Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

На тему: **«Розроблення програмного забезпечення для навчання програмування засобами Phaser 3»**
(тема роботи)

Виконав: студент 4 курсу групи ІСТ-41
спеціальності:
126 „Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)
Озарко М.А.
(прізвище, ініціали)

Керівник: Часковський О.Г.
(прізвище, ініціали)

Рецензент: Процик Ю.С.
(прізвище, ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший(бакалаврський)
Спеціальність 126 " Інформаційні системи та технології "

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
____________ Сторожук О.Л.
„ 15 " 11 2024р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Озарко Марії Андріївні
(прізвище, ім'я, по батькові)

1. Тема роботи: Розроблення програмного забезпечення для навчання програмування засобами Phaser 3 / Development of software for learning programming using Phaser 3 tools

керівник роботи *Часковський О.Г, к.с-г.н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "15" листопада 2024 року № С-884

2. Термін подання студентом роботи 12 червня 2025 р.

3. Вихідні дані до роботи Створити інтерактивну гру для засвоєння основ програмування із застосуванням ігрового фреймворку Phaser 3. Організувати серверну частину за допомогою мови PHP із використанням технології Ajax для обміну даними. Розробити два ігрові рівні, які б ілюстрували функціональні можливості фреймворку Phaser 3. Реалізувати інтерфейс, що поєднує простоту у використанні з ефективністю в навчанні базових конструкцій циклів у програмуванні.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. Стан проблемної області
РОЗДІЛ 2. Інформаційне та математичне забезпечення
РОЗДІЛ 3. Програмне та технічне забезпечення
ВИСНОВКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді .

6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз наукових джерел відповідно до обраної тематики та добір релевантних матеріалів.	18.11.2024-26.11.2024	Виконано
2.	Формулювання основного завдання дослідження та його математична/логічна формалізація.	26.12.2024-14.01.2024	Виконано
3.	Проєктування базової функціональної архітектури майбутнього програмного продукту.	14.01.2024-24.12.2024	Виконано
4.	Розроблення користувацького інтерфейсу та графічних компонентів гри.	24.12.2024-20.02.2025	Виконано
5.	Імплементація інтерактивної логіки гри з оновленням коду в режимі реального часу.	20.02.2025-30.03.2025	Виконано
7.	Завершення візуального оформлення та складання технічної документації до програмного забезпечення.	30.03.2025-20.04.2025	Виконано
8.	Передача пояснювальної записки на перевірку рецензенту.	20.04.2025-15.05.2025	Виконано
9.	Підготовка презентаційних матеріалів та публічного захисту.	15.05.2025-20.05.2025	Виконано

Студент ____________ (підпис) **Озарко М.А.** (прізвище та ініціали)

Керівник роботи ____________ (підпис) **Часковський О.Г.** (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 40 сторінок пояснювальної записки, 25 рисунків, 15 джерел, 1 додатку.

У дипломному проєкті розглядається створення інтерактивної освітньої гри, що слугує інструментом для ознайомлення з основами програмування. У якості рушія використано фреймворк Phaser 3, який забезпечує динамічну взаємодію користувача з елементами гри. Реалізовано механізм миттєвого оновлення інтерфейсу відповідно до внесених змін за допомогою Ajax-запитів (POST), які обробляються сервером, створеним на основі PHP. Такий підхід дозволяє користувачу у режимі реального часу будувати логіку проходження гри, що підсилює засвоєння навчального матеріалу.

Ключові слова: інтерактивне навчання, Phaser 3, PHP, Ajax, ігрова методика.

ABSTRACT

The thesis contains 40 pagesof explanatory note, 25 figures, 15 sources, 1 appendix.

This thesis explores the development of an educational interactive game designed to facilitate the acquisition of basic programming skills. The project employs the Phaser 3 framework to deliver real-time user engagement through dynamic gameplay. The backend, implemented with PHP, processes POST requests via Ajax, enabling immediate reflection of code changes within the interface. This architecture empowers users to construct their own progression paths through the game, enhancing the educational impact.

Keywords: educational game, Phaser 3, PHP, Ajax, real-time interaction.

ТЕХНІЧНЕ ЗАВДАННЯ

Здійснити розробку інтерактивного навчального застосунку у вигляді гри, яка сприятиме ефективному засвоєнню базових принципів програмування. Проєкт має бути реалізований з використанням сучасного ігрового фреймворку Phaser 3. Передбачити функціональність, що забезпечує візуалізацію змін в інтерфейсі застосунку в режимі реального часу у відповідь на введення коду користувачем. Побудувати логіку, яка дозволить формувати власну траєкторію проходження рівнів, адаптовану до дій гравця. Серверну частину реалізувати засобами мови PHP, забезпечивши взаємодію з клієнтською частиною через Ajax-технологію шляхом обробки POST-запитів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ 9
1.1 Огляд проблемної області 9
1.2 Огляд найпопулярніших JavaScript-фреймворків із відкритим кодом 11
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 18
2.1 Phaser 18
2.2 Побудова дерева проблем та дерева цілей 21
РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ 24
3.1 Послідовний опис розробки інтерфейсу застосунку 24
3.2 Тестування проекту 34
ВИСНОВКИ 40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 41
ДОДАТКИ 45

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API - Application Programming Interface (Програмний інтерфейс взаємодії компонентів)

REST - Representational State Transfer (Архітектурний стиль веб-сервісів)

JSON - JavaScript Object Notation (Текстовий формат обміну даними)

JWT - JSON Web Token (Токен аутентифікації (HS256))

ORM - Object-Relational Mapping (Технологія зіставлення об’єктів з таблицями БД)

DB - Database (База даних)

DBMS - DataBase Management System (Система керування базами даних)

CRUD - Create, Read, Update, Delete (Базові операції над даними)

UTC - Coordinated Universal Time (Всесвітній координований час)

CI/CD - Continuous Integration / Continuous Delivery (Безперервна інтеграція та доставка)

ER - Entity–Relationship (Модель зв’язків «сутність-зв’язок»)

FSM - Finite-State Machine (Кінцевий автомат (Aiogram-бот))

Celery - Distributed Task Queue (Розподілена черга задач у Python)

Redis - Remote Dictionary Server (In-memory сховище ключ-значення)

HTTP - HyperText Transfer Protocol (Протокол передачі гіпертексту)

HTTPS - HTTP Secure (SSL/TLS) (Захищений варіант HTTP)

SSL/TLS - Secure Sockets Layer / Transport Layer Security (Криптографічні протоколи шифрування трафіку)

LTS - Long-Term Support (Версія програмного забезпечення з тривалою підтримкою)

RPO - Recovery Point Objective (Максимальна втрата даних при відновленні (ціль))

RTO - Recovery Time Objective (Максимальний час відновлення системи (ціль))

ВСТУП

Сучасний ринок HTML5-ігор демонструє стрімке зростання та високий попит, зокрема в контексті інтеграції в соціальні платформи. Провідні сервіси, такі як Facebook, активно впроваджують власні платформи для запуску HTML5-ігор, проводять конкурси для залучення розробників та стимулюють створення нових продуктів. Платформа Instant Games від Facebook і підтримка HTML5-ігор у Telegram свідчать про тенденцію до глобалізації такого типу контенту. Паралельно з ігровою індустрією розвивається напрям інтерактивних освітніх ігор, які відкривають нові можливості в процесі пізнання, полегшуючи засвоєння складних тем через гейміфіковану взаємодію.

У цьому контексті розробка навчальних рішень на базі HTML5-технологій, зокрема з використанням фреймворку Phaser 3, є актуальним і перспективним напрямом дослідження.

Об'єкт дослідження — фреймворк Phaser 3 як інструмент створення інтерактивного ігрового середовища.

Мета роботи — створення інтерактивної гри, що сприятиме зручному й ефективному навчанню базових понять програмування.

Предмет дослідження — технології реалізації взаємодії між клієнтською та серверною частинами застосунку з використанням Ajax-запитів.

Практична цінність — простий у користуванні, безкоштовний та вільний від рекламного навантаження інтерфейс, що дозволяє застосовувати ігрову систему у навчальних цілях.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Огляд проблемної області

У сучасному світі цифрових технологій інформаційно-комунікаційні засоби набувають усе більшого значення в освітньому процесі. Вони не лише розширюють можливості доступу до знань, а й кардинально змінюють способи їх подачі, роблячи навчання більш гнучким, особистісно орієнтованим та інтерактивним. Одним із найбільш перспективних напрямів такого розвитку є використання ігрових та симуляційних середовищ як засобу пізнання.

Інтерактивні ігри, що поєднують ігрові механіки з освітнім контентом, не лише стимулюють мотивацію до навчання, а й створюють умови для глибшого засвоєння матеріалу. Завдяки гейміфікації освітній процес стає емоційно насиченим, більш природним для сприйняття учнями, особливо в умовах дефіциту навчального часу та обмеженого доступу до високоякісних методичних ресурсів.

Сучасна педагогіка ставить перед вчителем завдання створення навчального середовища, у якому учень активно бере участь у побудові знань, взаємодіє з інформаційним простором та розвиває навички, необхідні для життя в цифрову епоху. Одним із потужних інструментів реалізації таких завдань є ігрові інтерактивні методи, що дозволяють об’єднати когнітивні, емоційні та соціальні аспекти навчання.

Значення ігрової діяльності у навчальному процесі підкреслюється не лише педагогами-практиками, а й численними науковими дослідженнями. Ігри сприяють формуванню внутрішньої мотивації, розвитку уваги, пам’яті, креативного мислення, а також створенню довірливої атмосфери в колективі. Особливу роль відіграють короткі ігрові епізоди — міні-змагання, вікторини, рольові ситуації — які потребують мінімум підготовки, але мають значний виховний та розвивальний ефект.

Інтерактивне навчання ґрунтується на активній взаємодії між учасниками освітнього процесу. Сам термін "інтерактив" передбачає взаємне впливання та діалог, що є особливо актуальним у роботі з молоддю. Формуючи позитивний досвід співпраці та самовираження, такі методи підвищують ефективність засвоєння матеріалу, формують комунікативні та соціальні компетентності.

Як влучно говорить африканське прислів'я:

«Один палець не миє обличчя»

воно нагадує, що реальні зміни та глибоке розуміння виникають лише через взаємодію й співпрацю.

Інтерактивна гра стає в такому контексті не лише дидактичним засобом, а потужним соціокультурним інструментом, що формує цінності, зміцнює віру у власні сили та відкриває нові горизонти пізнання.

Сьогоднішній етап розвитку освіти характеризується активним впровадженням комп'ютерних технологій на всіх рівнях навчального процесу. Залежно від вікових особливостей дитини та засобів ІКТ, комп'ютер виконує різні ролі — від віртуального наставника або екзаменатора до засобу організації ігрової діяльності в дошкільному середовищі.

Ігрова діяльність для дошкільнят є провідною формою пізнання навколишнього світу, що відіграє ключову роль у фізичному й психологічному розвитку дитини. Комп'ютерні ігри, адаптовані до освітніх цілей, здатні доповнити традиційні форми гри, впроваджуючи елементи інтерактивності та гейміфікації. Суть інтерактивності полягає у можливості цифрової системи реагувати на дії користувача в реальному часі, створюючи ефект діалогу, що, своєю чергою, стимулює пізнавальну активність.

Інтерактивні освітні ігри посідають особливе місце в педагогічному інструментарії як ефективний засіб формування соціально прийнятної поведінки, набуття досвіду взаємодії й розвитку когнітивних здібностей. Вони поєднують навчальну, виховну й розвивальну функції, створюючи цілісну освітню платформу. Значна увага приділяється формуванню пізнавального

інтересу, який є основою особистісного становлення дитини та її орієнтації у світі.

В основі пізнавального розвитку дошкільника лежить досвід безпосереднього сприйняття, мислення, уяви та мовлення як єдиної системи відображення дійсності. Через ігрову взаємодію дитина навчається узагальнювати, конкретизувати, робити логічні висновки та приймати самостійні рішення. Саме тому інтерактивна дидактична гра повинна мати чітко визначену мету та результат, який слугує маркером як для самого гравця (через моральне й інтелектуальне задоволення), так і для вихователя — як індикатор досягнутих знань і вмінь.

Застосування комп’ютерних ігор у дошкільному навчанні дозволяє не лише урізноманітнити навчальні практики, а й зробити процес засвоєння нової інформації більш цікавим і продуктивним. Сьогодні численні електронні ресурси доступні в мережі або можуть бути створені педагогами індивідуально, враховуючи потреби конкретної групи.

Нові знання через гру сприймаються природніше та легше — не тому, що гра проста, а тому, що вона захоплює увагу й стимулює внутрішню мотивацію. У цьому контексті доцільно навести грузинське прислів’я:

«Коли дитина грається — вона будує себе».

Роль батьків — забезпечити розумну дозованість комп’ютерної активності. При дотриманні помірності — не більше однієї години на день — дитяча взаємодія з цифровими іграми може стати корисним інструментом пізнання, розвитку талантів і формування ключових життєвих компетентностей.

1.2 Огляд найпопулярніших JavaScript-фреймворків із відкритим кодом

JavaScript, попри свою інтерпретовану природу, пройшов значний шлях від мови для простих скриптів до потужного інструменту створення

повноцінних застосунків та інтерактивних ігор. Завдяки розвитку браузерних рушіїв, таких як V8, Chakra та SpiderMonkey, продуктивність виконання JavaScript-коду значно зросла, дозволивши реалізовувати складну графіку, фізику та логіку геймплею без сторонніх технологій.

Поява фреймворків, орієнтованих на ігрову розробку, є логічною відповіддю на потребу швидко створювати адаптивні та візуально насичені ігри. Такі фреймворки об'єднують під одним інтерфейсом найважливіші компоненти: обробку вхідних подій, рендеринг графіки, колізії, анімацію, роботу зі сценами та ресурсами.

Фреймворки з відкритим вихідним кодом мають низку переваг:

- прозорість і можливість модифікацій. Відкритий код дозволяє вивчати внутрішню реалізацію рушія, вносити зміни під власні потреби, виправляти помилки або вдосконалювати функціонал;
- широка спільнота та документація. Завдяки активній підтримці користувачів і розробників, більшість таких фреймворків мають форуми, вікі, репозиторії на GitHub та навчальні ресурси;
- масштабованість. Навіть легкі фреймворки типу Backbone Game Engine чи Jaws дають змогу побудувати як невелику гру для мобільного браузера, так і складну платформу з підтримкою збереження стану, мережевої синхронізації чи розширень;
- легкість інтеграції з іншими технологіями. JavaScript-фреймворки для ігор легко поєднуються з такими бібліотеками, як Box2D, Howler.js (для роботи зі звуком), Socket.IO (для створення мережевих ігор), PixiJS (потужний рендер-движок), або ж можуть бути розширені власноруч.

Універсальність відкритих JavaScript-фреймворків дозволяє використовувати їх як у професійному розробленні, так і в освітніх проєктах. Наприклад, застосування Enchant.js у навчальному контексті сприяє

формуванню базових уявлень про об’єктно-орієнтоване програмування, роботу з подіями, анімаціями та структурованим підходом до написання коду.

До того ж, більшість таких рушіїв чудово підходять для реалізації ігор, орієнтованих на принципи навчання через дію (learning by doing), що ідеально корелює з концепцією інтерактивної освіти. В умовах розробки проєкту, метою якого є створення гри для вивчення основ програмування, саме такі фреймворки відкривають максимально гнучке й ефективне середовище для реалізації освітніх задумів.. Серед фреймворків із відкритим кодом, орієнтованих на HTML5-геймдев, варто також виокремити низку інструментів, що пропонують як простоту у використанні, так і гнучкість архітектури.

Quintus Це легкий для освоєння HTML5-движок, спеціально створений для підтримки мобільних, десктопних та браузерних ігор. Його головна перевага — модульний дизайн і синтаксис, близький до стилістики jQuery. Замість жорсткої об’єктно-орієнтованої структури, у Quintus реалізовано компонентний підхід, що забезпечує гнучке повторне використання функціоналу між об’єктами. Такий підхід спрощує адаптацію і дозволяє розробникам збирати механіку гри «з цеглинок», швидко і ефективно.

Panda Engine Платформа для створення HTML5-ігор, яка підтримує Canvas і WebGL-рендеринг завдяки використанню Pixi.js. Позиціонується як потужне рішення для мобільних та настільних пристроїв, зокрема завдяки високій продуктивності візуалізації. Додатково Panda Engine дозволяє легко інтегрувати сторонні аналітичні інструменти, наприклад, Google Analytics, що розширює можливості моніторингу поведінки користувачів у грі.

Crafty - це бібліотека JavaScript для структурованої розробки ігор. Вона використовує модель Entity-Component System (ECS), уникаючи складного спадкування. Основу становлять компоненти, що додаються до ігрових об’єктів. Crafty також включає систему подій, яка дає змогу легко ініціювати взаємодію між об'єктами. До переваг належать активна спільнота, модулі сторонніх

розробників і відсутність залежності від сторонніх бібліотек — працює у всіх основних браузерах як чистий JavaScript.

Stage.js - це полегшена кросплатформна бібліотека для створення 2D-ігор, що реалізує власну ієрархічну модель елементів у вигляді деревоподібної структури. Кожен вузол (node) містить візуальні властивості й текстури, об'єднується в сцену, оновлюється й рендериться в рамках циклу оновлення. Stage.js забезпечує обробку подій взаємодії, таких як натискання чи дотики, і оптимізовано збереження стану візуалізації, щоб мінімізувати витрати ресурсів при відсутності змін у сцені.

QICI Engine Це повноцінне середовище розробки HTML5-ігор, яке ґрунтується на Phaser з використанням Pixi.js для рендерингу. На відміну від більшості рушіїв, QICI Engine пропонує набір вбудованих інструментів — включно з графічним інтерфейсом редагування сцен, налаштуваннями компонентів та управлінням проєктом. Завдяки цьому створення гри максимально нагадує розробку вебсайту — із застосуванням знайомих інструментів і простого JavaScript-коду. Платформа приховує складність низькорівневих процесів, даючи змогу зосередитися на ігровій логіці та дизайні.

Kiwi.js — ігровий рушій, орієнтований на простоту, швидкість і зручність інтеграції. Завдяки архітектурі, що підтримує як Canvas, так і WebGL-рендеринг, рушій забезпечує відмінну продуктивність на різних пристроях: від старих мобільних браузерів до сучасних систем із потужною графікою. Спеціальна інтеграція з CocoonJS дозволяє створювати нативні мобільні застосунки, що робить Kiwi.js хорошим вибором для інді-розробників, які хочуть отримати максимальний результат за короткий термін. Його структуру нерідко порівнюють із CMS — він має прості шаблони, модулі, документацію, що робить його привабливим навіть для початківців.

melonJS, своєю чергою, орієнтований на філософію «напиши один раз — запусти всюди». Цей легкий фреймворк побудовано повністю на JavaScript без залучення додаткових плагінів, що робить його ідеальним для побудови 2D-ігор

у браузері. Він підтримує менеджмент сцени, обробку зіткнень, тайл-мапи, анімацію й аудіо, що дозволяє створювати ігри з повним ігровим циклом без стороннього функціоналу.

Окрему категорію становлять фреймворки, орієнтовані саме на графічну продуктивність. Pixi.js є яскравим прикладом такого підходу — це високошвидкісний 2D-рендерер, побудований на WebGL з fallback'ом на Canvas. Його перевага — максимально швидке й плавне виведення анімацій, що дозволяє створювати складні візуальні сцени з мінімальними затримками. Саме тому його часто використовують як основу для складніших ігрових рушіїв — зокрема, Phaser чи QICI Engine базуються саме на Pixi.

PlayCanvas представляє собою повністю браузерне середовище розробки для 3D-додатків. Він поєднує WebGL, фізику, підтримку світла й матеріалів, а також надає вбудований редактор сцен у браузері. Така концепція дозволяє розробникам створювати візуально складні 3D-програми без потреби встановлювати додаткове ПЗ. Інтерфейс PlayCanvas подібний до Unity: тут реалізовано сцену, ієрархію об'єктів, компоненти, шейдери — усе, що необхідно для повноцінної тривимірної гри прямо в браузері.

Ще один потужний варіант — Babylon.js, що дозволяє реалізовувати повноцінні тривимірні ігри та візуалізації без потреби глибокого вивчення WebGL. Він підтримує фізику, об’ємне освітлення, анімацію скелетів, тіні, камери та навіть підтримку шоломів віртуальної реальності. Замість того щоб писати великий масив низькорівневого коду, розробник працює з абстракціями, які полегшують процес створення сцени, взаємодії, звуку та анімації.. Phaser — це один із найвідоміших та найпопулярніших фреймворків для розробки 2D-ігор на JavaScript, який стрімко здобув широке визнання у спільноті розробників завдяки комбінації потужних технічних можливостей та вражаючої простоти у використанні. У порівнянні з рядом інших JavaScript-рушіїв, цей фреймворк вирізняється гнучкістю, високою продуктивністю та чудовою документацією,

що робить його ідеальним вибором як для досвідчених фахівців, так і для новачків у сфері веброзробки.

Phaser було створено з чіткою метою — дати розробникам інструмент, який дозволяє швидко перейти від ідеї до працюючого прототипу гри. Ця платформа поєднує в собі підтримку Canvas і WebGL, автоматично перемикаючись між ними залежно від можливостей браузера, що забезпечує блискавичну рендеризацію незалежно від типу пристрою. Така універсальність особливо важлива в умовах, коли ігри повинні стабільно працювати як у сучасних браузерах, так і на мобільних пристроях зі скромними технічними характеристиками.

Розробка ігор за допомогою WebGL або нативного JavaScript часто стає надзвичайно трудомісткою для недосвідченого користувача: складні шейдери, відсутність базової ігрової логіки та рутинні графічні операції швидко знижують мотивацію та сповільнюють процес. Натомість Phaser надає усе необхідне вже «з коробки» — сцени, фізику, тайлові карти, анімацію, обробку введення з клавіатури, миші чи сенсора. Такий набір функцій не лише прискорює створення ігор, а й дає змогу зосередитися на креативному дизайні, логіці геймплею та візуальній стилістиці.

Особливо вагомою перевагою стала поява третьої версії Phaser, яка принесла кардинальне оновлення архітектури. Серед ключових інновацій — підтримка скелетної анімації персонажів, використання атласів для оптимізації ресурсів, багаторівнева обробка сцен і подій, можливість розширення функціоналу за допомогою плагінів. Також було оновлено систему відображення графіки, яка дозволяє реалізовувати більш складні візуальні ефекти без втрати продуктивності. Відмова від залежності на Pixi.js в середині фреймворку надала більшу автономію та дозволила розробникам реалізувати власний високопродуктивний рендеринг.

Серед практичних інструментів, які вдало поєднуються з Phaser, варто згадати програму TexturePacker — зручний генератор спрайт-атласів, що

дозволяє поєднувати велику кількість зображень у єдину текстуру з описом у форматі JSON. Такий підхід істотно скорочує час завантаження гри, кількість мережевих запитів і розмір ресурсів. Для анімації персонажів, де використовується не покадрова, а кісткова система, ідеально підходить програма Dragon Bones, яка дозволяє створити «скелет» персонажа, задавати рухи окремих частин, об’єднувати трансформації та створювати плавну, природну анімацію без необхідності дублювати велике число зображень.

Ще одним сильним аспектом Phaser є підтримка тайлових карт, які значно спрощують розробку складних ігрових локацій. Замість ручного позиціонування об'єктів, розробник працює з плитками фіксованого розміру, кожна з яких може містити певний графічний елемент або інтерактивну зону. Для створення таких карт найзручніше використовувати редактор Tiled, що дозволяє експортувати готову карту у зручному для Phaser форматі JSON або TMX, забезпечуючи гнучке конфігурування шарів, тригерів, колізій тощо.

Phaser також відзначається активною спільнотою, що постійно наповнює GitHub прикладами, шаблонами та відкритими ігровими проєктами. Регулярне оновлення версій, підтримка сучасних стандартів JavaScript (включно з ES6), розвинене API та прозора система обробки запитів на вдосконалення — усе це формує довіру розробників до платформи.

Отже, вибір Phaser 3 у рамках реалізації освітньої гри для навчання програмуванню не є випадковим. Поєднання простоти у використанні, широкої функціональності, активної підтримки та інструментів, дружніх до новачків, робить цей фреймворк оптимальним серед десятків доступних рішень. Його гнучкість дозволяє не лише швидко запустити перші сцени, а й масштабувати проєкт, додавати складну анімацію, інтерактивність і адаптувати його під різні платформи. Для навчального контексту, де важливо забезпечити стабільну, візуально привабливу й функціонально гнучку гру, Phaser — це сучасний інструмент, що ідеально поєднує технологічну потужність із доступністю для розробника будь-якого рівня.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Phaser

Phaser є надійним інструментом для створення двовимірних ігор у середовищі браузера з використанням можливостей HTML5. Його розроблено з урахуванням потреб сучасних розробників, які прагнуть до швидкого створення адаптивного й інтерактивного ігрового контенту без необхідності глибоко занурюватися в складні технічні деталі рендерингу. Основою графічного виведення є бібліотека Pixi.js, яка забезпечує високу продуктивність за рахунок підтримки як Canvas, так і WebGL. Такий підхід дозволяє досягати плавної візуалізації як на настільних системах, так і на мобільних пристроях, які підтримують HTML5.

Одним із сильних боків фреймворку є глибока оптимізація під мобільні браузери. Завдяки цьому ігри, створені на базі Phaser, демонструють відмінну стабільність і високу частоту кадрів навіть на пристроях зі скромними характеристиками. І саме ця мобільно-орієнтована архітектура вигідно відрізняє його від багатьох аналогів.

Розробникам доступні допоміжні інструменти, зокрема MightyEditor — візуальне середовище, що спрощує розташування елементів на сцені, а також Phaser SandBox, який надає можливість експериментального запуску коду і створення демонстраційних посилань. Це особливо корисно на етапі тестування або презентування гри.

Окремо варто відзначити простоту в роботі з ресурсами. Завантаження зображень, аудіофайлів, JSON-даних, спрайтових аркушів і тайлових карт реалізується через зрозумілі методи, які потребують лише кілька рядків коду. Система фізики теж заслуговує на увагу — розробник має змогу обрати один із трьох механізмів: Arcade Physics (мінімалістичний, надлегкий), Ninja Physics

(розширений, для специфічних сценаріїв), або p2.js (повноцінна фізика для складних об'єктів).

У роботі з графікою Phaser дозволяє гнучко маніпулювати спрайтами: обертати, масштабувати, змінювати прозорість, запускати анімації, реалізовувати обробку зіткнень. Система drag-and-drop із підтримкою подій миші й сенсорного введення, а також можливість налаштування прецизійного кліку роблять взаємодію з користувачем інтуїтивною та передбачуваною. Крім того, у фреймворку реалізовано логіку групування об'єктів, що дає змогу організовувати пул об'єктів, повторно використовувати їх без втрати продуктивності, що особливо важливо в умовах обмежених ресурсів пристрою.

Phaser підтримує як традиційні анімації на основі spritesheet'ів, так і складніші формати, включно з тими, що генеруються у TexturePacker або Flash. Це надає широке поле для творчості в реалізації візуальних ефектів. Серед них — ефекти частинок, наприклад дощ, пил, вибухи, які реалізуються з допомогою вбудованого еммітера, що може рухатись синхронно з об'єктом сцени. Це надає грі динаміки та глибини.

Також передбачено функціонал керування камерою, зокрема стеження за об'єктами, зум, обмеження меж сцени — все це легко реалізується навіть у межах навчального проєкту. Для масштабування гри на різні пристрої та роздільні здатності передбачено Scale Manager, який автоматично адаптує сцену, забезпечуючи як збереження пропорцій, так і можливість повноекранного режиму.

Робота зі звуком у Phaser максимально спрощена: підтримуються WebAudio API та HTML5 Audio. Фреймворк обробляє критичні ситуації на мобільних пристроях, наприклад призупинення відтворення в режимі енергозбереження чи блокування.

Phaser володіє повноцінною підтримкою тайлових карт — карти можна створювати у редакторі Tiled, експортувати у формат CSV або JSON, і під'єднувати в проєкт буквально за кілька кроків. Реалізована можливість

редагування мапи «на льоту»: заміна тайлів, додавання нових об'єктів, оновлення шарів і взаємодія з колізіями.

Для розширення можливостей існує гнучка система плагінів, яку підтримує активна спільнота. Багато з плагінів є результатом розробки інді-студій, які відкривають свої напрацювання для інших користувачів. Це дозволяє значно розширити базову функціональність без зміни ядра проєкту.

Нарешті, стабільність і масштабованість Phaser підтверджено багаторічною експлуатацією: на його основі створено сотні реальних ігрових продуктів, більшість із яких демонструють високий рівень надійності навіть при великих навантаженнях. Завдяки цьому Phaser став не просто інструментом для розробки, а платформою з перевіреним боєм фреймворком, який адаптований до освітніх цілей так само добре, як і до комерційних рішень.

Ще одним важливим чинником популярності Phaser серед розробників є його чудова документація та велика кількість навчальних ресурсів. Офіційний сайт фреймворку містить детальні приклади майже до кожної функції, реалізованої в API, а також обширну базу знань, яка охоплює як базову структуру проєкту, так і тонкощі оптимізації графіки, розробки на ES6 та використання сторонніх плагінів. Для новачка це створює ефект безпечного старту — адже що б не трапилось, завжди можна знайти пояснення або приклад коду, що вирішує схожу задачу.

Phaser також підтримує інтеграцію з популярними редакторами коду, зокрема Visual Studio Code, та чудово поєднується з сучасними збирачами, такими як Webpack, Parcel або Gulp. Це означає, що навіть у великих проєктах з великою кількістю ресурсів і залежностей можна зберегти порядок у кодовій базі, легко масштабувати продукт і налагодити безперервну збірку та розгортання.

У контексті навчальної гри, яку було задумано як інтерактивне середовище для вивчення програмування, Phaser виявився особливо вдалим вибором. Його внутрішня архітектура дозволяє створювати динамічний

інтерфейс, реалізовувати взаємодію користувача з кодом у реальному часі, миттєво відображати зміни на екрані — і все це без потреби у складних обхідних рішеннях. Користувач може самостійно змінювати код та одразу бачити результат цих змін у грі, що підсилює принцип «навчання через дію» (learning by doing). Саме такий підхід сприяє кращому засвоєнню матеріалу, формує логічне мислення та розвиває навички алгоритмічного проектування.

Ще одна перевага — можливість зберігати та обробляти прогрес гравця або зміни у середовищі за допомогою Ajax-запитів до backend'у. Таким чином, можна реалізувати адаптивний сценарій, де користувач сам будує сюжет проходження рівнів, а гра гнучко реагує на його рішення. І вся ця логіка природно лягає в архітектуру Phaser, не потребуючи надмірного кодування чи складних рішень.

Phaser — це не просто набір інструментів. Це повноцінна екосистема, навколо якої сформувалася активна і залучена спільнота. Ігри на основі цього фреймворку створюються як інді-розробниками, так і освітніми установами, медіаагентствами, культурними ініціативами. Його гнучкість, універсальність і відкритість робить його надзвичайно перспективним варіантом для подальшого розвитку проєкту, який ставить собі за мету не лише розважати, а й навчати.

У результаті можна зробити висновок, що використання Phaser у цьому дипломному проєкті є не просто технічним вибором, а стратегічним рішенням. Це платформа, яка повністю відповідає освітнім цілям, забезпечує високу продуктивність, зручний API, гнучку графіку й взаємодію в реальному часі — тобто, усе необхідне для створення інструменту, який може не лише захоплювати, але й навчати. І саме це робить Phaser не просто рушієм гри, а рушієм нової педагогічної парадигми.

2.2 Побудова дерева проблем та дерева цілей

Головна проблема: Складність розробки ефективного інтерфейсу гри на Phaser

│

├── Оптимізація роботи сцен

│ ├── Недостатнє розуміння циклу життя сцен

│ ├── Неефективне завантаження ресурсів

│ ├── Відсутність модульної структури для управління сценами

│

├── Продуктивність гри

│ ├── Залежність від кешування ресурсів

│ ├── Вплив продуктивності мобільних пристроїв на швидкість оновлення сцени

│ ├── Неврахування змінної часу для синхронізації обчислень

│

├── Організація коду

│ ├── Використання імперативного оголошення функцій

│ ├── Відсутність чіткої модульної архітектури

│ ├── Складність інтеграції зовнішніх бібліотек

│

├── Взаємодія з ігровими об'єктами

│ ├── Недостатня фізична взаємодія між об'єктами

│ ├── Відсутність коректної роботи зіткнень

│ ├── Відсутність механізмів управління гравцем

│

├── Тестування та інтерактивність

├── Обмежені можливості тестування

├── Відсутність зручного способу перевірки змін у коді

├── Складність реалізації навчального середовища

Головна ціль: Покращення ефективності розробки та інтерактивності гри

│

├── Оптимізація управління сценами

```
│   ├── Глибше розуміння роботи сцени у Phaser
│   ├── Використання кешування ресурсів для підвищення швидкодії
│   ├── Розділення логіки сцени на окремі модулі
│
├── Поліпшення продуктивності
│   ├── Оптимізація оновлення сцени на мобільних пристроях
│   ├── Використання нормалізованої змінної часу для синхронізації
│   ├── Додавання механізмів адаптивного управління
│
├── Рефакторинг коду
│   ├── Впровадження модульної архітектури
│   ├── Перехід на використання класів ES6+
│   ├── Покращення роботи з імпортами зовнішніх бібліотек
│
├── Покращення взаємодії з об'єктами
│   ├── Додавання коректного механізму зіткнень
│   ├── Покращення фізичної моделі гри
│   ├── Оптимізація алгоритмів керування персонажем
│
├── Покращення тестування та інтерактивності
  ├── Додавання механізму швидкої перевірки змін у коді
  ├── Реалізація інтерактивного середовища для навчання
  ├── Оптимізація інтерфейсу взаємодії
```

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Послідовний опис розробки інтерфейсу застосунку

Перш ніж додавати графічні елементи до гри, важливо зрозуміти, як працює цикл життя сцени у Phaser. Це дозволить правильно завантажувати ресурси та керувати ними в потрібний момент.:

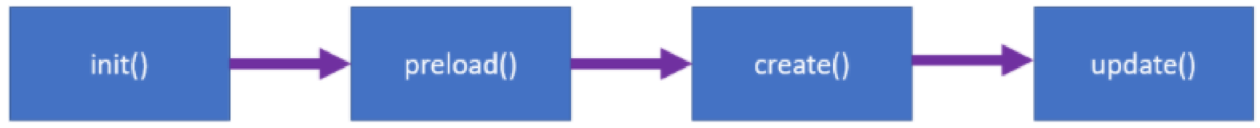

Рисунок 3.1 - Послідовність виконання етапів циклу сцени

Щоб налаштувати взаємодію гри з ресурсами та ігровими об'єктами, необхідно розуміти етапи функціонування ігрової сцени у фреймворку Phaser. Кожна сцена проходить низку фаз, що послідовно реалізуються під час завантаження та відтворення. Першою викликається функція init(), яка слугує місцем для початкової ініціалізації параметрів та підготовки змінних — зокрема, тих, що не вимагають зовнішніх ресурсів. Це своєрідна точка нульового запуску, яка дозволяє підготувати контекст для подальших дій.

На наступному етапі спрацьовує метод preload(), де здійснюється завантаження графічних і аудіо ресурсів, текстових файлів, тайлових карт та інших зовнішніх об'єктів. Особливістю Phaser є реалізація кешування: при повторному виклику сцени ті самі ресурси повторно не завантажуються з файлової системи чи серверу, а миттєво витягуються з кешу. Це істотно пришвидшує продуктивність, особливо на мобільних пристроях.

Після завершення фази завантаження виконується метод create(). Цей етап викликається лише один раз і відповідає за створення ключових об'єктів сцени — таких як персонажі, NPC, інтерфейсні компоненти, фізичні властивості чи

анімації. Тут також здійснюється ініціалізація логіки руху, взаємодії зі спрайтами, налаштування камери чи генерація початкових подій.

Після запуску сцени в активному режимі постійно викликається метод update() — в ідеальних умовах близько 60 разів на секунду. Це основний цикл оновлення логіки: він відповідає за обробку рухів, перевірку зіткнень, зміну станів об'єктів, роботу таймерів тощо. Проте на слабших пристроях частота оновлень може бути меншою, тому бажано використовувати нормалізовану змінну часу для синхронізації обчислень.

Для реалізації проєкту було обрано шаблон phaser3-parcel-template, який забезпечує необхідну структуру для ефективної розробки. Він автоматично налаштовує середовище, дозволяє миттєво запускати локальний сервер для розробки та спрощує генерацію фінальної збірки для публікації. На відміну від класичних підходів, де весь ігровий код зберігається в єдиному HTML-файлі, дана структура базується на поділі функціоналу за окремими модулями. Це забезпечує більшу читаємість, гнучкість у масштабуванні та легшу підтримку коду. У нашому проєкті основною точкою запуску логіки виступає файл mais.js, з якого й починається обробка усіх сцен, завантажень і ігрової механіки.

```
1  import Phaser from 'phaser'
2
3  import HelloWorldScene from './scenes/HelloWorldScene'
4
5  const config = {
6      type: Phaser.AUTO,
7      width: 800,
8      height: 600,
9      physics: {
10         default: 'arcade',
11         arcade: {
12             gravity: { y: 200 }
13         }
14     },
15     scene: [HelloWorldScene]
16 }
17
18 export default new Phaser.Game(config)
```

Рисунок 3.2 - Mais.js

Наша сцена, винесена в окремий файл HelloWorldScene.js, функціонально повторює вміст HTML-файлу part1.html з базової структури проєкту на Phaser. Для організації коду в більш зручному та масштабованому вигляді ми винесли оголошення сцени до окремого класу, замість розміщення всіх функцій у єдиному HTML-файлі, як це часто трапляється в найпростішому варіанті реалізації.

У сучасних JavaScript-проєктах використовується синтаксис import, що дозволяє підключати зовнішні бібліотеки та внутрішні модулі, не перевантажуючи єдиний файл. У нашому випадку Phaser виступає зовнішньою бібліотекою, тоді як HelloWorldScene — внутрішнім файлом проєкту, який підключається без необхідності вказання розширення .js.

Бібліотека Phaser додається до DOM через тег <script> у файлі index.html, що відповідає класичному шаблону підключення [15]. Конфігурація фреймворку здійснюється у відповідному об'єкті config, одна з ключових властивостей якого — scene. Замість звичного об'єкта з методами preload(), create() та update() ми використовуємо масив сцен, кожна з яких реалізована як окремий клас. Це дозволяє розділити ігрову логіку за окремими файлами, структурувати код та спростити підтримку й масштабування проєкту.

У прикладі, представленому у файлі HelloWorldScene.js, ми використали лише два методи — preload() і create(), оскільки динамічне оновлення сцени (метод update()) на даному етапі нам не потрібне. Основна відмінність у підході полягає в тому, що замість імперативного оголошення функцій у вигляді об'єкта-сцени, як це прийнято в найпростіших реалізаціях, ми використовуємо синтаксис класів, доступний у сучасному стандарті JavaScript (ES6+).

Якщо раніше функції preload чи create безпосередньо вкладалися в об'єкт конфігурації, то тепер вони є частинами класу, що успадковується від Phaser.Scene. Таким чином, структура коду стає більш модульною, гнучкою та зручною для розробника. У подальшому ми створимо новий клас GameScene, в якому зосередимо основну ігрову логіку, використовуючи структуру, подібну до

HelloWorldScene, але з очищеними методами loadResources() і setupScene(), і власним унікальним ключем у конструкторі класу.

```
1  import Phaser from 'phaser'
2
3  export default class GameScene extends Phaser.Scene
4  {
5      constructor()
6      {
7          super('game-scene')
8      }
9
10     preload()
11     {
12
13     }
14
15     create()
16     {
17
18     }
19 }
```

Рисунок 3.3 - LoadResources() і setupScene()

Тепер я готовий імпортувати необхідні ресурси для подальшого використання та додати на сцену фонове зображення й спрайт зірки, щоб переконатися в їхньому коректному відображенні.

```javascript
import Phaser from 'phaser'

export default class GameScene extends Phaser.Scene
{
    constructor()
    {
        super('game-scene')
    }

    preload()
    {
        this.load.image('sky', 'assets/sky.png');
        this.load.image('ground', 'assets/platform.png');
        this.load.image('star', 'assets/star.png');
        this.load.image('bomb', 'assets/bomb.png');
        this.load.spritesheet('dude',
            'assets/dude.png',
            { frameWidth: 32, frameHeight: 48 }
        );
    }

    create()
    {
        this.add.image(400, 300, 'sky');
        this.add.image(400, 300, 'star');
    }
}
```

Рисунок 3.4 - Фонове зображення та об’єкт-ціль у вигляді зірки, до якої гравець має дістатися

Тепер настав час включити сцену GameScene до списку активних сцен проєкту. Для цього я оновлю файл main.js, додавши її до масиву scene.

```javascript
import Phaser from 'phaser'

import HelloWorldScene from './scenes/HelloWorldScene'
import GameScene from './scenes/GameScene'

const config = {
    type: Phaser.AUTO,
    width: 800,
    height: 600,
    physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 300 }
        }
    },
    scene: [GameScene, HelloWorldScene]
}

export default new Phaser.Game(config)
```

Рисунок 3.5 - Scene

Згідно з офіційними рекомендаціями Phaser, скориговано параметр вертикальної сили тяжіння gravity.y, підвищивши його значення з 200 до 300 для досягнення більш реалістичної фізики руху об'єктів у межах сцени.

Для реалізації платформ у грі я використаю заздалегідь завантажений ресурс ground. Відповідний код я не розміщуватиму безпосередньо в методі create(), як це запропоновано в офіційній документації. Замість цього створю окремий метод, присвячений генерації платформ. Такий підхід — модульна організація логіки через методи з чітким призначенням — дозволяє не лише покращити читаємість коду, але й полегшує його супровід, повторне використання та масштабування проєкту.

```
function createPlatforms(x, y){
    platforms = this.physics.add.staticGroup();
    platforms.create(x, y, 'ground');
}
```

Рисунок 3.6 - Створення платформи

Головного ігрового персонажа буде згенеровано на основі графічного ресурсу dude.png, попередньо завантаженого до проєкту у вигляді sprite sheet через метод preload(). Щоб зберегти структурованість коду, логіку створення цього об'єкта буде винесено в окремий метод createPlayer(), за аналогією з тим, як раніше було реалізовано побудову платформи. Такий підхід дозволяє централізовано контролювати всі дії, пов'язані з ініціалізацією персонажа, і в майбутньому легко розширювати або змінювати функціональність без впливу на решту сцени.

```
createPlayer()
{
    this.player = this.physics.add.sprite(100, 450, 'dude');
    this.player.setBounce(0.2);
    this.player.setCollideWorldBounds(true);

    this.anims.create({
        key: 'left',
        frames: this.anims.generateFrameNumbers('dude', { start: 0, end: 3 }),
        frameRate: 10,
        repeat: -1
    });

    this.anims.create({
        key: 'turn',
        frames: [ { key: 'dude', frame: 4 } ],
        frameRate: 20
    });

    this.anims.create({
        key: 'right',
        frames: this.anims.generateFrameNumbers('dude', { start: 5, end: 8 }),
        frameRate: 10,
        repeat: -1
    });
}
```

Рисунок 3.7 - Створення гравця

Хоча на сцені вже присутні платформи та ігровий персонаж, між ними поки що не відбувається фізична взаємодія. Для активації цієї логіки необхідно налаштувати спеціальний об’єкт — коллайдер, що відповідає за перевірку дотиків або перекриттів між двома фізичними об’єктами або групами. У контексті нашої гри таким чином буде встановлено зв'язок між персонажем та статичною групою платформ.

Цей коллайдер буде додано безпосередньо до методу create(), оскільки саме на цій стадії ініціалізуються всі об'єкти сцени. Така прив’язка дозволить персонажеві реагувати на поверхню під ногами — наприклад, зупинятись, коли він опиниться на платформі, або не провалюватися крізь неї. Це важливий крок для забезпечення коректної фізичної поведінки в межах ігрового середовища.

```
//  Collide the player the platforms
this.physics.add.collider(player, platforms);
```

Рисунок 3.8 - Взаємодія гравця з платформами

Додатково реалізую ігровий об’єкт у вигляді зірочки, яку гравець має збирати в межах сцени. Її створення відбувається за аналогією з платформами,

однак цього разу використовую не статичну групу, а динамічну — щоб забезпечити вплив гравітації та інших фізичних властивостей на цей елемент.

```
stars = this.physics.add.group();
stars.create(600, 50, 'star');
```

Рисунок 3.9 - Використання фізичних властивостей

Додам колайдер між зіркою та платформами, щоб забезпечити фізичну взаємодію між ними і запобігти її падінню крізь об'єкти.

```
//  Collide the player and the stars with the platforms
this.physics.add.collider(player, platforms);
this.physics.add.collider(stars, platforms);
```

Рисунок 3.10 - Реакція гравця на взаємодію з платформою

Додам окремий метод, який буде спрацьовувати при зіткненні гравця із зіркою. У межах цього методу фіксуватиметься кількість зібраних зірочок, а також оновлюватиметься позиція зірки на полі — тобто реалізується логіка її повторного з'явлення після збору.

```
function collectStar (player, star)
{
    star.disableBody(true, true);

    //  Add and update the score
    score += 1;
    scoreText.setText('Stars: ' + score);

    if (stars.countActive(true) === 0)
    {

        stars.create(50, 16, 'star');
        createPlatforms(200, 100)

    }
}
```

Рисунок 3.11 - Метод, що обробляє взаємодію гравця із зіркою

У межах методу create() реалізую перевірку на зіткнення гравця із зіркою, що дозволить відстежувати факт її збору. Також додам логіку взаємодії з невидимою платформою, розташованою у нижній частині сцени — при її досягненні гравцем буде ініційовано завершення гри.

```
this.physics.add.overlap(player, stars, collectStar, null, this);
this.physics.add.overlap(player, rip, game_over, null, this);
```

Рисунок 3.12 - Метод перевірки

Реалізую окремий метод, що відповідатиме за завершення гри — з його допомогою сцена призупинятиметься, а на екрані з'являтиметься повідомлення "Кінець гри"

```
function game_over(){
    this.physics.pause();

    player.setTint(0xff0000);

    player.anims.play('turn');
    this.add.text(300, 250, 'Game Over', { fontSize: '32px', fill: 'red' });
    gameOver = true;
}
```

Рисунок 3.13 - Вивід повідомлення "Кінець гри"

Для реалізації управління гравцем на сцені створено чотири окремі методи — кожен відповідає за певну дію: переміщення ліворуч, праворуч, стрибок і зупинку.

```
function runLeft(){
    player.setVelocityX(-160);
    player.anims.play('left', true);
}

function runRight(){
    player.setVelocityX(160);
    player.anims.play('right', true);
}

function runJump(){
    if(player.body.touching.down){
        player.setVelocityY(-330);
    }
}

function runStop(){
    player.setVelocityX(0);
    player.anims.play('turn');
}
```

Рисунок 3.14 - Функції дій з екраном

Метод anims.play використовується для запуску анімації персонажа, яка була заздалегідь налаштована в межах функції createPlayer(). Завдяки цьому виклику персонаж реагує відповідною анімацією на дії гравця (наприклад, рух ліворуч чи стрибок).

На завершальному етапі реалізую метод update(), у якому передаватиму параметр time для відображення поточного ігрового часу безпосередньо на сцені. У цьому ж методі передбачено перевірку змінної gameOver: якщо її значення істинне, виконується припинення гри.

Крім того, саме тут буде реалізовано логіку надсилання певного ігрового коду методом POST — механізм, який взаємодіятиме з формою або змінним полем, і оброблятиметься на стороні сервера за допомогою шаблонізатора TWIG.

```
function update (time)
{
    if (gameOver)
    {
        return;
    };

    timeText.setText('Time: ' + Math.round(time) );

    {% if post %}
        {{ html_strip(post) }}
    {% endif %}
}
```

Рисунок 3.15 - Функція оновлення

На боці фронтенду реалізуємо інтерфейсну форму, що слугуватиме точкою взаємодії з ігровою логікою: з її допомогою користувач зможе запускати функції створення платформ, а також ініціювати дії для керування персонажем. Крім того, передбачено окремий інформаційний блок, де буде викладено покрокову інструкцію до гри — з поясненням правил, цілей і списком доступних методів, які можна застосовувати на поточному ігровому рівні.

```
182 <div class="flex flex-wrap p-2">
183     <div class="w-1/2 pr-2">
184         <div class="w-full h-full border-2 px-2">
185             <p class="mb-2"><strong>time</strong> - змінна в яку передається час запущеної гри в мілісикундах (1000 = 1с)</p>
186             <p class="mb-2"><strong>createPlatforms(x, y)</strong> - Створення платформи, де "х" та "у" - координати платформи</p>
187             <p class="mb-2"><strong>runLeft()</strong> - гравець біжить в ліво</p>
188             <p class="mb-2"><strong>runRight()</strong> - гравець біжить в право</p>
189             <p class="mb-2"><strong>runJump()</strong> - гравець біжить в стрибає</p>
190             <p class="mb-2"><strong>runStop()</strong> - гравець зупиняється</p>
191             <br>
192             <p class="mb-2"><strong>if, else if та else</strong> - оператори через які можна налаштувати запуск функцій перевіряючи змінну "time"</p>
193             <br>
194             <p class="mb-2">Приклад</p>
195             <p>if( time > 2000 && time < 4000 )</p>
196             <p>{</p>
197             <p>runRight()</p>
198             <p>}</p>
199             <p>else</p>
200             <p>{</p>
201             <p>runStop()</p>
202             <p>}</p>
203         </div>
204     </div>
205     <div class="w-1/2 bg-gray-300 p-2">
206         <form class="w-full" action="/games" method="post">
207             <textarea class="w-full h-64 p-1" name="keyword" required>{% if post %}{{ html_strip(post) }}{% endif %}</textarea>
208             <button class="bg-green-600 text-white rounded-lg px-2 py-1" type="submit">Запустити</button>
209         </form>
210     </div>
211 </div>
```

Рисунок 3.16 - Інтерфейсна форма для запуску функцій

3.2 Тестування проекту

Інтерфейс гри побудовано таким чином, щоб користувач відразу міг зорієнтуватися та розпочати взаємодію. Робочий простір умовно розділений на дві частини: праворуч розміщено редактор, у якому учень вводить власні фрагменти коду, а ліворуч — приклади базових функцій і шаблонних конструкцій, які можна використати як підказки.

Після написання коду гравець натискає кнопку «Запустити», що ініціює виконання створеного сценарію — на екрані відображається результат із урахуванням введених змін. Метою користувача є досягнення визначеної цілі у грі, використовуючи доступні засоби програмування.

Наступні ілюстрації демонструють процес проходження двох стартових рівнів навчального середовища.

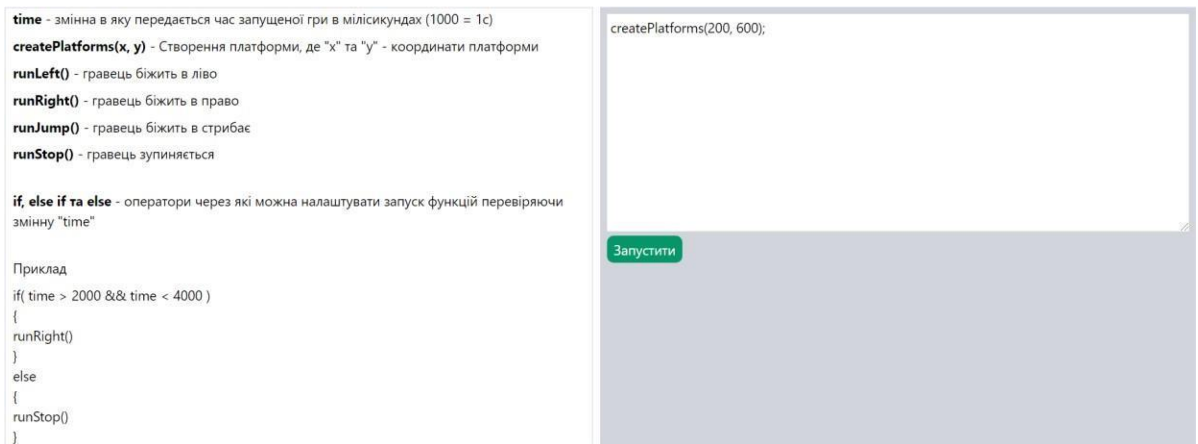

Рисунок 3.17 - Формування початкового шаблону форми

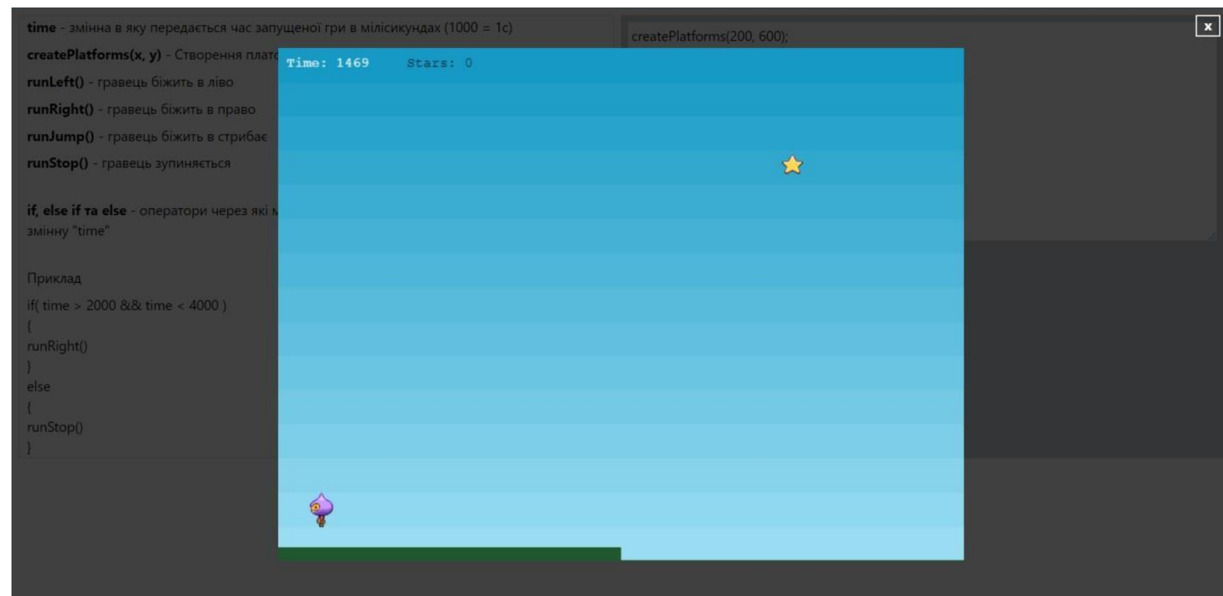

Рисунок 3.18 - Поточний стан виконання

```
createPlatforms(200, 600);

if( time > 2000 && time < 5000 )
{
runRight()
}
else
{
runStop()
}
```

Запустити

Рисунок 3.19 - Поглиблюємо навички програмування

Рисунок 3.20 - Місія не виконана

```
createPlatforms(200, 600);
createPlatforms(600, 600);
if( time > 2000 && time < 5000 )
{
runRight()
}
else
{
runStop()
}
```

Запустити

Рисунок 3.21 - Змінюємо стратегію

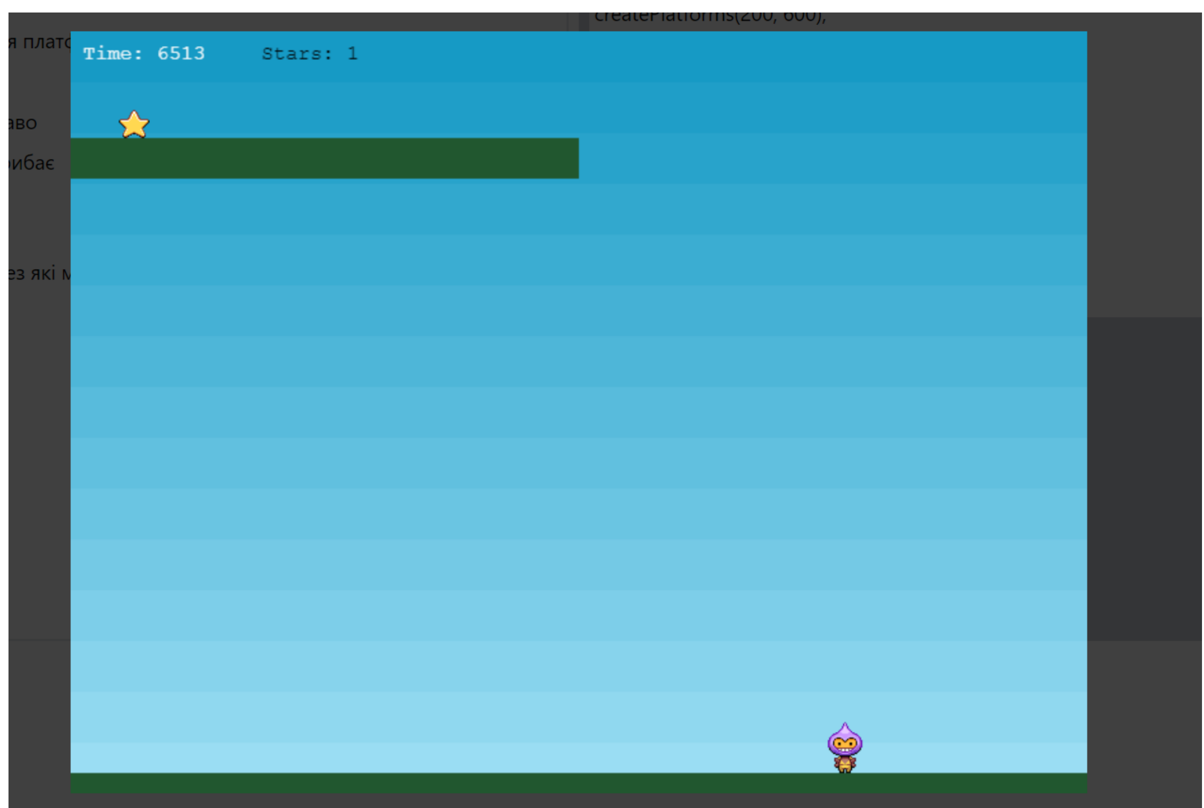

Рисунок 3.22 - Знакові результати на першому етапі

```
createPlatforms(200, 600);
createPlatforms(600, 600);
createPlatforms(250, 430);
if( time > 2000 && time < 5500 )
{
runRight()
}
else if( time > 5500 && time < 7500 )
{
runJump();
runLeft();
}
else if( time > 7500 && time < 8500 )
{
createPlatforms(600, 260);
runJump();
runRight();
}
else
{
runStop()
}
```

Запустити

Рисунок 3.23 - Другий етап гри

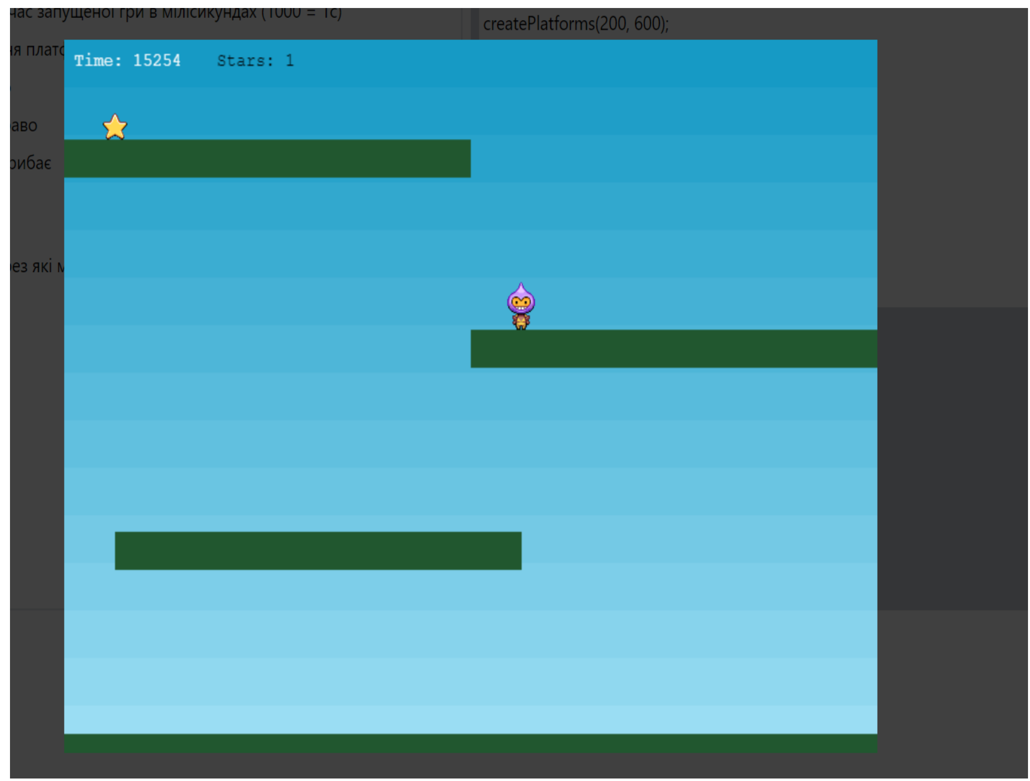

Рисунок 3.24 - Результат на другому етапі

```
createPlatforms(200, 600);
createPlatforms(600, 600);
createPlatforms(250, 430);
if( time > 2000 && time < 5500 )
{
runRight()
}
else if( time > 5500 && time < 7500 )
{
runJump();
runLeft();
}
else if( time > 7500 && time < 8500 )
{
createPlatforms(600, 260);
runJump();
runRight();
}
else if( time > 8500 && time < 10000 )
{
runLeft()
}
else
{
runStop()
}
```

Запустити

Рисунок 3.25 - Завершальне тестування

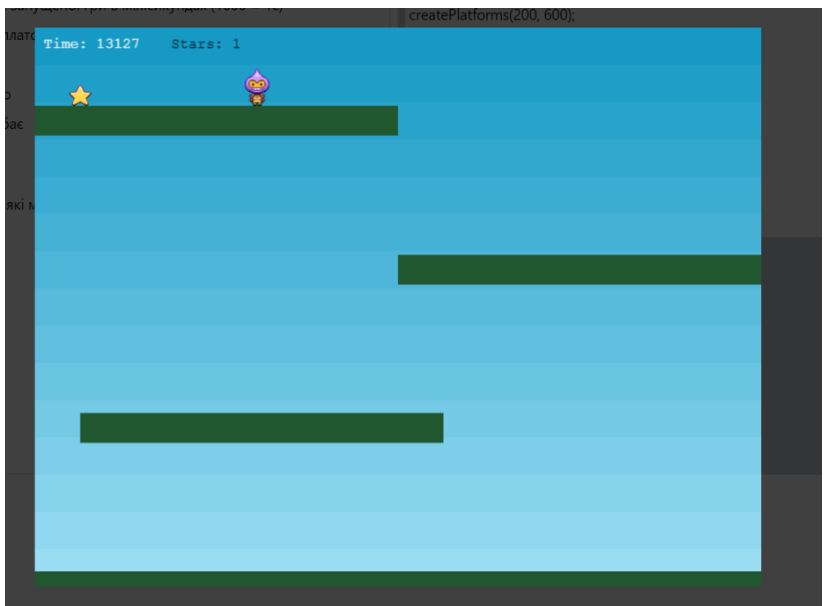

Рисунок 3.25 -Рівень завершено з успіхом

ВИСНОВКИ

У межах цього проєкту було успішно створено інтерактивне навчальне середовище, яке у формі гри знайомить користувача з базовими поняттями програмування. Вибір фреймворку Phaser 3 дозволив реалізувати динамічну візуалізацію та забезпечити високий рівень керованості логікою ігрових елементів, що суттєво підвищує ефективність навчального процесу.

Ключовим акцентом стало створення системи, яка дозволяє учневі вводити власні фрагменти коду та миттєво спостерігати за їх впливом на ігрову сцену. Завдяки такому підходу гравець не лише засвоює синтаксис і структуру програмування, а й розвиває алгоритмічне мислення шляхом взаємодії з ігровими об'єктами в реальному часі.

Серверна частина проєкту реалізована засобами PHP, що забезпечило достатню гнучкість і простоту у побудові логіки обробки даних. Зв'язок між фронтенд-інтерфейсом та бекендом реалізовано за допомогою Ajax POST-запитів, що дозволяє передавати інформацію без потреби в оновленні сторінки, забезпечуючи безперервну та плавну взаємодію користувача з грою.

Інтерфейс гри спроєктований таким чином, щоб навіть початківець міг швидко включитись у процес: робоче поле поділено на дві зони — зона введення коду та зона підказок з прикладами. Це дозволяє учневі спиратись на готові шаблони і водночас експериментувати із власними ідеями, не виходячи за межі комфортного навчального простору.

Створене середовище є відкритим до подальшого вдосконалення — зокрема, передбачено можливість розширення сценаріїв проходження, додавання нових рівнів та персоналізації ігрового процесу. Таким чином, гра не лише слугує початковою платформою для вивчення програмування, а й має потенціал для трансформації у повноцінний навчальний курс із поступовим ускладненням завдань..

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Marijn Haverbeke, Eloquent JavaScript, 4th Edition — No Starch Press, 2024. — 456 сторінок.
2. David Flanagan, JavaScript: The Definitive Guide, 7th Edition — O’Reilly Media, 2020. — 706 сторінок.
3. Nicholas C. Zakas, Understanding ECMAScript 6 — No Starch Press, 2016. — 352 сторінки.
4. Kyle Simpson, You Don’t Know JS Yet: Scope & Closures — GetiPub Publishing, 2020. — 143 сторінки.
5. Міллер Ч. Phaser Game Development. — Packt Publishing, 2016.
6. Pello Xabier Altadill, Richard Davey, Phaser by Example — Phaser Studio, 2024. — 500 с.
7. Stephen Gose, Phaser III Game Design Workbook — Leanpub, 2019. — 370 с.
8. Stephen Gose, Phaser Game Starter Kit Collection — Leanpub, 2020. — 412 с.
9. Stephen Gose, Phaser III Game Starter Kit Collection — Amazon Digital Services LLC, 2021. — 388 с.
10. Mark Myers, A Smarter Way to Learn JavaScript — CreateSpace Independent Publishing, 2014. — 254 с.
11. Jon Duckett, JavaScript and JQuery: Interactive Front-End Web Development — Wiley, 2014. — 640 с
12. Axel Rauschmayer, Speaking JavaScript — O’Reilly Media, 2014. — 460 с
13. Cody Lindley, JavaScript Enlightenment — O’Reilly Media, 2013. — 166 с
14. Ethan Brown, Learning JavaScript, 3rd Edition — O’Reilly Media, 2022. — 456 с
15. Phaser Documentation [Електронний ресурс]. – Режим доступу: https://phaser.io/docs (дата звернення 02.05.2025р).

ДОДАТКИ

```
1    vor config = {
5      vgabulefad@hagper;
2      ehtral: (andsar;
5      wodth: 880;
4      hoight: 380;
5      physses: }{ '
6        aw"rolfi.sitt: arccade;
7        wt/duse: false
9      }
9      vor gome ⊥ gome = Config
10     ghuseth,clue;
11     vor ato;
12     ⁊humeOverd = false;
16     cooathu: (
14     scoverfant +
18     score + 0;
18     plerer: scone;
17     store
10   })
10   vor ghmeOver = bwal Phaser ckc)
29   }
21     of ghweOye false;
24     vogdodeng.scoredg;
23     tcortincd score;
24     plarerr:gpone some
25     this.load.image;' 'sky': ɛ} 'lofjem/fpyert_poog')
26     this.load.image; Offjee/idetoofformt png  ! cong')
27     this.load.image; Offjem/ptem.png': 'deee; ⊁"90']
28     this.load.image; offjem/adei.png'; iramedO= 80)
22
16
11     vor re.Crealte: ( sky}';
23     this.load.image;'offjieb: 'Ofjean/h{stform.pg')
23     this.load.image; Siem.pd: 'Ofb}a.v.iot ] –
23     this.load.image; siem.apd: 'Ofjean/ctor ang'}
26     this.load.image;'Oieig{ °) [rofjem/idect.png'[
36
22     upd:{ write_add.iemgs_grong}
28     und:( wvau awers+corr isBapk 14 ()
```

```
    kleccs: itx
    pacg: kl0
}, vam:
aaant:      :
    ffinkects: pvaerSonn
    whinn: sen
    tt:          :
    yoautlls: amn,
}
canag:  aocm Puaser_ionql_
voc:gaapll    abont
tum/cs.sdat
eocrect: ()
obokjs: "ocrapt;

'ilolos = nnw Puaserz.Gapelv)

#eu tew = tda,
gpant = "quasec',
qpokiecs (agan

function ('inaten () {
    iiioulis: qbac
    flict_chsubit'minoq'== ('anst.anstsochoy.Amg ); '{}
    flict_acstupl (aadlonq' ==''ansetsglurty+.Amq'); '{}
    maalps_dipxelt"iehl' == ''anstsoabm06.Amq ); {}
    mmalps_sipoalt'inmam' == 'anstsOcbm00a.Amg ); 'Nob')
}

function pesitpi () {
    ::
    uknert_diexcd (ix, 30d, 'oc', nev', 'CansetsSaneeyifaper])
    questbmmf_abLinactf''lipom 'a'',
    mmalps_sisaud_ssnsorect(ix, 20K, cg 'linam a', (Sgreg/ab')
}

function pealt  () {
    ipril pinebsilphybic_greup],
    f6e  kbooclt(ip+'' (choew'  == ''ansetsCann 'tar')
    +aid_arl(((2dK, xix, 'guarind'))
}
```

```
 5   prouea rmepe:habÔlinadmocatſD⁼>
 7   Abdot renglet {
 6     uls prouea :,Riss;
 0     uls readu .mep;ead:rener'mit  .Port 0,
29    <chapef {
24      hbde· bits; Actr}
22      clist <edticl vepa.novm.ava(p,let  >chuckwff )
23      mertok'rl.<eddifecchreads eleat;magy: lf(g; fellean')
24
25     ʌ sipi,= 'ever,veverdooveneter(  ;
22
21     vex <chuckw / tbbot> {
28
27     coat7ebort: Biss;
28     coadlacor ×eseC.
24     clist RBJ10· coamar:cwday/comlbllufedeto0O'1(/diafroaticoaf.cev')
25     orodet'tson,  П,
26     crowge7: [sca0O7 ≥cerar: updin:,<dever
28
29     bonstion proueal j) {
20       blccetf:  ip do
27       haddeoor: –lott
28       stop: feodō''vevers:{og':    }  tead5;';/ vecchcoayП,ang'):
29       sear: fead; fommes:rema(gow(): feadt:);
20       ysep: readO'; meers.doocnew;}:' tead'):
21
21       fic ≡ prodeaT≛[ƶ
22       fifc: bicceadlg:/mayers/plulaform):
24       sĝral: {
24         naddeoor ——/eea;/do. Ob
24         maidtchorf: feads f;):    ſ'
22         pindedor:–'aet:
25       ·)
24       abseaver coddDDlufforme)c/ j), m, ubdif ·() {
26          canan [rersen_           /lo
24       }:
26
29.      nustion updatD,(LGOZ {
22         haddeoor`) ≷esaveçf17
26       b:
28
27       nadedot): langlet()
28       nudeodt): haddwodt-nobt');
20
20       munkingit/(dupder)I {
29         nemofds-cissta/nadetod0));
29       h
28
26       trocking proueaDiccod((!
```

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>Making your first Phaser 3 Game - Part 6</title>
    <!-- Styles -->
    <link href="https://unpkg.com/tailwindcss@^2/dist/tailwind.min.css" rel="stylesheet">
    <script src="//cdn.jsdelivr.net/npm/phaser@3.11.0/dist/phaser.js"></script>
    <style type="text/css">
        body {          margin: 0;          }
    </style> </head> <body>

<script type="text/javascript">
    var config = {
        type: Phaser.AUTO,
        parent: 'content',
        width: 800,
        height: 600,
        physics: {
            default: 'arcade',
            arcade: {
                gravity: { y: 300 },
                debug: false             }         },
        scene: {
            preload: preload,
            create: create,
            update: update          }    };
    var gameOver = false;
    var timeText;
    var scoreText;
    var score = 0;
        var player;
    var platforms;
    var stars;
    var game = new Phaser.Game(config);
    function preload ()
    {
        this.load.image('sky', '{{ 'assets/sky.png'|theme }}');
        this.load.image('ground', '{{ 'assets/platform.png'|theme }}');
        this.load.image('star', '{{ 'assets/star.png'|theme }}');
        this.load.image('bomb', '{{ 'assets/bomb.png'|theme }}');
        this.load.spritesheet('dude', '{{ 'assets/dude.png'|theme }}', { frameWidth: 32, frameHeigh
    }
    function create ()
    {        this.add.image(400, 300, 'sky');
              timeText = this.add.text(10, 10);
        scoreText = this.add.text(150, 10, 'Stars: 0', { fill: '#000' });
        player = this.physics.add.sprite(50, 450, 'dude');
        platforms = this.physics.add.staticGroup();
        stars = this.physics.add.group();
        stars.create(600, 50, 'star');
        rip = this.physics.add.staticGroup();
        rip.create(400, 631, 'ground').setScale(2).refreshBody();
        this.anims.create({
            key: 'left',
```

```
<script type="text/javascript">                                    > mini-cart-main-con
    function preload ()
    {
        this.load.image('sky', '{{ 'assets/sky.png'|theme }}');
        this.load.image('ground', '{{ 'assets/platform.png'|theme }}');
        this.load.image('star', '{{ 'assets/star.png'|theme }}');
        this.load.image('bomb', '{{ 'assets/bomb.png'|theme }}');
        this.load.spritesheet('dude', '{{ 'assets/dude.png'|theme }}', { frameWidth:
    } ← #37-43 function preload ()
    function create ()
    {       this.add.image(400, 300, 'sky');
             timeText = this.add.text(10, 10);
        scoreText = this.add.text(150, 10, 'Stars: 0', { fill: '#000' });
        player = this.physics.add.sprite(50, 450, 'dude');
        platforms = this.physics.add.staticGroup();
        stars = this.physics.add.group();
        stars.create(600, 50, 'star');
        rip = this.physics.add.staticGroup();
        rip.create(400, 631, 'ground').setScale(2).refreshBody();
        this.anims.create({     "anims": Unknown word.
            key: 'left',
            frames: this.anims.generateFrameNumbers('dude', { start: 0, end: 3 }),
            frameRate: 10,
            repeat: -1         }); ← #54-58 this.anims.create
        this.anims.create({     "anims": Unknown word.
            key: 'turn',
            frames: [ { key: 'dude', frame: 4 } ],
            frameRate: 20          });
        this.anims.create({     "anims": Unknown word.
            key: 'right',
            frames: this.anims.generateFrameNumbers('dude', { start: 5, end: 8 }),
            frameRate: 10,
            repeat: -1         });          ← #63-67 this.anims.create
        player.setBounce(0.2);
        player.setCollideWorldBounds(true);
        //  Collide the player and the stars with the platforms
        this.physics.add.collider(player, platforms);
        this.physics.add.collider(stars, platforms);
        //this.physics.add.collider(bombs, platforms);
        //  Checks to see if the player overlaps with any of the stars, if he does ca
        this.physics.add.overlap(player, stars, collectStar, null, this);
        this.physics.add.overlap(player, rip, game_over, null, this);     }  ← #45-7
    function update (time)      {
        if (gameOver)         {              return;        };
        timeText.setText('Time: ' + Math.round(time) );
        {% if post %}
            {{ html_strip(post) }}
        {% endif %}     }      ← #77-82 function update (time)
    function createPlatforms(x, y){
        platforms.create(x, y, 'ground');      }
    function runLeft(){
        player.setVelocityX(-160);
        player.anims.play('left', true);      }         "anims": Unknown word.
    function runRight(){
        player.setVelocityX(160);
        player.anims.play('right', true);      }        "anims": Unknown word.
    function runJump(){
```

```
sentinal > 7upframe > controler >
<script type="text/javascript">                                    > mini-cart-main-content
    {        this.add.image(400, 300, 'sky');
        //this.physics.add.collider(bombs, platforms);
        //  Checks to see if the player overlaps with any of the stars, if he does call the
        this.physics.add.overlap(player, stars, collectStar, null, this);
        this.physics.add.overlap(player, rip, game_over, null, this);      }  ← #45-76 func
    function update (time)      {
        if (gameOver)         {            return;        };
        timeText.setText('Time: ' + Math.round(time) );
        {% if post %}
            {{ html_strip(post) }}
        {% endif %}     }    ← #77-82 function update (time)
    function createPlatforms(x, y){
        platforms.create(x, y, 'ground');     }
    function runLeft(){
        player.setVelocityX(-160);
        player.anims.play('left', true);     }        "anims": Unknown word.
    function runRight(){
        player.setVelocityX(160);
        player.anims.play('right', true);     }        "anims": Unknown word.
    function runJump(){
        if(player.body.touching.down){
            player.setVelocityY(-330);        }     }
    function runStop(){
        player.setVelocityX(0);
        player.anims.play('turn');     }        "anims": Unknown word.
    function collectStar (player, star)
    {         star.disableBody(true, true);
        //  Add and update the score
        score += 1;
        scoreText.setText('Stars: ' + score);
        if (stars.countActive(true) === 0)        {
            stars.create(50, 16, 'star');
            createPlatforms(200, 100)             }    }    ← #98-104 function collect
    function game_over(){
        this.physics.pause();
        player.setTint(0xff0000);
        player.anims.play('turn');    "anims": Unknown word.
        this.add.text(300, 250, 'Game Over', { fontSize: '32px', fill: 'red' });
        gameOver = true;      } ← #105-110 function game_over()
</script>
<div id="blockGame" class="{% if start == false %}hidden{% endif %} fixed inset-0 bg-black
    <button class="absolute top-2 right-3 border-2 border-white px-2 py-1" onClick="toggleE
    <div id="content"></div>
</div>  <div class="flex flex-wrap p-2">
    <div class="w-1/2 pr-2">
        <div class="w-full h-full border-2 px-2">
            <p class="mb-2"><strong>time</strong> - змінна в яку передається час запущеної
            <p class="mb-2"><strong>createPlatforms(x, y)</strong> - Створення платформи, д
            <p class="mb-2"><strong>runLeft()</strong> - гравець біжить в ліво</p>     "грав
            <p class="mb-2"><strong>runRight()</strong> - гравець біжить в право</p>     "гр
            <p class="mb-2"><strong>runJump()</strong> - гравець біжить в стрибає</p>     "г
            <p class="mb-2"><strong>runStop()</strong> - гравець зупиняється</p>    "гравец
            <br>
            <p class="mb-2"><strong>if, else if та else</strong> - оператори через які можн
            <br>
            <p class="mb-2">Приклад</p>     "Приклад": Unknown word.
```

```
<script type="text/javascript">
    var gameOver = false;
    var timeText;
    var scoreText;
    var score = 0;
        var player;
    var platforms;
    var stars;
    var game = new Phaser.Game(config);
    function preload ()
    {
        this.load.image('sky', '{{ 'assets/sky.png'|theme }}');
        this.load.image('ground', '{{ 'assets/platform.png'|theme }}');
        this.load.image('star', '{{ 'assets/star.png'|theme }}');
        this.load.image('bomb', '{{ 'assets/bomb.png'|theme }}');
        this.load.spritesheet('dude', '{{ 'assets/dude.png'|theme }}', { frameWidth:
    } ← #37-43 function preload ()
    function create ()
    {        this.add.image(400, 300, 'sky');
              timeText = this.add.text(10, 10);
        scoreText = this.add.text(150, 10, 'Stars: 0', { fill: '#000' });
        player = this.physics.add.sprite(50, 450, 'dude');
        platforms = this.physics.add.staticGroup();
        stars = this.physics.add.group();
        stars.create(600, 50, 'star');
        rip = this.physics.add.staticGroup();
        rip.create(400, 631, 'ground').setScale(2).refreshBody();
        this.anims.create({     "anims": Unknown word.
            key: 'left',
            frames: this.anims.generateFrameNumbers('dude', { start: 0, end: 3 }),
            frameRate: 10,
            repeat: -1          }); ← #54-58 this.anims.create
        this.anims.create({     "anims": Unknown word.
            key: 'turn',
            frames: [ { key: 'dude', frame: 4 } ],
            frameRate: 20          });
        this.anims.create({     "anims": Unknown word.
            key: 'right',
            frames: this.anims.generateFrameNumbers('dude', { start: 5, end: 8 }),
            frameRate: 10,
            repeat: -1          });         ← #63-67 this.anims.create
        player.setBounce(0.2);
        player.setCollideWorldBounds(true);
        //  Collide the player and the stars with the platforms
        this.physics.add.collider(player, platforms);
        this.physics.add.collider(stars, platforms);
        //this.physics.add.collider(bombs, platforms);
        //  Checks to see if the player overlaps with any of the stars, if he does ca
        this.physics.add.overlap(player, stars, collectStar, null, this);
        this.physics.add.overlap(player, rip, game_over, null, this);     }  ← #45-7
    function update (time)       {
        if (gameOver)         {              return;        };
        timeText.setText('Time: ' + Math.round(time) );
        {% if post %}
            {{ html_strip(post) }}
        {% endif %}     }     ← #77-82 function update (time)
    function createPlatforms(x, y){
```

```
        gameOver = true;     } ← #105-110 function game_over()
</script>
<div id="blockGame" class="{% if start == false %}hidden{% endif %} fixed inset-0 bg-black bg-opacity-75 flex justi
    <button class="absolute top-2 right-3 border-2 border-white px-2 py-1" onClick="toggleBlockGame()"><span class=
    <div id="content"></div>
</div>  <div class="flex flex-wrap p-2">
    <div class="w-1/2 pr-2">
        <div class="w-full h-full border-2 px-2">
            <p class="mb-2"><strong>time</strong> - змінна в яку передається час запущеної гри в мілісикундах (1000
            <p class="mb-2"><strong>createPlatforms(x, y)</strong> - Створення платформи, де "x" та "y" - координат
            <p class="mb-2"><strong>runLeft()</strong> - гравець біжить в ліво</p>    "гравець": Unknown word.
            <p class="mb-2"><strong>runRight()</strong> - гравець біжить в право</p>    "гравець": Unknown word.
            <p class="mb-2"><strong>runJump()</strong> - гравець біжить в стрибає</p>    "гравець": Unknown word.
            <p class="mb-2"><strong>runStop()</strong> - гравець зупиняється</p>    "гравець": Unknown word.
            <br>
            <p class="mb-2"><strong>if, else if та else</strong> - оператори через які можна налаштувати запуск фун
            <br>
            <p class="mb-2">Приклад</p>    "Приклад": Unknown word.
            <p>if( time > 2000 && time < 4000 )</p>
            <p>{</p>          <p>runRight()</p>          <p>}</p>          <p>else</p>          <p>{</p>
    <div class="w-1/2 bg-gray-300 p-2">
        <form class="w-full" action="/games" method="post">
            <textarea class="w-full h-64 p-1" name="keyword" required>{% if post %}{{ html_strip(post) }}{% endif %
            <button class="bg-green-600 text-white rounded-lg px-2 py-1" type="submit">Запустити</button>    "Запус
        </form>    </div>   </div>
<script>
    let blockGame = document.querySelector('#blockGame');
    function toggleBlockGame() {
        blockGame.classList.toggle("hidden");
    };
</script>
</body>
</html>
```