МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ 2D ГРИ ТАНКОВИЙ БІЙ

Виконав: студент групи 2К-21
Спеціальності 123 Комп’ютерна інженерія
Максим МАРТИНЕНКО
Керівник:
Дмитро ПОДОРОШКО

Черкаси 2025

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному цифровому середовищі браузерні ігри набувають популярності завдяки своїй доступності, відсутності потреби в установці та кросплатформеності. Створення повноцінного клієнтського функціоналу гри з використанням лише стандартних web-технологій (HTML5, CSS3, JavaScript) є актуальним напрямом, який дозволяє поєднувати навчальні, розважальні й комерційні аспекти. Реалізація подібних застосунків відкриває широкі можливості для навчання веброзробці, розробки ігрових механік та забезпечує високу швидкість розгортання гри серед користувачів.

Об’єктом дослідження є клієнтська частина 2D гри як частина браузерного застосунку.

Предмет дослідження. Інтерфейс, ігрова логіка, обробка подій та рендеринг у Canvas API на основі HTML5 і JavaScript.

Метою роботи є розробка клієнтської частини 2D гри «танковий бій» з використанням Canvas API для рендерингу графіки та реалізації основних ігрових механік у середовищі веббраузера.

Для досягнення мети роботи було виконано наступні завдання:

- Проаналізувати сучасні технології для створення 2D ігор у web-середовищі.
- Спроєктувати архітектуру клієнтської частини гри.
- Реалізувати механіки: рух, стрільба, колізії, HUD, таймер, музика.
- Провести тестування гри у браузері та виконати оптимізацію.
- Надати пропозиції щодо подальшого розвитку гри.

У процесі виконання роботи були використані такі методи:

- аналіз програмних аналогів – вивчення існуючих реалізацій 2D-ігор схожого жанру для визначення найбільш ефективних підходів до структурування клієнтської частини, управління інтерфейсом та взаємодії з користувачем;

- архітектурне проєктування – побудова логічної структури клієнтського застосунку, визначення функціональних модулів та їхньої взаємодії, з урахуванням принципів масштабованості та повторного використання коду;
- розробка за модульним принципом – реалізація функціоналу у вигляді незалежних компонентів (рух, стрільба, HUD, інтерфейс профілю тощо), що забезпечує гнучкість і простоту подальшого супроводу;
- тестування – перевірка працездатності ключових механік гри в типових і граничних сценаріях, включно з обробкою подій, завантаженням ресурсів та взаємодією з DOM;
- візуальний аналіз результатів – оцінка правильності відображення графіки, елементів інтерфейсу, анімації та поведінки гри відповідно до заданих умов;
- налагодження продуктивності – оптимізація обчислень, рендерингу та обробки подій для забезпечення стабільної роботи гри в браузері з мінімальними затримками.

РОЗДІЛ 1
ОГЛЯД ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ 2D-ІГОР

1.1. Види 2D-ігор та вимоги до клієнтської частини

Двовимірні (2D) ігри є ігровими застосунками, в яких простір моделюється у двох вимірах – по горизонтальній (X) та вертикальній (Y) осях. Незважаючи на широке впровадження тривимірної графіки, 2D-ігри продовжують займати значне місце в індустрії цифрових розваг. Їх популярність пояснюється низькими апаратними вимогами, швидкістю розробки, простотою реалізації та особливою привабливістю стилістики піксельної графіки. Крім того, 2D-ігри часто використовуються в освітніх цілях або як інди-проєкти, що дає змогу реалізовувати ідеї без необхідності у складних рушіях або тривимірному моделюванні [1].

Залежно від жанрових особливостей та характеру ігрового процесу, 2D-ігри класифікують на такі основні види:

- платформери (platformers) – ігри, у яких персонаж переміщується по рівнях з перешкодами та платформами (наприклад, Super Mario Bros);
- покрокові та реального часу стратегії (turn-based / real-time strategies) – ігри, що оперують розміщенням і переміщенням об'єктів у площині (наприклад, Advance Wars);
- аркадні ігри (arcades) – жанр з динамічним геймплеєм і простою механікою (наприклад, Space Invaders);
- рольові 2D-ігри (2D RPGs) – ігри з сюжетною структурою, прокачуванням персонажів та бойовою системою, здебільшого у псевдотривимірному середовищі (Chrono Trigger);
- пазли та головоломки (puzzle games) – логічні ігри з фіксованою або поступово ускладненою механікою (Tetris, Sokoban);
- шутери (shooters) – переважно скролингові або фіксовані, де гравець стріляє у ворогів (Metal Slug, Enter the Gungeon);

- симулятори та економічні ігри – з фокусом на управління ресурсами або об'єктами, часто мають інтерфейс, орієнтований на точність та ефективність (наприклад, Game Dev Tycoon).

Серед поширених жанрів 2D-ігор варто виокремити платформери, стрілялки (shoot 'em up), логічні ігри, рольові пригодницькі ігри (RPG), файтинги та ігри типу tower defense. Кожен із зазначених типів має свої особливості реалізації механік, сценаріїв взаємодії з гравцем та побудови інтерфейсу, однак усі вони можуть бути реалізовані засобами веброзробки. Зокрема, для ігор жанру «танковий бій» характерні такі елементи, як рух в обмеженому просторі, стрільба, зіткнення об’єктів, а також базові елементи штучного інтелекту для управління супротивниками [2].

Клієнтська частина (frontend) 2D-гри виконує критичну функцію відображення, обробки користувацького введення та забезпечення зворотного зв’язку в режимі реального часу. Розрізняють наступні вимоги до клієнтської частини:

1. Функціональні вимоги:

- рендеринг графіки – підтримка ефективної побудови спрайтів, анімацій, ефектів на основі тайлів або об'єктів;
- обробка введення – реакція на клавіатуру, мишу або дотик із мінімальною затримкою;
- ігрова логіка – реалізація станів гри (меню, геймплей, пауза тощо) у вигляді окремих модулів або класів;
- взаємодія з бекендом або мережею (якщо присутній мультиплеєр або онлайн-дані).

2. Нефункціональні вимоги:

- продуктивність – стабільне оновлення екрану зі швидкістю щонайменше 30 FPS, оптимізоване споживання ресурсів (CPU, RAM);
- масштабованість інтерфейсу – адаптація до різних роздільностей екрана, підтримка співвідношення сторін.

- Портативність – підтримка декількох операційних систем (якщо використовуються кросплатформенні бібліотеки).
- Безперебійність – відсутність артефактів або некоректного перемальовування, обмеження зайвого перерендерення.

Таким чином, розробка клієнтської частини 2D-ігри для браузера потребує дотримання низки функціональних і нефункціональних вимог. Передусім, необхідно забезпечити стабільну роботу гри з частотою кадрів не нижче 30 FPS навіть на малопотужних пристроях. Це досягається шляхом оптимального використання графічного API (зокрема Canvas), мінімізації роботи з DOM та ефективного управління ресурсами [3].

Клієнтська частина має бути побудована за модульним принципом, де кожен функціональний компонент (логіка руху, рендеринг, обробка подій, графічний інтерфейс) реалізується окремо. Такий підхід спрощує тестування, підтримку та масштабування гри. Інтерфейс користувача має бути адаптивним, тобто коректно відображатися на різних типах пристроїв – десктопах, планшетах, смартфонах.

Особливу увагу слід приділяти часу завантаження. Оптимізація ваги графічних ресурсів, скриптів і звукових файлів є критичною для зниження затримки перед початком гри. Сучасні браузери дозволяють реалізовувати асинхронне завантаження ресурсів, що позитивно впливає на користувацький досвід [4].

Не менш важливою є коректна обробка введення з клавіатури, миші або сенсорного екрана. Взаємодія з користувачем повинна бути максимально чутливою та без затримок. Canvas API використовується для динамічного рендерингу ігрових сцен, включаючи переміщення об’єктів, анімацію вибухів і трасування снарядів. При цьому код має містити обробку виняткових ситуацій, що запобігає аварійному завершенню програми або її зависанню.

Отже, клієнтська частина браузерної 2D-ігри виконує ключову роль у створенні взаємодії між користувачем та ігровим середовищем. Її реалізація вимагає не лише знання інструментів веброзробки, а й розуміння принципів

побудови ігрової архітектури, оптимізації та забезпечення надійності програмного забезпечення.

1.2. Аналіз інструментів для розробки web-ігор (HTML, CSS, JS)

Розробка клієнтської частини браузерної гри передбачає використання перевірених вебтехнологій, що забезпечують кросплатформеність, інтерактивність та продуктивність. Найбільш доцільним у цьому контексті є застосування базового стеку, що включає HTML5, CSS3 та JavaScript. Дані технології підтримуються всіма сучасними браузерами, мають відкриту специфікацію та забезпечують достатній рівень функціональності для реалізації повноцінного ігрового інтерфейсу без залучення додаткових рушіїв або фреймворків [5].

HTML5 є базовим інструментом для структурування вмісту сторінки та визначення її логічних зон. У контексті розробки гри HTML забезпечує побудову структури інтерфейсу: розміщення ігрового полотна (canvas), елементів керування, панелі стану (HUD), профілю гравця, музичного плеєра та модальних вікон. Особливу цінність для ігрової розробки мають нові можливості HTML5, серед яких:

- Canvas API – для рендерингу растрової графіки;
- Audio API – для інтеграції звукових ефектів і фонової музики;
- Web Storage API – для збереження даних гри без потреби в сервері;
- Form validation – для перевірки введених даних без скриптів.

Також важливим є використання семантичних тегів (наприклад, <section>, <article>, <nav>), що підвищують логічну цілісність проєкту, а також полегшують підтримку та масштабування інтерфейсу [6].

CSS3 виконує функцію стилізації HTML-структури та візуального оформлення.

У межах розробки гри використано сучасні можливості CSS, включаючи:

- градієнти, тіні, неонове підсвічування – для стилізації інтерфейсу в аркадному стилі;
- медіа-запити (media queries) – для адаптивності інтерфейсу під різні типи екранів;
- анімовані переходи та трансформації (transition, transform) – для плавної зміни станів елементів;
- @keyframes – для створення циклічної анімації (напр., рух фону або миготіння кнопок).

Для підтримки структурованості проєкту стилі розділено на окремі файли відповідно до логіки: стилі головної сторінки (main.css), HUD (hud.css), меню рівнів (levels.css) тощо. Така модульна організація покращує читабельність коду та спрощує його обслуговування [7].

JavaScript реалізує всю функціональну логіку гри, включаючи:

- ігровий цикл з оновленням стану об’єктів у реальному часі;
- обробку подій користувача (натискання клавіш, клік миші, touch-сигнали);
- анімацію, рух, зіткнення, створення снарядів;
- динамічне оновлення HUD (кількість ворогів, життів, час гри);
- керування модальними вікнами, меню рівнів, паузою, звуком.

Код організовано за модульним принципом:

- main.js – стартова ініціалізація гри та загальний менеджмент;
- movement.js – логіка руху гравця;
- enemy.js – обробка ворогів і зіткнень;
- workinghud.js – взаємодія з інтерфейсом;
- stages.js – завантаження та перемикання рівнів.

Особливу увагу приділено роботі з DOM API, через який JavaScript керує динамічними елементами сторінки, вмикає та вимикає вікна, змінює класи стилів, а також ініціює збереження даних через localStorage. Взаємодія із canvas

реалізована через контекст 2D, де викликаються методи малювання та очищення сцени.

Крім базових можливостей, JavaScript у даному проєкті дозволяє реалізувати:

- відкладене завантаження ресурсів (lazy loading);
- оптимізацію через кешування зображень;
- контроль частоти оновлення сцени (через requestAnimationFrame) [8].

Завдяки сумісному використанню HTML, CSS та JavaScript вдалося реалізувати повноцінну, візуально привабливу та технічно ефективну клієнтську частину гри. Такий підхід є оптимальним для проєктів середнього рівня складності, які не вимагають використання важких рушіїв на кшталт Unity або Phaser, проте забезпечують високу якість взаємодії та користувацький досвід.

1.3. Огляд Canvas API як основного рендер-інструменту

Canvas API, що є частиною стандарту HTML5, виступає ефективним інструментом для реалізації двовимірної графіки у вебсередовищі. У контексті розробки браузерних 2D-ігор цей інтерфейс забезпечує можливість безпосереднього рендерингу растрових зображень у межах елемента canvas, що розташовується на вебсторінці та функціонує як віртуальне полотно. На відміну від об’єктів DOM, які є самостійними структурними одиницями з власними властивостями, стилями та подіями, canvas не має внутрішньої об’єктної моделі: всі графічні компоненти сцени створюються та зберігаються виключно в пам’яті програми, а не в структурі документа [9].

Canvas – це HTML-елемент <canvas>, який виступає контейнером для піксельного рендерингу. Він не містить жодних внутрішніх елементів, на відміну від DOM-графіки (наприклад, SVG), а вся графіка малюється скриптами за допомогою JavaScript. Це дає змогу досягати максимальної продуктивності при

виведенні великої кількості візуальних об’єктів, а також повного контролю над вмістом кадру.

Головною перевагою даного підходу є висока продуктивність рендерингу, зумовлена відсутністю необхідності постійної взаємодії з DOM-деревом та пов’язаними з ним витратами на обробку подій і повторне обчислення стилів. У середовищі JavaScript доступ до полотна здійснюється за допомогою виклику методу getContext («2d»), що повертає об’єкт двовимірного контексту. Саме через цей об’єкт виконуються всі операції з графічними примітивами: малювання ліній, прямокутників, кривих, відображення тексту та зображень, трансформації, накладення прозорості, фільтрація пікселів тощо [10].

У рамках реалізації клієнтської частини гри Canvas API використовується для формування основної сцени, яка включає ключові об’єкти: гравця, ворогів, перешкоди, снаряди, вибухи та елементи середовища. Графічне оновлення здійснюється за допомогою циклічної функції, синхронізованої з частотою оновлення екрана через метод requestAnimationFrame(). Завдяки цьому досягається плавність анімації та стабільність виведення зображення. Кожен кадр повністю перемальовується, що дозволяє точно контролювати порядок візуалізації об’єктів, створювати ефекти глибини, затемнення чи багатошаровості.

Окрім сцени, за допомогою Canvas API реалізовано елементи інтерфейсу гри. До них належать: індикатор кількості життів гравця, лічильник залишених ворогів, таймер, повідомлення про завершення рівня або поразку. Інтерфейс побудовано без залучення сторонніх бібліотек, що дало змогу зберегти контроль над продуктивністю, уникнути надмірної залежності від зовнішніх інструментів і забезпечити кращу адаптацію до специфіки проєкту [11].

Незважаючи на численні переваги, Canvas API має і певні функціональні обмеження. Відсутність вбудованої об’єктної моделі ускладнює реалізацію логіки зіткнень, збереження стану об’єктів або взаємодії між ними. Усі ці аспекти повинні бути опрацьовані вручну шляхом створення відповідних структур даних і алгоритмів. Крім того, Canvas не містить вбудованого засобу управління

сценами, шарами чи подіями, що потребує додаткового програмного забезпечення логіки всередині застосунку.

Водночас API характеризується високою сумісністю з основними браузерами, що забезпечує стабільність відображення ігор на широкому спектрі пристроїв. Його інтеграція з іншими вебтехнологіями (зокрема Web Audio API, Gamepad API, Pointer Events, WebSockets) відкриває широкі можливості для розширення функціональності ігрових застосунків [12].

Таким чином, Canvas API є оптимальним рішенням для розробки 2D-графіки в браузері в умовах, коли важливими є контроль над рендерингом, мінімізація ресурсних витрат та гнучкість в організації логіки. У межах реалізованого проєкту Canvas API використано як основний інструмент візуалізації, що забезпечив продуктивну та стабільну роботу гри, незалежну від зовнішніх рушіїв чи надлишкових фреймворків.

РОЗДІЛ 2
АРХІТЕКТУРА КЛІЄНТСЬКОЇ ЧАСТИНИ ГРИ

2.1. Структура файлів проєкту та модульний підхід

У процесі розробки клієнтської частини гри важливим чинником, що безпосередньо впливає на продуктивність розробки, масштабованість системи та зручність подальшого супроводу, є грамотна організація файлової структури та використання модульного підходу. У складних інтерактивних проєктах, зокрема в браузерних іграх, програмний код охоплює значну кількість логічно пов’язаних, але функціонально різнорідних компонентів, зокрема, від обробки користувацького вводу до візуалізації, звукового супроводу та логіки гри.

Чітка і логічна структура файлів дозволяє уникнути дублювання коду, спростити навігацію в проєкті для команди розробників та забезпечити дотримання принципів чистої архітектури. Модульний підхід передбачає розділення системи на незалежні, взаємодіючі частини (модулі), кожна з яких відповідає за окремий аспект функціональності. Такий підхід підтримує принципи Single Responsibility та Separation of Concerns, що є основоположними у сучасному програмному інжинірингу [13].

У межах цього підрозділу буде проаналізовано типову структуру проєкту клієнтської частини гри, реалізованої з використанням HTML5, CSS та JavaScript (або фреймворків на зразок React, Phaser.js чи Babylon.js), розглянуто ключові модулі системи, їхню взаємодію, призначення та обґрунтовано вибір певної архітектурної моделі. Особлива увага буде приділена практичним аспектам: організації папок, розміщенню скриптів, шаблонів, ресурсів та використанню шаблонів проєктування, зокрема MVC (Model-View-Controller) або Component-Based Architecture, що широко використовуються у геймдеві.

Клієнтська частина гри «2D танковий бій» реалізована як вебзастосунок, що складається з набору взаємопов’язаних HTML, CSS і JavaScript файлів. Для забезпечення гнучкості, розширюваності та зручності супроводу застосовано

модульний підхід до побудови структури проєкту, що передбачає розділення коду на логічно відокремлені компоненти відповідно до їхніх функціональних ролей [14].

Фізична структура проєкту має ієрархічну форму з базовим кореневим каталогом, у якому розміщені підкаталоги static/, templates/, а також основний HTML-файл гри (tanki.html). Каталог static/ містить усі ресурси клієнтської частини, у тому числі графіку, стилі та скрипти, й ділиться на декілька підрозділів: CSS/, JavaScript/, Images/.

У підкаталозі CSS/ зберігаються стилі, що відповідають за оформлення різних сторінок гри. Для кожного розділу було створено окремий файл стилів, зокрема: load_to_game_1.css, load_to_game_2.css, game.css, що відповідають за головне меню, екран рівнів та інтерфейс безпосередньо під час гри. Це забезпечує ізоляцію оформлення та полегшує редагування інтерфейсу.

Папка JavaScript/ містить модульні скрипти, поділені за функціональністю. Якщо розглянути окремі файли, то вони будуть відповідати за наступні функції, такі як:

- логіку управління (game.js);
- оновлення HUD та таймерів (hud.js);
- генерацію ворогів та обробку колізій (enemy.js);
- старт і зупинку гри, оновлення сцен (main.js);
- перевірку авторизації, профілю, перемикання модальних вікон тощо (profile.js та інші).

Усі скрипти підключаються до HTML через тег <script> з атрибутом defer, що дозволяє виконувати їх лише після повного завантаження DOM, зберігаючи при цьому порядок виконання.

Папка Images/ зберігає усі графічні ресурси гри – фонові зображення, спрайти танків, логотипи, декоративні елементи, а також анімовані зображення, які використовуються для фону з рухомими танками [15].

У файлі tanki.html реалізовано основну структуру ігрового вікна, включаючи елемент canvas, що виконує роль візуального поля, блоки HUD,

кнопки керування, музичний програвач, а також модальне вікно профілю гравця з можливістю змінювати пароль. Усі елементи оформлено з урахуванням принципів доступності, адаптивності та стилістичної цілісності.

Модульний підхід забезпечує низку переваг, таких як: спрощене тестування окремих компонентів, зручність масштабування проєкту, повторне використання коду, поліпшення читабельності. Це дозволяє ефективно організувати розробку гри, що особливо важливо при розширенні функціоналу або залученні декількох розробників.

2.2. Сценарії завантаження рівнів, профілю та інтерфейсу

Функціонування клієнтської частини гри «2D танковий бій» побудоване на основі сукупності HTML-сторінок і пов’язаних із ними сценаріїв, які реалізують логіку переходів між станами гри, обробку даних користувача та оновлення вмісту інтерфейсу. Усі компоненти клієнтської частини згруповані в окремі модулі, які відповідають за певний тип взаємодії або ігрову логіку. Такий підхід забезпечує модульність, повторне використання коду та полегшує супровід гри [16].

В основі проєкту лежить набір структурованих HTML-документів, кожен з яких відповідає певному функціональному стану гри. Відображення елементів інтерфейсу, робота з подіями користувача, оновлення сцен та вивід ігрової інформації реалізовані засобами JavaScript. Ініціалізація кожної сторінки відбувається через підключення окремого скрипту, що відповідає за її логіку та взаємодію з DOM.

Серед основних сторінок клієнтської частини можна виокремити наступні:

- Головне меню (load_to_game_1.html) – стартова сторінка гри, з якої користувач здійснює вхід у профіль або реєстрацію, має доступ до Discord, а також переходить до вибору рівня. На сторінці реалізовано відображення кнопки входу у профіль, музичного плеєра, кнопки старту та анімованого фону.

- Сторінка вибору рівня (load_to_game_2.html) – містить динамічний набір кнопок, що відповідають доступним рівням. Колір кнопок змінюється залежно від прогресу гравця. Рівні активуються поетапно, на основі даних, збережених у локальному сховищі браузера. Також присутній доступ до профілю та загальні стилістичні елементи оформлення.
- Основне ігрове вікно (tanki.html) – містить основний canvas-елемент, де відбувається рендер ігрової сцени, а також елементи HUD, такі як: лічильники ворогів та життів, таймер, кнопки паузи та контролю музики. Тут реалізовано запуск ігрового циклу та динамічну обробку подій гри.
- Форми автентифікації (index.html, register.html) – реалізують логіку входу та реєстрації користувача, з базовою перевіркою введених даних. Дані користувача зберігаються локально (Login, Email, Password) або передаються на сервер для подальшого зберігання.

Кожна зі сторінок має власну HTML-структуру, однак використовує спільні стилі, скрипти та логіку завантаження, що забезпечує цілісність візуального оформлення. Основні блоки інтерфейсу закладені в HTML, а поведінка цих блоків керується через JavaScript, що забезпечує динамічну адаптацію без повного перезавантаження сторінки.

При переході між сторінками використовуються як внутрішні JavaScript-події (onclick, submit), так і зміна window.location.href для навігації. Дані про користувача – такі як ім’я, кількість пройдених рівнів, рівень доступу – зберігаються у LocalStorage, що дозволяє зберігати їх між сеансами без потреби у серверному сховищі. Наприклад, при вході гравця у головне меню дані про його профіль автоматично підтягуються з локального сховища та відображаються в модальному вікні профілю [17].

Модальне вікно профілю є інтерактивним елементом, реалізованим через прихований HTML-блок, який активується за подією натискання кнопки профілю. Вікно містить такі поля, як логін користувача, кількість набраних очок, місце у таблиці лідерів, а також функцію зміни пароля. Зміна між блоками «профіль» і «зміна пароля» відбувається динамічно, без оновлення сторінки,

через заміну вмісту DOM-елементів за допомогою innerHTML, classList та style.display.

Інтерфейс вибору рівня реалізовано через динамічне оновлення кнопок на основі даних із LocalStorage. Кожна кнопка має унікальний ідентифікатор, і при зміні стану (рівень пройдено/не пройдено) її клас змінюється з level-locked на level-unlocked, що відповідно змінює її колір – з червоного на зелений. Таким чином, гравець візуально сприймає свій прогрес, що мотивує його до подальшого проходження наступних рівнів.

Під час гри (на сторінці tanki.html) ініціалізація рівня відбувається на основі вибраного значення, яке зберігається у LocalStorage перед переходом. Сценарій гри включає завантаження мапи рівня, генерацію ворогів, запуск основного ігрового циклу, відображення HUD та інтеграцію з музичним плеєром. Ігровий прогрес зберігається автоматично після завершення рівня або поразки гравця, з оновленням статусу у локальному сховищі.

Усі сторінки проєкту витримані в єдиній стилістиці, зокрема, неонові кольори, піксельні шрифти, рухомий фон, однакова структура розміщення кнопок і модальних елементів. Такий підхід сприяє зниженню когнітивного навантаження на користувача, забезпечує швидку орієнтацію в інтерфейсі та формує візуально завершений образ гри.

Для забезпечення масштабованості проєкту передбачено поділ логіки кожної сторінки на окремі скрипти (наприклад, home.js, register.js, leaderboard.js), що дозволяє незалежно змінювати поведінку певного компонента без втручання в інші частини системи. Це відповідає принципам компонентної архітектури та забезпечує гнучкість у подальшому розвитку гри.

Таким чином, реалізовані сценарії завантаження інтерфейсу, рівнів і профілю базуються на модульному підході, динамічній зміні вмісту DOM та використанні локального сховища даних. Завдяки цьому досягається безперервність ігрової взаємодії, узгодженість інтерфейсу та можливість персоналізації г ри відповідно до стану користувача.

2.3. Управління ресурсами, подіями та взаємодія з DOM

У клієнтській частині гри «2D танковий бій» ключовим завданням є забезпечення ефективного керування ресурсами – графічними, звуковими та інформаційними, а також обробка подій, які генеруються в результаті взаємодії користувача з ігровим інтерфейсом. Реалізація вказаної функціональності здійснена засобами мови JavaScript, із застосуванням подієво-орієнтованої архітектурної моделі, що відповідає вимогам сучасної веброзробки [18].

На етапі ініціалізації ігрової сцени завантажуються усі необхідні графічні ресурси, включаючи зображення танків, фону, елементів інтерфейсу, вибухів, об'єктів оточення тощо. Для запобігання виникненню артефактів або збоїв при рендерингу, реалізовано механізм попереднього кешування зображень через об'єкти типу Image. Завантаження виконується асинхронно, із перевіркою готовності кожного ресурсу до моменту запуску основного ігрового циклу. Аналогічно, для відтворення звукових ефектів та фонової музики застосовується Audio API, який забезпечує контроль гучності, відтворення, паузи, перемикання композицій та інтеграцію з елементами керування, розміщеними в HUD.

Обробка подій відбувається за допомогою системи слухачів подій (event listeners), що прив’язуються до глобального об’єкта window, або до конкретних DOM-елементів. Використовуються події клавіатури (keydown, keyup), миші (click, mouseover) та сенсорні (touchstart) для підтримки кросплатформенності. Наприклад, при натисканні стрілок реалізується рух танка у відповідному напрямку, а натискання клавіші Space активує постріл. Для уникнення багаторазового спрацювання події при утриманні клавіші використовується система логічних прапорів, які зберігають поточний стан клавіатури, дозволяючи виконувати одну дію лише один раз на зміну стану [19].

Важливою складовою є взаємодія з DOM – об’єктною моделлю документа, яка дозволяє динамічно змінювати вміст сторінки без її перезавантаження. Застосовуючи методи querySelector, getElementById, classList.add/remove, innerText, реалізовано механізми керування модальними вікнами (наприклад,

профілем гравця), оновлення кількості життів, таймера, кількості ворогів тощо. Зміни в DOM здійснюються у відповідь на ігрові події. Наприклад, при втраті життів відповідний індикатор на екрані змінює вигляд (колір, іконку), а при натисканні кнопки паузи оновлюється її текстове наповнення [20].

Особливу увагу приділено обробці нестандартних сценаріїв. Зокрема, передбачено поведінку гри при втраті фокусу вікна браузера (visibilitychange), натисканні кнопки «Назад» у браузері, перезапуску сторінки без втрати прогресу (beforeunload). У таких випадках застосовуються додаткові перевірки стану гри, збереження проміжних результатів у localStorage та оновлення внутрішніх змінних для коректного відновлення ігрового процесу.

Реалізація вказаних функцій є тісно пов’язаною з модульною структурою клієнтського коду. Кожен модуль відповідає за обробку конкретної категорії подій і взаємодіє лише з тими елементами DOM, які входять у його зону відповідальності. Такий підхід підвищує стабільність, прогнозованість ігрової логіки, дозволяє ефективно проводити налагодження й ізолювати джерела можливих помилок. Наприклад, модуль workinghud.js відповідає лише за оновлення даних у HUD, movement.js – за обробку клавіш переміщення, а main.js координує загальний цикл гри та відстеження стану.

Таким чином, реалізована система управління ресурсами, подіями та DOM забезпечує високу реактивність, візуальну узгодженість та функціональну гнучкість клієнтської частини гри. Завдяки цьому гра коректно реагує на дії користувача, підтримує стабільний ігровий ритм і дає змогу реалізувати різнорівневі ігрові сценарії без втрати продуктивності чи цілісності інтерфейсу.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ІГРОВИХ МЕХАНІК ТА ОПТИМІЗАЦІЯ

3.1. Реалізація руху, стрільби, колізій та HUD

Ключовими компонентами ігрового процесу у 2D грі «Танковий бій» є управління рухом гравця, реалізація стрільби, обробка зіткнень (колізій), а також динамічне відображення ігрової інформації за допомогою інтерфейсу HUD. Всі ці елементи реалізовані на клієнтській частині за допомогою JavaScript, із використанням Canvas API для графічного рендерингу.

Рух гравця реалізовано через події keydown та keyup, які встановлюють або скидають логічні прапорці (moveUp, moveDown, moveLeft, moveRight). У кожному кадрі ігрового циклу, який оновлюється через requestAnimationFrame(), координати танка змінюються відповідно до активних напрямків руху. Це дозволяє досягти плавного і стабільного переміщення, яке не залежить від частоти натискань клавіш. Фрагмент коду реалізації руху танка зображено в Лістинг 3.1.

Лістинг 3.1 – Реалізація руху танка

```
document.addEventListener("keydown", function (e)) {
    switch (e.key) {
        case "ArrowUp":
            player.moveUP = true;
            break;
        case "ArrowDown":
            player.moveDown = true;
            break;
        case "ArrowLeft":
            player.moveLeft = true;
            break;
        case "ArrowRight":
            player.moveRight = true;
            break;
        case " ":
            player.shoot();
            break;
    }
};
```

Стрільба реалізована як створення снарядів – об’єктів, які додаються до масиву bullets при натисканні клавіші Space. Кожен снаряд має напрямок, швидкість та координати. Під час кожного оновлення сцени всі активні снаряди переміщуються та перевіряються на зіткнення з іншими об’єктами. Якщо снаряд влучає в ціль (ворога або стіну), він знищується, а ворожий танк або перешкода – відповідно отримують пошкодження або зникають. Процес стрільби по ворогу зображено на рис. 3.2.

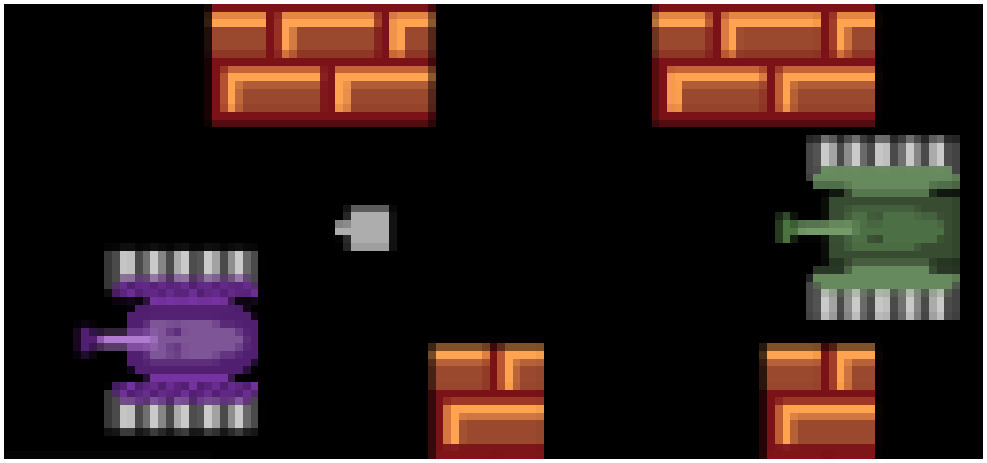

Рисунок 3.2 – Процес стрільби по ворогу

Обробка колізій виконується за допомогою методу прямокутного перетину (Axis-Aligned Bounding Box, AABB). Для кожного об'єкта на полі обчислюються його межі, після чого здійснюється перевірка на перетин із іншими активними об'єктами, такими як: танки, снаряди, блоки стін. У разі виявлення колізії виконується відповідна реакція – блокування руху, знищення об'єкта, зменшення кількості життів. Виведення лічильника життів, ворогів та таймера показано на рис. 3.3.

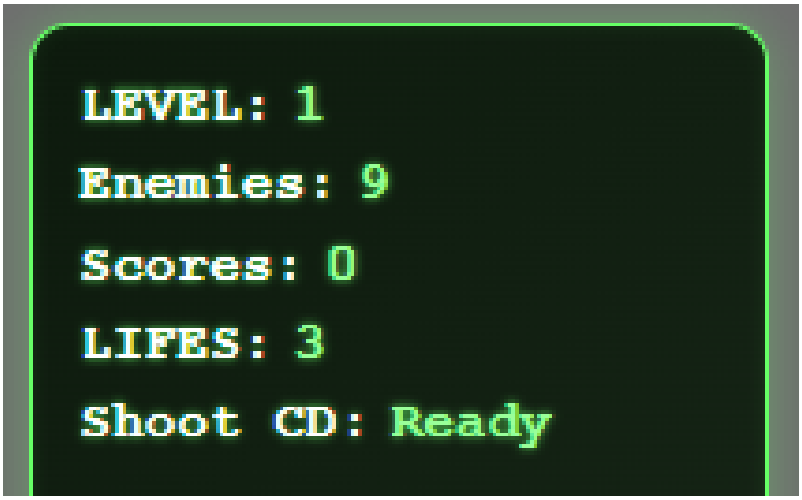

Рисунок 3.3 – Виведення лічильника життів, ворогів та таймера (HUD)

HUD (Head-Up Display) – це графічний інтерфейс, який показує гравцю поточну кількість життів, ворогів, таймер і кнопки керування (наприклад, «Пауза», «Музика»). Елементи HUD оновлюються в режимі реального часу шляхом взаємодії з DOM – за допомогою document.querySelector() та innerText змінюються значення відповідних полів. Наприклад, при знищенні ворога лічильник #enemyCount автоматично зменшується. У разі втрати життя відповідний елемент графічно оновлюється (зміна іконки, кольору або зникнення).

Для покращення інтерфейсної взаємодії кнопки HUD мають інтерактивність, адже, кнопка «Пауза» змінює свій напис на «Продовжити» залежно від стану гри, а кнопка звуку вмикає/вимикає фонову музику, що реалізується через DOM-події та класи CSS. Таймер реалізовано через setInterval(), що щосекундно зменшує значення й оновлює відповідне поле.

Завдяки модульному підходу до побудови гри кожна з цих функцій розташована в окремому логічному блоці коду: movement.js, bullet.js, collision.js, hud.js, що дозволяє підтримувати читабельність і спрощує налагодження.

Таким чином, у цьому підрозділі реалізовано основні елементи ігрової логіки, що забезпечують динамічність, інтерактивність та зворотний зв'язок для користувача. Ці механіки є центральними для формування базового геймплею і

створюють основу ігрового досвіду. Фрагмент коду, що реалізує рух гравця та стрільбу, подано в додатку А.

3.2. Анімація фону, музичний плеєр, пауза та таймер

У сучасному веброзробленні дедалі більше уваги приділяється не лише функціональності, а й візуальній та емоційній складовій користувацького досвіду. Анімація, звукові ефекти та інтерактивні елементи перетворюють звичайні сайти на динамічні, живі простори взаємодії. Одним із яскравих прикладів такої інтеграції є поєднання анімованого фону, вбудованого музичного плеєра, функції паузи та таймера відтворення, які разом створюють гармонійну мультимедійну композицію.

Анімація фону – це не просто естетичне рішення, а спосіб впливу на настрій користувача, його залучення до контенту та зниження візуальної втоми. Завдяки сучасним можливостям CSS3 та апаратному прискоренню браузерів, фонові ефекти можуть бути плавними, енергоефективними та оптимізованими для різних пристроїв.

Інтеграція аудіоплеєра є не менш важливим аспектом взаємодії, особливо для сайтів, пов’язаних з медіа, мистецтвом або освітніми ресурсами. Сучасні браузери підтримують відтворення звуку через тег <audio>, що дозволяє реалізувати повноцінний плеєр із керуванням – включаючи запуск, паузу та обробку подій часу.

Функціонал таймера дає змогу користувачеві відстежувати тривалість аудіовідтворення, а також інтегрувати цей час у ширший контекст взаємодії — наприклад, у форматі "відлік до події" або "залишилось часу". Це не лише покращує доступність інтерфейсу, а й відповідає очікуванням сучасного UX-дизайну.

У цьому розділі буде детально розглянуто технічні та дизайнерські аспекти реалізації зазначених компонентів за допомогою HTML5, CSS3 та JavaScript. Розглядатимуться основні принципи побудови анімованих фонових сцен,

реалізація аудіоплеєра з підтримкою паузи та обчисленням часу, а також особливості їх інтеграції у єдиний узгоджений інтерфейс. Метою є не лише створення функціонального елементу, але й демонстрація того, як технології здатні створити привабливий, інтуїтивно зрозумілий і візуально захопливий досвід для користувача.

Для підвищення візуальної привабливості гри «2D танковий бій» реалізовано окремі елементи візуального і звукового супроводу, які доповнюють основні механіки гри. Зокрема, до таких елементів належать анімований фон, музичний плеєр, механізм зупинки гри (пауза) та внутрішній таймер, що фіксує тривалість гри або рівня.

Анімація фону реалізована у вигляді рухомих танків або об'єктів, які не беруть участі в геймплеї, але постійно переміщуються по екрану, створюючи ефект «живого» середовища. Візуалізація цих об’єктів здійснюється на окремому шарі в canvas або безпосередньо через DOM із застосуванням CSS-анімацій (@keyframes) та властивостей transform, translateX, opacity. Кожен фонтанк має власну траєкторію руху, яка може бути лінійною або криволінійною. Завдяки цьому створюється глибина сцени без навантаження на основний рендер-цикл гри. Анімація фону та музичний плеєр зображено на рис.3.4.

Рисунок 3.4 – Анімація фону та музичний плеєр

Музичний плеєр вбудовано у нижню частину інтерфейсу. Його реалізація базується на вбудованому HTML-елементі <audio>, керування яким

здійснюється через JavaScript. Гравець має можливість вмикати або вимикати музику натисканням на відповідну кнопку HUD. Зміна стану кнопки реалізована через класову логіку CSS та перемикання іконок (наприклад, «звук увімкнено» ↔ «звук вимкнено»). Плеєр підтримує відтворення у циклі та автоматично запускається після початку гри. Зміна анімації фону представлена на рис. 3.5.

Рисунок 3.5 – Зміна анімації фону

Механізм паузи дозволяє гравцю тимчасово зупинити гру, що особливо актуально у динамічних сценах. При натисканні кнопки «Pause» викликається функція, яка змінює стан внутрішньої змінної isPaused та припиняє оновлення canvas через припинення викликів requestAnimationFrame. У цьому режимі всі рухомі об’єкти «заморожуються», а HUD змінює свій вигляд (наприклад, кнопка «Pause» змінює текст на «Resume»). У разі повторного натискання – оновлення ігрової сцени відновлюється. Вигляд паузи у грі зображено на рис. 3.6.

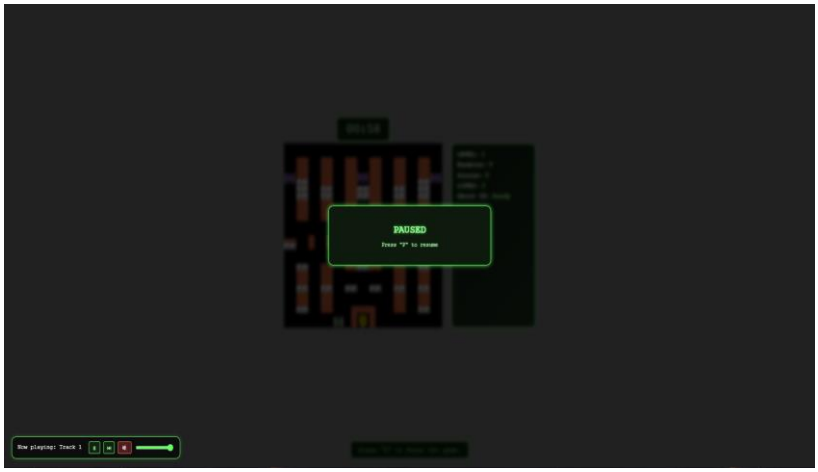

Рисунок 3.6 – Вигляд паузи у грі

Таймер гри реалізовано через метод setInterval, який з інтервалом у 1000 мс збільшує лічильник часу та відображає його у відповідному елементі HUD. Значення таймера оновлюється динамічно, і може використовуватись як для зворотного відліку часу до завершення рівня, так і для фіксації часу проходження. У разі зупинки гри (пауза) оновлення таймера також призупиняється, що синхронізується зі станом isPaused. Вигляд таймера зображено на рис. 3.7.

Рисунок 3.7 – Таймер гри

Кожен із перелічених елементів спрямований на покращення користувацького досвіду. Анімація та звук підсилюють атмосферу гри, пауза забезпечує контроль над процесом, а таймер – формує чітке уявлення про хід проходження рівня. HTML-структура canvas-елементів та відповідні CSS-стилі наведені в додатках Б.

3.3. Оптимізація продуктивності гри та налагодження

Забезпечення стабільної та швидкої роботи клієнтської частини є ключовою умовою для створення якісного ігрового застосунку. У даному підрозділі розглядаються заходи, спрямовані на оптимізацію рендерингу, зменшення навантаження на браузер та спрощення процесу виявлення й усунення помилок.

Одним із найважливіших аспектів оптимізації стала робота з Canvas API, який постійно оновлює ігрову сцену в межах циклу requestAnimationFrame. Для зменшення навантаження було реалізовано механізм кешування зображень, тобто усі спрайти попередньо завантажуються в пам’ять, а не створюються під час кожного кадру. Це дозволяє уникнути затримок при відтворенні нових об’єктів і зменшує кількість звернень до файлової системи.

Крім того, графіка, яка не змінюється між кадрами (наприклад, статичні елементи фону), малюється один раз і не оновлюється, що суттєво скорочує кількість операцій у кожному циклі. Анімація танків, снарядів і вибухів реалізована через зміну координат об’єктів, без необхідності перемальовування всієї сцени.

Було зведено до мінімуму використання глобальних змінних, що дозволило уникнути конфліктів у логіці гри. Локальні змінні та ізольовані функції у відповідних модулях (movement.js, bullet.js, hud.js) підвищили керованість коду.

Для зменшення DOM-навантаження всі зміни у HUD виконуються лише у разі фактичної зміни значення. Наприклад, лічильник ворогів оновлюється лише після знищення противника, що запобігає зайвим перерендеренням елементів інтерфейсу.

Тестування проводилося вручну в основних браузерах (Google Chrome, та Edge) на різних розширеннях екрана. Для перевірки коректності логіки руху, колізій, стрільби та оновлення HUD використовувалися інструменти браузера –

консоль, логування подій, інспектор елементів. Для локалізації складних помилок використовувався console.trace() та інші діагностичні функції.

Особливу увагу приділено запобіганню критичним помилкам, які можуть спричинити зависання гри або її зупинку. Було реалізовано перевірку коректності вхідних даних, обробку некоректних дій користувача (наприклад, багаторазове натискання клавіш або швидкий перезапуск гри), а також логіку зупинки гри при втраті фокусу сторінки (visibilitychange). Тестування гри в Google Chrome зображено на рис. 3.8.

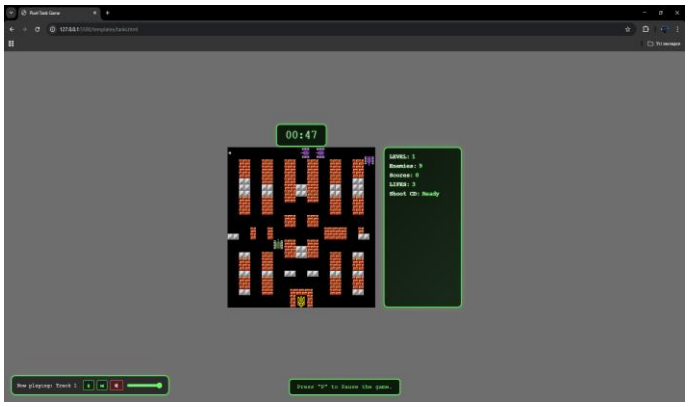

Рисунок 3.8 – Тестування в Google Chrome

Проведені заходи оптимізації дозволили забезпечити плавну роботу гри при 60 FPS на більшості сучасних пристроїв, включаючи ноутбуки середнього рівня та сучасні мобільні телефони. Це підтверджує ефективність застосованого підходу до розробки та якість реалізації ігрового процесу. Тестування в Microsoft Ege зображено на рис. 3.9.

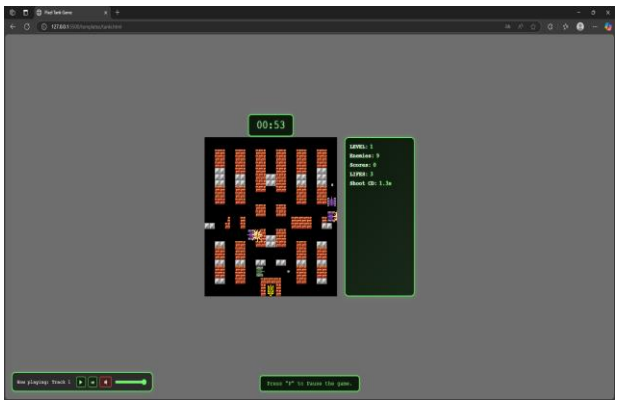

Рисунок 3.9 – Тестування в Microsoft Ege

3.4 Тестування гри

Після завершення реалізації функціоналу гри було проведено тестування для перевірки її працездатності, стабільності та сумісності з різними браузерами. Метою тестування було виявлення можливих помилок, збоїв у логіці гри, візуальних дефектів інтерфейсу та некоректної обробки подій.

Основним методом було функціональне ручне тестування, яке охоплює перевірку усіх ключових дій користувача: запуск гри, рух гравця, стрільбу, знищення ворогів, взаємодію з HUD, паузу, зміну налаштувань та перемикання між сторінками. Тестування проводилося в таких браузерах: Google Chrome (v124), Mozilla Firefox (v125), Microsoft Edge (v123), що дозволило оцінити кросбраузерну сумісність.

Додатково перевірялись нестандартні сценарії, такі як повторне натискання клавіш, швидка перезагрузка сторінки, втрата фокусу браузера, некоректні введення в полях авторизації. Тестовий план гри включав перевірку функцій, представлених у табл. 3.1.

Таблиця 3.1 – Перевірка функцій гри при її тестуванні

№	Тестована функція	Вхідні дані	Очікуваний результат	Результат
1	Рух танка	Натискання клавіш «←», «↑», «→», «↓».	Танчик переміщується у відповідному напрямку	Успішно
2	Стрільба	Натискання Space	Випущено снаряд, влучає у ворога	Успішно
3	Зіткнення снаряда з ворогом	Прямий постріл	Ворог знищується, зменшується лічильник ворогів	Успішно
4	Зменшення кількості життів	Дотик ворожого снаряду до гравця	Зменшується кількість життів, відображення в HUD	Успішно
5	Пауза гри	Натискання кнопки «P»	Гра зупиняється, анімація зупиняється	Успішно
6	Відновлення гри	Натискання кнопки «P»	Продовження гри з поточного стану	Успішно
7	Відображення профілю	Клік по іконці профілю	Відкривається модальне вікно з даними	Успішно
8	Зміна пароля	Введення нового пароля	Пароль змінено, підтвердження у вікні	Успішно
9	Перемикання рівнів	Натискання кнопки рівня	Завантажується відповідна сторінка	Успішно
10	Відтворення/ вимкнення музики	Кнопка аудіо в HUD	Музика вмикається або вимикається	Успішно

У результаті тестування суттєвих помилок не виявлено. Усі перевірені компоненти функціонували згідно із запланованою логікою. Невеликі візуальні неточності були усунуті на етапі налагодження.

Таким чином, проведене тестування підтвердило коректну роботу клієнтської частини гри у більшості актуальних середовищ. Реалізовані механізми забезпечують стабільний геймплей, чуйний інтерфейс та належний рівень взаємодії користувача з ігровим простором. Узагальнені результати подано у вигляді тестового плану та візуально в додатку Б.

ВИСНОВКИ

Робота носить практичний характер і включає повноцінну реалізацію клієнтської частини браузерної гри, що демонструє набуті знання з програмування, веброзробки та проєктування інтерфейсів у межах обраної спеціальності. У процесі виконання кваліфікаційної роботи було створено працюючий приклад вебзастосунку, що реалізує класичну 2D-гру жанру «танковий бій», адаптовану для запуску безпосередньо в браузері.

У теоретичній частині проведено аналіз типів 2D-ігор, визначено їхні функціональні особливості, а також обґрунтовано вибір технологій для реалізації клієнтської частини: HTML5, CSS3, JavaScript та Canvas API. Надано характеристику кожному інструменту, описано принципи їх взаємодії в контексті рендерингу, анімації, стилізації та побудови ігрової логіки. Це дозволило сформувати технічно обґрунтовану модель побудови гри.

У практичній частині реалізовано архітектуру клієнтської частини на основі модульного підходу, що забезпечує зручність підтримки та масштабування проєкту. Розроблено логіку руху гравця, стрільби, взаємодії з ворогами, обробки зіткнень, запуску анімацій та оновлення HUD у реальному часі. Додатково реалізовано фонову анімацію, вбудований музичний супровід, систему паузи та таймер. Структура HTML-документів, таблиці стилів і логіка JavaScript коду розмежовані за функціональними частинами, що відповідає сучасним принципам фронтенд-розробки.

Особливу увагу приділено питанням продуктивності: зменшено кількість звернень до DOM, використано кешування ресурсів, Canvas API оптимізовано для плавного оновлення графіки. У результаті було досягнуто стабільної частоти оновлення гри (60 FPS) на більшості сучасних браузерів.

Проведене тестування в середовищах Google Chrome, Mozilla Firefox та Microsoft Edge підтвердило коректну роботу гри на різних екранах. Усі критично важливі функції – від обробки клавіш до логіки гри – працюють згідно із заданими сценаріями. Результати тестів задокументовано у вигляді таблиці

тестового покриття, а ключові елементи інтерфейсу продемонстровано у вигляді скріншотів.

Таким чином, поставлені у вступі мета та завдання були повністю досягнуті. Розроблений програмний продукт може бути використаний як основа для подальшого розвитку: інтеграції серверної частини (FastAPI), реалізації мультиплеєрного режиму, додавання системи збереження прогресу, нових рівнів, досягнень та рейтингових турнірів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Taylor, A. Mastering 2D RPG Adventure Game Development in Java: A Comprehensive Guide to Designing, Coding, and Launching Your Own 2D Role-Playing Game. 1-е вид. Kindle eBook. 2024. 82 с. URL: https://www.amazon.com/Mastering-2D-RPG-Adventure-Game-Development-Java-ebook/dp/B0DF32YHH4 (дата звернення: 05.04.2025).
2. Jung, D. Android 2D Game Programming with Java: Easy for Teens by Using Game Library. Birmingham: Independently published, 2023. 255 с. URL: https://www.amazon.de/-/en/DONGGEUN-JUNG/dp/B0BVC8MXKB amazon.de (дата звернення: 05.04.2025).
3. ZetCode. Java 2D Games Programming e-Book. 1-е вид. 2022. 115 с. URL: https://zetcode.com/ebooks/javagames/ reddit.com+15zetcode.com+15link.springer.com+15 (дата звернення: 07.04.2025).
4. Nair, S. B. Learning LibGDX Game Development – Second Edition. Packt Publishing, 2021. (≈300 с.). URL: https://subscription.packtpub.com/book/game-development/9781783554775/pref reddit.com+5 (дата звернення: 12.04.2025).
5. Baimagambetov, Almas. Learn JavaFX Game and App Development: With FXGL 17. New Canaan: Elm Street Books (Apress), 2022. 224 с. URL: https://link.springer.com/book/10.1007/978-1-4842-8625-8 link.springer.com+10oreilly.com+10foojay.io+10 (дата звернення: 12.04.2025).
6. Jung, D. Android 2D Game Programming with Java: Easy for Teens by Using Game Library. Independently published, 2023. 255 с. URL: https://www.amazon.de/-/en/DONGGEUN-JUNG/dp/B0BVC8MXKB (дата звернення: 12.04.2025).
7. HTML Living Standard // WHATWG. – URL: https://html.spec.whatwg.org/ (дата звернення: 17.04.2025).
8. CSS Snapshot 2021 // W3C. – URL: https://www.w3.org/TR/css-2021/ (дата звернення: 25.04.2025).

9. JavaScript Documentation // MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 27.04.2025)
10. Canvas API Documentation // MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (дата звернення: 30.04.2025)
11. ECMAScript 2021 Language Specification // Ecma International. – URL: https://ecma-international.org/publications-and-standards/standards/ecma-262/
12. JavaScript and jQuery: Interactive Front-End Web Development / Jon Duckett. – Wiley, 2021, 640 p (дата звернення: 01.05.2025).
13. HTML5 in Action / Rob Crowther, Joe Lennon, Ash Blue, Greg Wanish. – Manning Publications, 2021. 375 p (дата звернення: 15.05.2025).
14. GitHub: Canvas-based Tank Game Examples – URL: https://github.com/topics/tank-game (дата звернення: 01.06.2025).
15. 2D breakout game using pure JavaScript // MDN Web Docs. – URL: https://developer.mozilla.org/en-US/docs/Games/Tutorials/2D_Breakout_game_pure_JavaScript (дата звернення: 01.06.2025).
16. Mastering 2D Game Development with JavaScript, HTML, and CSS // Bomberbot. – URL: https://www.bomberbot.com/javascript/mastering-2d-game-development-with-javascript-html-and-css-a-comprehensive-guide/ (дата звернення: 01.06.2025).
17. Introduction to JavaScript Games Development // Simplilearn. – URL: https://www.simplilearn.com/tutorials/javascript-tutorial/javascript-games
18. Game Development Tutorials // Spicy Yoghurt. – URL: https://spicyyoghurt.com/game-development-tutorials (дата звернення: 01.06.2025).
19. Game Development – Building My First 2D TileMap In A Video Game // FAUN Developer Community. – URL: https://faun.pub/game-development-building-my-first-2d-tilemap-in-a-video-game-c8d90ad1af59 (дата звернення: 03.06.2025).

20. Best JavaScript and HTML5 game engines // LogRocket Blog. – URL: https://blog.logrocket.com/best-javascript-html5-game-engines-2025/ (дата звернення: 03.06.2025).