим доступу: <https://www.wayline.io/blog/iterative-prototyping-secret-weapon-game-development>.
3. Doodle Jump [Електронний ресурс]. – 2024. – Режим доступу: <https://doodle-jump.co>.
4. Mega Jump [Електронний ресурс] // Halfbrick Studios. – 2023. – Режим доступу: <https://www.halfbrick.com/games/megajump>.
5. Helix Jump [Електронний ресурс] // Poki. – 2024. – Режим доступу: <https://poki.com/en/g/helix-jump>.
6. Martin Fowler. Architecture [Електронний ресурс]. – 2023. – Режим доступу: <https://martinfowler.com/architecture/>.
7. Unity Interview Questions [Електронний ресурс] // We Create Problems. – 2023. – Режим доступу: <https://www.wecreateproblems.com/interview-questions/unity-interview-questions>.
8. Why is Entity Component System (ECS) so awesome for game development? [Електронний ресурс] // Medium. – 2023. – Режим доступу: <https://medium.com/source-true/why-is-entity-component-system-ecs-so-awesome-for-game-development-f554e1367c17>.
9. MVP (Model View Presenter) Architecture Pattern in Android with Example [Електронний ресурс] // GeeksforGeeks. – 2023. – Режим доступу: <https://www.geeksforgeeks.org/mvp-model-view-presenter-architecture-pattern-in-android-with-example/>.

10. MVVM (Model View ViewModel) Architecture Pattern in .NET MAUI [Электронный ресурс] // Microsoft Learn. – 2024. – Режим доступа: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>.
11. Difference Between MVC, MVP and MVVM Architecture Pattern in Android [Электронный ресурс] // GeeksforGeeks. – 2024. – Режим доступа: <https://www.geeksforgeeks.org/android/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/>.
12. JavaScript [Электронный ресурс] // MDN Web Docs. – 2024. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.
13. JavaScript [Электронный ресурс] // Wix. – 2024. – Режим доступа: <https://www.wix.com/encyclopedia/definition/javascript>.
14. Вступ до HTML5 [Электронный ресурс] // W3SchoolsUA. – 2024. – Режим доступа: https://w3schoolsua.github.io/html/html5_intro.html.
15. CSS [Электронный ресурс] // MDN Web Docs. – 2024. – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
16. Firebase [Электронный ресурс]. – Google, 2024. – Режим доступа: <https://firebase.google.com>.
17. Firebase Authentication [Электронный ресурс] // Firebase. – 2024. – Режим доступа: <https://firebase.google.com/docs/auth>.
18. Azure [Электронный ресурс] // Microsoft Azure. – 2024. – Режим доступа: <https://azure.microsoft.com/en-us>.
19. Young C. Game Changers: Everyday Gamemakers and the Development of the Video Game Industry [Электронный ресурс] // ResearchGate. – 2021. – Режим доступа: <https://www.researchgate.net/publication/356982414>.
20. Unity Manual [Электронный ресурс] // Unity Technologies. – 2024. – Режим доступа: <https://docs.unity3d.com/Manual/index.html>.
21. Godot Engine [Электронный ресурс]. – 2024. – Режим доступа: <https://godotengine.org>.

22. GameMaker [Електронний ресурс]. – 2024. – Режим доступу: <https://gamemaker.io/en>.
23. Phaser [Електронний ресурс]. – 2024. – Режим доступу: <https://phaser.io>.
24. PixiJS [Електронний ресурс]. – 2024. – Режим доступу: <https://pixijs.com>.
25. Functional and Non-Functional Requirements: Specification and Types [Електронний ресурс] // AltexSoft. – 2024. – Режим доступу: <https://www.altexsoft.com/blog/functional-and-non-functional-requirements-specification-and-types/>.
26. Guide to User Requirements [Електронний ресурс] // QAT Global. – 2024. – Режим доступу: <https://qat.com/guide-user-requirements/>.
27. Firebase Firestore [Електронний ресурс] // Firebase. – 2024. – Режим доступу: <https://firebase.google.com/docs/firestore>.
28. Cloud Firestore [Електронний ресурс] // Google Cloud. – 2024. – Режим доступу: <https://cloud.google.com/products/firestore>.
29. Як стати геймдевелопером у 2023 році? [Електронний ресурс] // DOU. – 2023. – Режим доступу: <https://dou.ua/forums/topic/44058/>.
30. How to avoid Firebase Cloud Functions invocations [Електронний ресурс] // Stack Overflow. – 2024. – Режим доступу: <https://stackoverflow.com/questions/77880224>.

ДОДАТКИ

					ІІЗ.КР.Б-121-25-ІІЗ	
						68

Лістинг index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Endless Jumper Platformer</title>
  <script type="module">
    import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.1/firebase-app.js";
    import { getFirestore, collection, addDoc, query, orderBy, limit, getDocs } from
"https://www.gstatic.com/firebasejs/11.6.1/firebase-firestore.js";
    import { getAuth, signInAnonymously, onAuthStateChanged } from
"https://www.gstatic.com/firebasejs/11.6.1/firebase-auth.js";

    document.addEventListener('DOMContentLoaded', () => {
      // --- Firebase Setup ---
      let db, auth;
      try {
        // Firebase configuration updated as per your request
        const firebaseConfig = {
          apiKey: "AIzaSyCeELYG4yMBxe9GXNkD-GSzig3Iy71Xr7s",
          authDomain: "diploma-for-illya.firebaseio.com",
          projectId: "diploma-for-illya",
          storageBucket: "diploma-for-illya.firebaseio.com",
          messagingSenderId: "933282700060",
          appId: "1:933282700060:web:6ce84c3daa30985eb2b8d9"
        };
        const app = initializeApp(firebaseConfig);
        db = getFirestore(app);
        auth = getAuth(app);

        onAuthStateChanged(auth, (user) => {
          if (!user) {
            signInAnonymously(auth);
          }
        });
      } catch (e) {
        console.error("Error initializing Firebase. Leaderboard features will be disabled.", e);
        document.getElementById('leaderboard-btn').style.display = 'none';
        document.getElementById('leaderboard-btn-gameover').style.display = 'none';
      }

      const canvas = document.getElementById('gameCanvas');
      const ctx = canvas.getContext('2d');

      // --- UI and Screen Elements ---
      const startScreen = document.getElementById('start-screen');
      const startGameButton = document.getElementById('start-game-button');
      const leaderboardBtn = document.getElementById('leaderboard-btn');

```

```

const leaderboardScreen = document.getElementById('leaderboard-screen');
const backToMenuBtn = document.getElementById('back-to-menu-btn');
const leaderboardList = document.getElementById('leaderboard-list');
const gameContainer = document.getElementById('game-container');
const gameOverScreen = document.getElementById('game-over-screen');
const restartButton = document.getElementById('restart-button');
const submitScoreForm = document.getElementById('submit-score-form');
const finalScoreEl = document.getElementById('final-score');
const scoreEl = document.getElementById('score');
const healthEl = document.getElementById('health');
const powerupTimerEl = document.getElementById('powerup-timer');
const boostStatusEl = document.getElementById('boost-status'); // NEW: UI for boost

// --- Game Configuration ---
let canvasWidth = 450;
let canvasHeight = 800;
let gravity = 0.4;

// --- Game State Variables ---
let player, keys, platforms, bullets, enemies, particles, weaponPowerups, boostPowerups,
cameraY, score, health;
let isGameOver = false;
let animationFrameId;
let gameTime = 0;

// --- Asset Management ---
let gameAssets = { player: null, enemy: null, platform: null, bullet: null };
const defaultAssetURLs = {
  player: "data:image/svg+xml,%3csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 30 40'%3e%3crect width='30' height='40' rx='5' fill='%2348bb78'%3e%3crect x='7' y='10' width='5' height='5' fill='white'%3e%3crect x='18' y='10' width='5' height='5' fill='white'%3e%3c/svg%3e",
  enemy: "data:image/svg+xml,%3csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 30 30'%3e%3crect width='30' height='30' rx='5' fill='%23e53e3e'%3e%3crect x='7' y='7' width='5' height='5' fill='black'%3e%3crect x='18' y='7' width='5' height='5' fill='black'%3e%3cpath d='M7 20 Q 15 25 23 20' stroke='black' fill='none' stroke-width='2'%3e%3c/svg%3e",
  platform: "data:image/svg+xml,%3csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 100 20'%3e%3crect width='100' height='20' rx='10' ry='10' fill='%23a0aec0'%3e%3c/svg%3e",
  bullet: "data:image/svg+xml,%3csvg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 10 20'%3e%3cpath d='M5 0 L10 10 L5 20 L0 10 Z' fill='%23f6e05e'%3e%3c/svg%3e"
};
keys = { right: false, left: false, shoot: false };

const playerProto = {
  width: 30, height: 40, speed: 4, jumpForce: 12, dx: 0, dy: 0, onGround: false,
  shootCooldown: 0, baseShootCooldown: 20, powerupTimer: 0, hasBoost: false, // NEW:
  hasBoost property

  draw() {

```

```

        if (gameAssets.player) {
            ctx.drawImage(gameAssets.player, this.x, this.y - cameraY, this.width, this.height);
        }
    },
    update() {
        this.dx = 0;
        if (keys.right) this.dx = this.speed;
        if (keys.left) this.dx = -this.speed;
        this.x += this.dx;

        this.dy += gravity;
        this.y += this.dy;

        if (this.x < 0) this.x = 0;
        if (this.x + this.width > canvasWidth) this.x = canvasWidth - this.width;

        const currentCooldown = this.powerupTimer > 0 ? this.baseShootCooldown / 2 :
this.baseShootCooldown;
        if (this.shootCooldown > 0) this.shootCooldown--;
        if (keys.shoot && this.shootCooldown === 0) {
            this.shoot();
            this.shootCooldown = currentCooldown;
        }
        if (this.powerupTimer > 0) this.powerupTimer--;
    },
    shoot() {
        bullets.push({ x: this.x + this.width / 2 - 2.5, y: this.y + 15, width: 5, height: 10, dy: -10
    });
    },
    activatePowerup() {
        this.powerupTimer = 300; // 5 seconds at 60fps
    }
    };

function createPlatform(x, y, width, type = 'static') {
    return {
        x, y, width, height: 20, type,
        moveSpeed: (Math.random() * 0.5 + 0.5), moveDirection: 1, startX: x, moveRange:
Math.random() * 80 + 50,
        draw() {
            ctx.fillStyle = this.type === 'moving' ? '#63b3ed' : '#a0aec0';
            const radius = 10;
            ctx.beginPath();
            ctx.moveTo(this.x + radius, this.y - cameraY);
            ctx.lineTo(this.x + this.width - radius, this.y - cameraY);
            ctx.quadraticCurveTo(this.x + this.width, this.y - cameraY, this.x + this.width, this.y
- cameraY + radius);
            ctx.lineTo(this.x + this.width, this.y - cameraY + this.height - radius);
            ctx.quadraticCurveTo(this.x + this.width, this.y - cameraY + this.height, this.x +
this.width - radius, this.y - cameraY + this.height);
            ctx.lineTo(this.x + radius, this.y - cameraY + this.height);

```

```

        ctx.quadraticCurveTo(this.x, this.y - cameraY + this.height, this.x, this.y - cameraY
+ this.height - radius);
        ctx.lineTo(this.x, this.y - cameraY + radius);
        ctx.quadraticCurveTo(this.x, this.y - cameraY, this.x + radius, this.y - cameraY);
        ctx.closePath();
        ctx.fill();
    },
    update() {
        if (this.type === 'moving') {
            this.x += this.moveSpeed * this.moveDirection;
            if (Math.abs(this.x - this.startX) >= this.moveRange / 2) {
                this.moveDirection *= -1;
            }
        }
    }
};
}

function createEnemy(x, y, platform) {
    return {
        x: x + platform.width / 2 - 15, y: y - 30, width: 30, height: 30, speed: 1, direction: 1,
platform,
        draw() {
            if (gameAssets.enemy) {
                ctx.drawImage(gameAssets.enemy, this.x, this.y - cameraY, this.width,
this.height);
            }
        },
        update() {
            this.x += this.speed * this.direction;
            if (this.x < this.platform.x || this.x + this.width > this.platform.x +
this.platform.width) {
                this.direction *= -1;
            }
        }
    };
}

function createWeaponPowerup(x, y) {
    return {
        x, y, width: 20, height: 20,
        draw() {
            ctx.save();
            ctx.translate(this.x + this.width / 2, (this.y - cameraY) + this.height / 2);
            ctx.rotate(gameTime * 0.1);
            ctx.fillStyle = '#f6e05e';
            ctx.beginPath();
            for (let i = 0; i < 5; i++) {
                ctx.lineTo(Math.cos(i * 2 * Math.PI / 5 * 2) * (this.width/2), Math.sin(i * 2 *
Math.PI / 5 * 2) * (this.width/2));
            }
            ctx.closePath();
            ctx.fill();
            ctx.restore();
        }
    };
}

```

```

        ctx.lineTo(Math.cos((i * 2 + 1) * Math.PI / 5 * 2) * (this.width/4), Math.sin((i * 2
+ 1) * Math.PI / 5 * 2) * (this.width/4));
    }
    ctx.closePath();
    ctx.fill();
    ctx.restore();
}
}
}

// --- NEW: Function to create a boost power-up item ---
function createBoostPowerup(x, y) {
    return {
        x, y, width: 20, height: 20,
        draw() {
            ctx.fillStyle = '#48bb78'; // Green color for boost
            ctx.beginPath();
            ctx.arc(this.x + this.width / 2, (this.y - cameraY) + this.height / 2, this.width / 2, 0,
Math.PI * 2);
            ctx.fill();
            // Draw an upward arrow
            ctx.fillStyle = 'white';
            ctx.beginPath();
            ctx.moveTo(this.x + this.width / 2, this.y - cameraY + 4);
            ctx.lineTo(this.x + this.width - 4, this.y - cameraY + this.height / 2 + 2);
            ctx.lineTo(this.x + 4, this.y - cameraY + this.height / 2 + 2);
            ctx.closePath();
            ctx.fill();
        }
    }
}

function createParticle(x, y, color) {
    const size = Math.random() * 5 + 2;
    return {
        x, y, size, color, life: 60,
        dx: (Math.random() - 0.5) * 4, dy: (Math.random() - 0.5) * 4,
        draw() {
            ctx.fillStyle = this.color;
            ctx.globalAlpha = this.life / 60;
            ctx.fillRect(this.x, this.y - cameraY, this.size, this.size);
            ctx.globalAlpha = 1.0;
        },
        update() {
            this.x += this.dx;
            this.y += this.dy;
            this.life--;
        }
    }
}
}

```

```

function init() {
    isGameOver = false;
    gameOverScreen.style.display = 'none';
    submitScoreForm.style.display = 'block';
    submitScoreForm.innerHTML = `<input type="text" id="player-name-input"
placeholder="Enter Your Name"><button type="submit" class="btn">Submit Score</button>`;

    player = Object.create(playerProto);
    player.x = canvasWidth / 2 - player.width / 2;
    player.y = canvasHeight - 100;
    player.powerupTimer = 0;
    player.hasBoost = false; // Reset boost on init

    cameraY = 0;
    score = 0;
    health = 100;

    platforms = []; bullets = []; enemies = []; particles = []; weaponPowerups = [];
    boostPowerups = []; // Init boost array
    platforms.push(createPlatform(0, canvasHeight - 50, canvasWidth, 'static'));
    for (let i = canvasHeight - 200; i > -200; i -= 80) {
        generatePlatform(i);
    }

    updateUI();

    if (animationFrameId) cancelAnimationFrame(animationFrameId);
    gameLoop();
}

function generatePlatform(yPos) {
    const width = Math.random() * (120 - 60) + 60;
    const x = Math.random() * (canvasWidth - width);
    const type = Math.random() < 0.25 ? 'moving' : 'static';
    const newPlatform = createPlatform(x, yPos, width, type);

    if (type === 'moving') {
        newPlatform.startX = Math.max(newPlatform.moveRange / 2, Math.min(x,
canvasWidth - width - newPlatform.moveRange / 2));
        newPlatform.x = newPlatform.startX;
    }

    platforms.push(newPlatform);

    const hasEnemy = Math.random() < 0.2 && type === 'static';
    const hasWeapon = Math.random() < 0.1;
    const hasBoost = Math.random() < 0.08; // NEW: Chance to spawn boost

    if (hasEnemy) {
        enemies.push(createEnemy(x, yPos, newPlatform));
    } else if (hasWeapon) {

```

```

        weaponPowerups.push(createWeaponPowerup(x + width/2 - 10, yPos - 25));
    } else if (hasBoost) { // NEW: Spawn boost if conditions met
        boostPowerups.push(createBoostPowerup(x + width/2 - 10, yPos - 25));
    }
}

function update() {
    if (isGameOver) return;
    gameTime++;
    player.update();
    platforms.forEach(p => p.update());

    player.onGround = false;
    platforms.forEach(p => {
        if (player.dy > 0 &&
            player.x < p.x + p.width && player.x + player.width > p.x &&
            player.y + player.height > p.y && player.y + player.height < p.y + p.height +
player.dy) {

            // --- NEW: Check for boost and apply it on jump ---
            let jumpMultiplier = 1;
            if (player.hasBoost) {
                jumpMultiplier = 2;
                player.hasBoost = false;
                for (let i = 0; i < 20; i++) particles.push(createParticle(player.x + player.width / 2,
player.y + player.height, '#48bb78'));
            }

            player.dy = -player.jumpForce * jumpMultiplier;
            player.y = p.y - player.height;
            player.onGround = true;

            if (p.type === 'moving') {
                player.x += p.moveSpeed * p.moveDirection;
            }

            for (let i = 0; i < 5; i++) particles.push(createParticle(player.x + player.width / 2, p.y,
'#cbd5e0'));
        }
    });

    if (player.y - cameraY < canvasHeight / 3) {
        cameraY = player.y - canvasHeight / 3;
    }
    const currentScore = Math.floor(-cameraY / 10);
    if (currentScore > score) { score = currentScore; }

    const topPlatformY = platforms[platforms.length - 1].y;
    if (topPlatformY > cameraY - 100) { generatePlatform(topPlatformY - 80); }

    platforms = platforms.filter(p => p.y < cameraY + canvasHeight + 50);

```

```

enemies = enemies.filter(e => e.platform.y < cameraY + canvasHeight + 50);
weaponPowerups = weaponPowerups.filter(wp => wp.y < cameraY + canvasHeight +
50);
boostPowerups = boostPowerups.filter(bp => bp.y < cameraY + canvasHeight + 50); //
Clean up boosts

bullets.forEach((b, i) => {
    b.y += b.dy;
    if (b.y < cameraY) bullets.splice(i, 1);
});

enemies.forEach(e => e.update());

bullets.forEach((bullet, bIndex) => {
    enemies.forEach((enemy, eIndex) => {
        if (bullet.x < enemy.x + enemy.width && bullet.x + bullet.width > enemy.x &&
            bullet.y < enemy.y + enemy.height && bullet.y + bullet.height > enemy.y) {
            for (let i=0; i<15; i++) particles.push(createParticle(enemy.x+enemy.width/2,
enemy.y+enemy.height/2, '#f56565'));
            enemies.splice(eIndex, 1);
            bullets.splice(bIndex, 1);
            score += 50;
        }
    });
});

enemies.forEach(enemy => {
    if (player.x < enemy.x + enemy.width && player.x + player.width > enemy.x &&
        player.y < enemy.y + enemy.height && player.y + player.height > enemy.y) {
        health--;
        for (let i=0; i<5; i++) particles.push(createParticle(player.x+player.width/2,
player.y+player.height/2, '#f6e05e'));
        if (health <= 0) endGame();
    }
});

weaponPowerups.forEach((wp, index) => {
    if (player.x < wp.x + wp.width && player.x + player.width > wp.x &&
        player.y < wp.y + wp.height && player.y + player.height > wp.y) {
        player.activatePowerup();
        weaponPowerups.splice(index, 1);
        for (let i=0; i<20; i++) particles.push(createParticle(wp.x+wp.width/2,
wp.y+wp.height/2, '#f6e05e'));
    }
});

// --- NEW: Collision detection for boost power-ups ---
boostPowerups.forEach((bp, index) => {
    if (player.x < bp.x + bp.width && player.x + player.width > bp.x &&
        player.y < bp.y + bp.height && player.y + player.height > bp.y) {
        player.hasBoost = true;
    }
});

```

```

        boostPowerups.splice(index, 1);
        for (let i=0; i<20; i++) particles.push(createParticle(bp.x+bp.width/2,
bp.y+bp.height/2, '#48bb78'));
    }
});

particles.forEach((p, index) => {
    p.update();
    if (p.life <= 0) particles.splice(index, 1);
});

if (player.y > cameraY + canvasHeight) { endGame(); }
updateUI();
}

function draw() {
    ctx.clearRect(0, 0, canvasWidth, canvasHeight);
    ctx.fillStyle = 'rgba(255, 255, 255, 0.5)';
    for (let i=0; i<100; i++) {
        const x = (i * 37) % canvasWidth;
        let y = ((i * 89) - cameraY * 0.1) % canvasHeight;
        if(y < 0) y += canvasHeight;
        ctx.fillRect(x, y, 2, 2);
    }
    platforms.forEach(p => p.draw());
    enemies.forEach(e => e.draw());
    weaponPowerups.forEach(wp => wp.draw());
    boostPowerups.forEach(bp => bp.draw()); // NEW: Draw boost power-ups
    bullets.forEach(b => {
        if (gameAssets.bullet) {
            ctx.drawImage(gameAssets.bullet, b.x, b.y - cameraY, b.width * 1.5, b.height * 1.5);
        }
    });
    particles.forEach(p => p.draw());
    player.draw();
}

function updateUI() {
    scoreEl.textContent = `Score: ${score}`;
    healthEl.textContent = `Health: ${Math.max(0, health)}`;
    if (player && player.powerupTimer > 0) {
        powerupTimerEl.textContent = `Rapid Fire: ${(player.powerupTimer /
60).toFixed(1)}s`;
        powerupTimerEl.style.display = 'block';
    } else {
        powerupTimerEl.style.display = 'none';
    }
    // --- NEW: Update boost status UI ---
    if (player && player.hasBoost) {
        boostStatusEl.style.display = 'block';
    } else {

```

```

        boostStatusEl.style.display = 'none';
    }
}

function endGame() {
    if (isGameOver) return;
    isGameOver = true;
    cancelAnimationFrame(animationFrameId);
    finalScoreEl.textContent = `Your Score: ${score}`;
    gameOverScreen.style.display = 'flex';
    gameOverScreen.style.flexDirection = 'column';
}

function gameLoop() {
    if (isGameOver) return;
    update();
    draw();
    animationFrameId = requestAnimationFrame(gameLoop);
}

function resizeCanvas() {
    const aspectRatio = 9 / 16;
    const availableWidth = Math.min(window.innerWidth * 0.95, 500);
    const availableHeight = window.innerHeight * 0.9;
    let newWidth, newHeight;
    if (availableWidth / availableHeight > aspectRatio) {
        newHeight = availableHeight; newWidth = newHeight * aspectRatio;
    } else {
        newWidth = availableWidth; newHeight = newWidth / aspectRatio;
    }
    canvas.width = newWidth; canvas.height = newHeight;
    canvasWidth = canvas.width; canvasHeight = canvas.height;
}

function loadAllDefaultAssets() {
    return Promise.all(Object.keys(defaultAssetURLs).map(name => {
        return new Promise((resolve, reject) => {
            const img = new Image();
            img.onload = () => { gameAssets[name] = img; resolve(); };
            img.onerror = () => reject(`Failed to load ${name}`);
            img.src = defaultAssetURLs[name];
        });
    }));
}

async function showLeaderboard() {
    startScreen.style.display = 'none'; gameOverScreen.style.display = 'none';
    leaderboardScreen.style.display = 'flex';
    leaderboardList.innerHTML = '<li>Loading...</li>';

    if (!db) { leaderboardList.innerHTML = '<li>Leaderboard is unavailable.</li>'; return; }
}

```

```

    try {
      const q = query(collection(db, "scores"), orderBy("score", "desc"), limit(10));
      const querySnapshot = await getDocs(q);
      leaderboardList.innerHTML = "";
      if (querySnapshot.empty) { leaderboardList.innerHTML = '<li>No scores yet. Be the
first!</li>'; return; }
      let rank = 1;
      querySnapshot.forEach((doc) => {
        const data = doc.data();
        const li = document.createElement('li');
        li.innerHTML = `<span>${rank}</span>
${data.name}</span><span>${data.score}</span>`;
        leaderboardList.appendChild(li);
        rank++;
      });
    } catch (error) {
      console.error("Error getting leaderboard: ", error);
      leaderboardList.innerHTML = '<li>Could not load scores.</li>';
    }
  }

window.addEventListener('resize', resizeCanvas);
window.addEventListener('keydown', e => {
  if (e.code === 'ArrowRight' || e.code === 'KeyD') keys.right = true;
  if (e.code === 'ArrowLeft' || e.code === 'KeyA') keys.left = true;
  if (e.code === 'Space') keys.shoot = true;
});
window.addEventListener('keyup', e => {
  if (e.code === 'ArrowRight' || e.code === 'KeyD') keys.right = false;
  if (e.code === 'ArrowLeft' || e.code === 'KeyA') keys.left = false;
  if (e.code === 'Space') keys.shoot = false;
});
restartButton.addEventListener('click', init);
leaderboardBtn.addEventListener('click', showLeaderboard);
backToMenuBtn.addEventListener('click', () => {
  leaderboardScreen.style.display = 'none';
  startScreen.style.display = 'flex';
});
document.getElementById('leaderboard-btn-gameover').addEventListener('click',
showLeaderboard);

submitScoreForm.addEventListener('submit', async (e) => {
  e.preventDefault();
  if (!db) { alert('Cannot submit score, database not connected.');
```

```

        await addDoc(collection(db, "scores"), { name: playerName, score: score, createdAt:
new Date() });
        submitScoreForm.innerHTML = '<p>Score submitted!</p>';
    } catch (error) {
        console.error("Error adding document: ", error);
        submitBtn.textContent = 'Try Again'; submitBtn.disabled = false;
    }
});

const leftBtn = document.getElementById('left-btn');
const rightBtn = document.getElementById('right-btn');
const shootBtn = document.getElementById('shoot-btn');
leftBtn.addEventListener('touchstart', e => { e.preventDefault(); keys.left = true; }, {
passive: false });
leftBtn.addEventListener('touchend', e => { e.preventDefault(); keys.left = false; }, {
passive: false });
rightBtn.addEventListener('touchstart', e => { e.preventDefault(); keys.right = true; }, {
passive: false });
rightBtn.addEventListener('touchend', e => { e.preventDefault(); keys.right = false; }, {
passive: false });
shootBtn.addEventListener('touchstart', e => { e.preventDefault(); keys.shoot = true; }, {
passive: false });
shootBtn.addEventListener('touchend', e => { e.preventDefault(); keys.shoot = false; }, {
passive: false });

startGameButton.addEventListener('click', () => {
    startGameButton.textContent = 'Loading...'; startGameButton.disabled = true;
    loadAllDefaultAssets().then(() => {
        startScreen.style.display = 'none'; gameContainer.style.display = 'flex';
        resizeCanvas(); init();
    }).catch(error => {
        console.error("Asset loading failed:", error);
        startGameButton.textContent = 'Error - Refresh';
        alert("Could not load game assets. Please refresh the page to try again.");
    });
});
</script>
<style>
    body { margin: 0; padding: 0; display: flex; justify-content: center; align-items: center; min-
height: 100vh; background-color: #1a202c; font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif; color: white; flex-direction: column; overflow: hidden; }
    canvas { background-color: #2d3748; border-radius: 8px; box-shadow: 0 10px 25px rgba(0, 0,
0, 0.5); display: block; }
    .screen { display: none; flex-direction: column; align-items: center; justify-content: center;
gap: 1rem; background: rgba(26, 32, 44, 0.95); padding: 40px; border-radius: 16px; text-align:
center; box-shadow: 0 10px 25px rgba(0,0,0,0.5); width: 90%; max-width: 450px; z-index: 20; }
    #start-screen { display: flex; }
    .screen h1 { margin: 0; font-size: 3rem; color: #48bb78; }
    .screen h2 { margin-top: 0; font-size: 3rem; }

```

```

#game-container { position: relative; display: none; flex-direction: column; align-items: center;
justify-content: center; }
#controls { display: none; margin-top: 10px; gap: 10px; }
.control-btn { padding: 15px 25px; font-size: 20px; border: none; border-radius: 8px;
background-color: #4a5568; color: white; cursor: pointer; user-select: none; box-shadow: 0 4px 6px
rgba(0, 0, 0, 0.3); }
.control-btn:active { background-color: #2d3748; }

#ui-container { position: absolute; top: 20px; left: 50%; transform: translateX(-50%); display:
flex; flex-wrap: wrap; justify-content: center; gap: 10px 20px; background: rgba(45, 55, 72, 0.8);
padding: 10px 20px; border-radius: 12px; pointer-events: none; z-index: 10; }
#score, #health, #powerup-timer, #boost-status { font-size: 1.2rem; font-weight: bold; white-
space: nowrap; }
#powerup-timer { color: #f6e05e; }
#boost-status { color: #48bb78; display: none; } /* NEW: Style for boost status */

#game-over-screen { position: absolute; top: 50%; left: 50%; transform: translate(-50%, -
50%); }
.btn, #restart-button, #start-game-button { padding: 15px 30px; font-size: 1.5rem; border:
none; border-radius: 8px; background-color: #48bb78; color: white; cursor: pointer; margin-top:
10px; transition: background-color 0.3s; width: 100%; max-width: 300px; }
.btn:hover, #restart-button:hover, #start-game-button:hover { background-color: #38a169; }
.btn:disabled { background-color: #4a5568; cursor: not-allowed; }
.btn.secondary { background-color: #4299e1; }
.btn.secondary:hover { background-color: #3182ce; }

#leaderboard-list { list-style: none; padding: 0; width: 100%; }
#leaderboard-list li { display: flex; justify-content: space-between; background: #4a5568;
margin-bottom: 8px; padding: 10px 15px; border-radius: 5px; font-size: 1.2rem; }

#submit-score-form { display: block; margin-top: 20px; width: 100%; }
#player-name-input { font-size: 1.2rem; padding: 10px; border-radius: 8px; border: 2px solid
#4a5568; background: #2d3748; color: white; text-align: center; width: calc(100% - 24px); margin-
bottom: 10px; }
#submit-score-form button { width: 100%; }

@media (hover: none) and (pointer: coarse) {
#controls { display: flex; }
#start-screen p { display: none; }
}
</style>
</head>
<body>

<div id="start-screen" class="screen">
<h1>Endless Jumper</h1>
<p>Use A/D or Arrow Keys to move, Space to shoot.</p>
<button id="start-game-button" class="btn">Start Game</button>
<button id="leaderboard-btn" class="btn secondary">High Scores</button>
</div>

```

```

<div id="leaderboard-screen" class="screen">
  <h2>High Scores</h2>
  <ul id="leaderboard-list"></ul>
  <button id="back-to-menu-btn" class="btn">Back to Menu</button>
</div>

<div id="game-container">
  <div id="ui-container">
    <div id="score">Score: 0</div>
    <div id="health">Health: 100</div>
    <div id="powerup-timer"></div>
    <div id="boost-status">Boost Ready!</div> <!-- NEW: UI element for boost -->
  </div>
  <canvas id="gameCanvas"></canvas>
  <div id="controls">
    <button class="control-btn" id="left-btn">Left</button>
    <button class="control-btn" id="shoot-btn">Shoot</button>
    <button class="control-btn" id="right-btn">Right</button>
  </div>
  <div id="game-over-screen" class="screen">
    <h2>Game Over</h2>
    <p id="final-score"></p>
    <form id="submit-score-form">
      <input type="text" id="player-name-input" placeholder="Enter Your Name">
      <button type="submit" class="btn">Submit Score</button>
    </form>
    <button id="restart-button" class="btn">Play Again</button>
    <button id="leaderboard-btn-gameover" class="btn secondary">High Scores</button>
  </div>
</div>
</body>
</html>

```