

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА РОЗРОБКА ПРОГРАМНОГО
ЗАБЕЗПЕЧЕННЯ ТЕХНОЛОГІЙ БЛОКЧЕЙНУ
RESEARCH AND DEVELOPMENT OF SOFTWARE FOR BLOCKCHAIN
TECHNOLOGY

Виконав: студент(ка) 2 курсу, групи КНТ-211м
Спеціальності 122 Системи штучного
інтелекту

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Лизя Є. С.

(прізвище та ініціали)

Керівник Субботін С. О.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет комп'ютерних наук і технологій
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Лизі Євгенія Сергійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та розробка програмного забезпечення технологій блокчейну. Research and Development of Software for Blockchain Technology.

керівник проєкту (роботи) Субботін Сергій Олександрович, д.т.н., професор,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “04” листопада 2022 року
№ 363

2. Строк подання студентом проєкту (роботи) 01 грудня 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Конструювання програмного забезпечення. 4. Інструкції з використання програмного продукту. 5. Текст програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Субботін С. О., професор		
Нормоконтроль	Камінська Ж.К., ст.викладач		

7. Дата видачі завдання “11” жовтня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4-8 тижні	Розділ 2
4	Розробка програми. Тестування та експериментальне дослідження програми.	9-12 тижні	Розділ 3
5	Розробка інструкцій з використання програмного продукту	13 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	14-15 тижні	ПЗ, Додатки
7	Захист роботи	16 тиждень	

Студент(ка)

(підпис) Лизя Є. С.
(прізвище та ініціали)

Керівник проекту (роботи)

(підпис) Субботін С.О.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 87 с., 14 рис., 6 табл, 2 дод., 20 джерел.

IT-ТЕХНОЛОГІЇ, BLOCKCHAIN, SOLIDITY, SMART CONTRACTS

Об'єкт дослідження – розробка програмного забезпечення технологій блокчейну.

Предмет дослідження – засоби розробки ігрової програми з використанням технологій блокчейну Ethereum.

Мета роботи – програмна реалізація play to earn браузерної ігрової програми в жанрі платформеру з метою експертного дослідження технологій блокчейну Ethereum.

Матеріали, методи та технічні засоби: інтегроване середовище розробки Visual Studio Code, мови програмування javascript та solidity, ігровий рушій Phaser, фреймворк Hardhat.

Результати. Розроблено ігровий застосунок для браузерів з використанням технологій блокчейну Ethereum: головне меню, алгоритм бескінечної генерація ігрового рівня, смартконтракти токенів та NFT, механіка збору NFT колекцій та токенів, запуск на будь-якому веббраузері ПК.

Висновки. Розроблена програмна реалізація ігрової програми з використанням технологій блокчейну Ethereum.

Галузь використання – використання застосунків в веббраузерах з підключенням крипто гаманця MetaMask.

ABSTRACT

Explanatory note to the magistr's thesis: 87 pages, 14 figures, 6 tables, 2 appendicxes, 20 sources.

IT TECHNOLOGIES, GAME APPLICATION, 3D GAME, ANDROID GAME

The object of research is the development of software for blockchain technologies.

The subject of the study is the means of developing a game program using Ethereum blockchain technologies.

The purpose of the work is the software implementation of a play to earn browser game program in the platformer genre for the purpose of expert research of Ethereum blockchain technologies.

Materials, methods and technical means: integrated Visual Studio Code development environment, javascript and solidity programming languages, Phaser game engine, Hardhat framework.

The results. Developed game application for browsers using Ethereum blockchain technologies: main menu, game level infinite generation algorithm, token and NFT smart contracts, NFT collection and token collection mechanics, run on any PC web browser.

Conclusions. The software implementation of the game program was developed using Ethereum blockchain technologies.

The field of use is the use of applications in web browsers with the connection of the MetaMask crypto wallet.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	8
ВСТУП	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	10
1.1 Історія походження	10
1.2 Проблеми блокчейну та можливі шляхи їх вирішення	11
1.3 Види блокчейну	12
1.3.1 Публічні блокчейни	12
1.3.2 Приватні блокчейни	13
1.3.3 Консорціумні блокчейни	14
1.4 Сфери використання технологій блокчейну	14
1.4.1 Банківський сектор, інвестиції та біржі	15
1.4.2 Земельний реєстр	15
1.4.3 Посвідчення особи	15
1.4.4 Платіжний засіб	16
1.4.5 Ігрова індустрія	16
1.4.6 Онлайн голосування	16
1.4.7 Будівельний бізнес	17
1.5 Переваги та недоліки блокчейну	17
1.6 Приклад програм, які використовують технологію блокчейн	17
1.6.1 OpenSea	17
1.6.2 Decentraland	18
1.6.3 MetaMask	19
1.7 Постановка задач дипломної роботи	21
1.8 Висновки до розділу 1	22
2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	23
2.1 Аналіз вимог до системи	23
2.2 Розробка архітектури системи	23
2.3 Вибір фреймворку для розробки	26

2.3.1	Phaser.....	26
2.3.2	Node.js.....	29
2.3.3	Hardhat	30
2.3.4	Truffle.....	30
2.3.5	Порівняння Truffle і Hardhat.....	31
2.4	Обґрунтування вибору Hardhat в якості фреймворку для розробки blockchain функціоналу	33
2.5	Вибір середовища програмування.....	35
2.5.1	Visual Studio Code	35
2.5.2	Jetbrains Rider	38
2.6	Висновки до розділу 2.....	39
3	КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	41
3.1	Розробка ігрового процесу.....	41
3.2	Структура проекту	41
3.3	Опис процесі мінтинга	44
3.4	Опис програмної реалізації графічного інтерфейсу	45
3.5	Висновки до розділу 3.....	46
4	ІНСТРУКЦІЇ З ВИКОРИСТАННЯ ПРОГРАМНОГО ПРОДУКТУ	48
4.1	Системні вимоги.....	48
4.2	Функціонал системи.....	48
4.3	Інструкції з запуску системи	49
4.4	Опис роботи системи	49
4.5	Висновки до розділу 4.....	54
	ВИСНОВКИ.....	55
	ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	57
	ДОДАТОК А ТЕКСТ ПРОГРАМИ	59
	ДОДАТОК Б СЛАЙДИ ПРЕЗЕНТАЦІЇ	81

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – програмний інтерфейс;
CPU – центральний процесор;
CSS3 – Cascading Style Sheets 3;
DOM – Document Object Model;
JSON –JavaScript Object Notation;
HTML – HyperText Markup Language;
SASS – Syntactically Awesome Stylesheets;
SDK – Software development kit;
VR – віртуальна реальність;
БД – база даних;
ОС – операційна система.

ВСТУП

Сучасний світ дуже стрімко змінюється, особливо, все що зв'язано з ІТ технологіями. Одним з прикладів ІТ рішень, які призвели до інформаційного буму та революції цифрової економіки – є технологія блокчейну, яка дозволяє децентралізувати дані по всьому блокчейну і таким чином надати можливість користувачам цілком анонімно проводити фінансові транзакції. Окрім цього, технології блокчейну призвели до революції в індустрії комп'ютерних ігор, вебсайтів та в цілому серед цифрового контенту. На базі блокчейну будується Web 3.0 концепція, яка включає в себе децентралізацію та економіку на основі токенів.

На сьогодні розробка блокчейн застосунків має великий запит в різних сферах життєдіяльності, це можуть бути як і смартконтракти для автоматизації фінансових операцій, так і ігри, де використовуються токени як частина ігрового всесвіту. Розробка блокчейн застосунків є досить комплексним процесом та вимагає знань інформаційної безпеки.

Ціллю даної роботи є розробка play to earn ігрового застосунку на технології блокчейну, користуючись яким гравці можуть заробляти справжні гроші.

Вимоги до розроблюваного застосунку:

- ігрова механіка, де користувач стріляє по противникам
- система меню навігації
- алгоритм рандомного появи противників;
- смарт контракт, який забезпечує проведення транзакцій у випадку перемоги або програшу
- оптимізації під браузер на ПК

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Блокчейн (Blockchain або Block Chain) — вибудований за певними правилами безперервний послідовний ланцюжок блоків, що містять інформацію. Найчастіше копії ланцюжків блоків зберігаються на безлічі різних комп'ютерів незалежно один від одного. Вперше термін з'явився як назва повністю реплікованої розподіленої бази даних, реалізованої в системі «біткоїнів», через що блокчейн часто відносять до транзакцій в різних крипто валютах, проте технологія ланцюжків блоків може бути поширена на будь-які взаємопов'язані інформаційні блоки [1].

1.1 Історія походження

Вперше протокол, подібний до блокчейну, запропонував американський криптограф Девід Чаум у своїй дисертації 1982 року «Комп'ютерні системи, створені, підтримувані та користуються довірою взаємно підозрілих груп» (англ. Computer Systems Established, Maintained, and Trusted by Mutually). Далі, С. Хабер і У. Скотт Сторнетта у 1991 році у своїй роботі описали криптографічно захищений ланцюжок блоків. Математики мали на меті впровадити систему, в якій тимчасові мітки документів не можна було б підробити. У 1992 року Хабер, Сторнетта і Дейв Бейер включили до своєї технології дерево хешей, що підвищило її ефективність, дозволивши збирати кілька сертифікатів документів на один блок]. Зрозумівши комерційний потенціал технології, дослідники створили сервіс тимчасових міток під назвою Surety, щоб привести свою схему в дію. Хеші сертифікатів документів Surety з 1995 року щотижня публікуються у New York Times [1].

У 2008 році розробник під псевдонімом Сатосі Накамото (реальна особа залишається невідомою, можливо, що під цим ніком працювала група) запропонував загальний алгоритм системи біткойн, ключовим елементом якої

була система з безперервного послідовного ланцюжка блоків інформації, названої blockchain. Принциповою відмінністю від усіх попередніх варіантів подібних технологій (у тому числі Hashcash) стало об'єднання ланцюгового хеша з формальним механізмом вироблення консенсусу про коректність чергового блоку, що дозволило у всій системі відмовитися від необхідності верифікації інформації довіреним агентом (адміністратором) і система загалом стала децентралізованою [1].

У 2009 році було запущено першу версію криптовалюти біткойн з реалізацією децентралізованого блокчейну, який забезпечує зберігання всіх транзакцій у системі [1].

1.2 Проблеми блокчейну та можливі шляхи їх вирішення

Будучи технологією побудови масово-розподільної бази даних, блокчейн має ряд специфічних проблем, які затрудняють його використання. Серед цих проблем можна назвати такі [2]:

- постійний розмір файлів блокчейна
- обмеження пропускної здатності каналів зв'язку між узлами мережі та складності синхронізації окремих реплік, пов'язаних з цим обмеженням
- загальне обмеження продуктивності блокчейна, пов'язане зі специфікою роботи алгоритмів консенсуса

Розробка нових видів блокчейна зачастою пов'язана з подоланням або обходом цих проблем і обмежень. При цьому є ряд функцій, без яких не може обійтись ні одна система блокчейна:

- дані зберігаються в структурі з цепочки блоків, в якій кожен блок пов'язаний з попереднім. Зміна інформації в блоці неможлива без внесення змін у всі наступні блоки.

- у кожного учасника мережі є копія всіх даних (всіх цепочки блоків). Учасники взаємодіють між собою в одноранговому форматі (peer-to-peer).

– встановлений механізм консенсуса — визначена взаємодія вузлів, що забезпечує досягнення згоди щодо правильності інформації, записаної в черговий блок цепочки, і вибору, включеного в цепочку блоку з кількох можливих альтернатив.

Віталій Бутерін у статті «Про публічних і приватних блокчейнах» виділив три типи блокчейнів: публічні, приватні та консорціумні. Бутерін відзначає, що можлива велика різноманітність змішаних форм (наприклад, частні смарт-контракти на загальнодоступному блокчейні, шлюз обміну між державними та приватними блокчейнами), оптимальних для конкретної галузі або вирішуваної задачі. У деяких випадках відкритість явно краща, в інших випадках просто необхідний адміністративний контроль [2].

1.3 Види блокчейну

Існують публічні, приватні та консорціумні блокчейни. В наступних розділах будуть розглядатися сильні та слабкі сторони кожного, а також сфери їх використання [2].

1.3.1 Публічні блокчейни

Публічні блокчейни загальнодоступні. Будь-хто може читати блоки, надсилати в них інформацію та брати участь у механізмі консенсусу. Користувачі можуть залишатися анонімними. Такі блокчейни зазвичай повністю децентралізовані, тобто немає адміністраторів чи центрів довіри. Незмінність та цілісність інформації забезпечують економічні стимули та криптографічні перевірки з використанням таких механізмів, як доказ виконання роботи або доказ частки володіння [2].

Публічні блокчейни зазвичай мають суттєві обмеження в обсязі і швидкості розміщення даних у блоках.

Користувачі публічних блокчейнів багато в чому захищені від свавілля розробників: розробники спочатку відмовилися від дій без узгодження з представниками користувачів. З одного боку, це збільшує впевненість, що програма не матиме прихованих від користувачів функцій. З іншого боку, при тиску з боку державних органів розробники щиро можуть говорити, що вони не мають повноважень зробити це, навіть якби вони хотіли. При цьому зміни в роботі мережі можуть стати проблемою, оскільки не менше половини учасників мають погодитися з нововведеннями, але це не захищає від поділу блокчейну на паралельні проекти, що підтримують різні протоколи [2].

Більшість криптовалют використовують публічні блокчейни.

1.3.2 Приватні блокчейни

У приватних блокчейнах право запису інформації має лише один учасник або вузли, уповноважені цим єдиним адміністратором. Це централізовані персоніфіковані системи, оскільки є ієрархія повноважень. Збої можна швидко виправити вручну. Немає сенсу застосовувати доказ виконання роботи або доказ частки володіння - інформація без затримки потрапляє в блоки, що формуються за необхідності, і не вимагає додаткового підтвердження, що максимізує швидкість роботи мережі та мінімізує вартість транзакцій. Однак зберігається розподілений характер зберігання даних, при якому вузли містять повні копії у форматі взаємопов'язаних ланцюжків блоків. Доступ до інформації може бути загальним або мати довільні обмеження. Найчастіше йдеться про систему передачі усередині однієї компанії, що вимагає загального доступу до всієї інформації, але може передбачати загальнодоступну можливість аудиту [2].

Незважаючи на внутрішню персоналізацію, обмеження доступу до інформації можуть забезпечити у приватних блокчейнах більш високий рівень конфіденційності.

У приватному блокчейні легко реалізуються не тільки зміни правил, а й скасування транзакції, зміни інформації тощо. Це необхідно, наприклад, у

земельних кадастрах — без можливості виправити помилки подібні системи можуть стати некерованими і втратити легітимність.

Якщо вузли починають діяти зловмисно, це легко виявити та заблокувати їм доступ до мережі [2].

1.3.3 Консорціумні блокчейни

У консорціумних блокчейнах процес узгодження забезпечується кількома заздалегідь обумовленими рівноправними вузлами. Наприклад, консорціум із 15 банків домовляється вважати дійсним блок із мультипідписом не менше 10 учасників консорціуму. Швидкість появи нових блоків може бути дуже високою. При цьому учасники консорціуму доступ до інформації з блокчейну можуть зробити як загальнодоступним, так і обмежити обраним колом або запровадити інші кількісні, змістовні або тимчасові обмеження. Ці блокчейни вважатимуться «частково децентралізованими» [2].

Обмежена кількість довірених вузлів дозволяє модернізувати систему набагато простіше, ніж за публічного блокчейну. Але робота такої мережі можлива лише за умови, що основна частина вузлів працює сумлінно.

Консорціумні блокчейни найкорисніші для декількох організацій, яким потрібна єдина платформа проведення транзакцій чи обміну інформацією [2].

1.4 Сфери використання технологій блокчейну

В наступних розділах буде йти мова про використання технологій блокчейну в різних сферах життєдіяльності людини.

1.4.1 Банківський сектор, інвестиції та біржі

Про розробки та плани використання технології блокчейн заявили платіжні системи VISA, Mastercard, Unionpay та SWIFT.

Лондонський підрозділ Дойче Банку Innovation Lab розробляє систему інвестицій на основі блокчейн-технології, що прискорює, спрощує та здешевлює процес інвестування за рахунок виключення або скорочення ролі посередників, адвокатів (повірених), аудиторів та клірингових агентів.

У липні 2017 року S7 Airlines та Альфа-банк запустили в експлуатацію блокчейн-платформу автоматизації торгових операцій з агентами на базі Ethereum.

У липні 2022 року Центральний банк Індії почав використовувати блокчейн-технології для здійснення грошових переказів за кордон [3].

1.4.2 Земельний реєстр

Швеція, Україна та ОАЕ планують вести земельний реєстр за допомогою технології блокчейн.

Уряд Індії бореться із земельним шахрайством за допомогою блокчейн. Андхра-Прадеш став першим індійським штатом, де уряд зробив кроки щодо впровадження блокчейн-рішень. Для цього у місті Вішакхапатнам буде створено технологічний парк за участю блокчейн-компаній Apla, Phoenix та Oasis Grace [3].

1.4.3 Посвідчення особи

У 2014 році засновано компанію Bitnation, яка надає послуги традиційної держави, такі як посвідчення особи, нотаріат та ряд інших.

У червні 2017 року Accenture та Microsoft представили систему цифрових посвідчень особи на блокчейні.

У серпні 2017 уряд Бразилії розпочав тестування системи посвідчень особи на блокчейні.

Фінляндія ідентифікує біженців за допомогою блокчейн-технологій.

У Естонії працює блокчейн-система цифрового громадянства [3].

1.4.4 Платіжний засіб

Всесвітня продовольча програма використовує технологію блокчейн для забезпечення біженців продуктами харчування через торгові точки та мережі, що існують на місцях, замість безпосередньої роздачі продовольства або видачі біженцям готівки для покупки продуктів. Ідея належить Хуману Хададу (Houman Haddad). Для ідентифікації одержувачів продуктів використовується біометрія (сканування райдужної оболонки ока). Економія у 2018 році за рахунок застосування цієї технології лише в Йорданії склала 150 000 доларів на місяць [3].

1.4.5 Ігрова індустрія

На основі технологій блокчейн та розумних контрактів можна ігрові предмети подати у формі унікальних невзаємозамінних токенів (NFT) [3].

1.4.6 Онлайн голосування

Технологія блокчейн може використовуватись для проведення онлайн-голосувань. Такі голосування можуть приватні особи, компанії, держава. Для забезпечення анонімності і при цьому гарантії відсутності подвійного голосування може застосовуватися алгоритм простежуваного кільцевого підпису [3].

1.4.7 Будівельний бізнес

У серпні 2022 року Альфа-банк та Gaskar Group провели пілотування створеної ними цифрової системи для взаєморозрахунків між замовником та підрядниками з будівельно-монтажних робіт на базі блокчейн-платформи [3].

1.5 Переваги та недоліки блокчейну

На таблиці 1.1. приведені переваги та недоліки технології блокчейн.

Таблиця 1.1 - Переваги та недоліки

Переваги	Недоліки
Анонімний перевод цінності	Не всюди признана технологія
Цілодобовий доступ до коштів	Незворотні транзакції
Усуває потребу в посереднику.	Можливий повільний час транзакції
Повний контроль над своєю інформацією	Можлива висока ціна за транзакцію
Більший рівень прозорості	Потребується індивідуальна перевірка кожного запиту
Цілісність процесу	Потребує великої кількості енергії
Зменшує беспорядок	

1.6 Приклад програм, які використовують технологію блокчейн

1.6.1 OpenSea

OpenSea — це американський онлайн-ринок незамінних токенів (NFT) зі штаб-квартирою в Нью-Йорку. Компанію заснували Девін Фінзер і Алекс Аталла в 2017 році [4].

OpenSea є ринком, який дозволяє продавати незамінні токени безпосередньо за фіксованою ціною або через аукціон на основі стандарту Ethereum ERC-721.

Графічний інтерфейс програми зображений на рисунку 1.1.

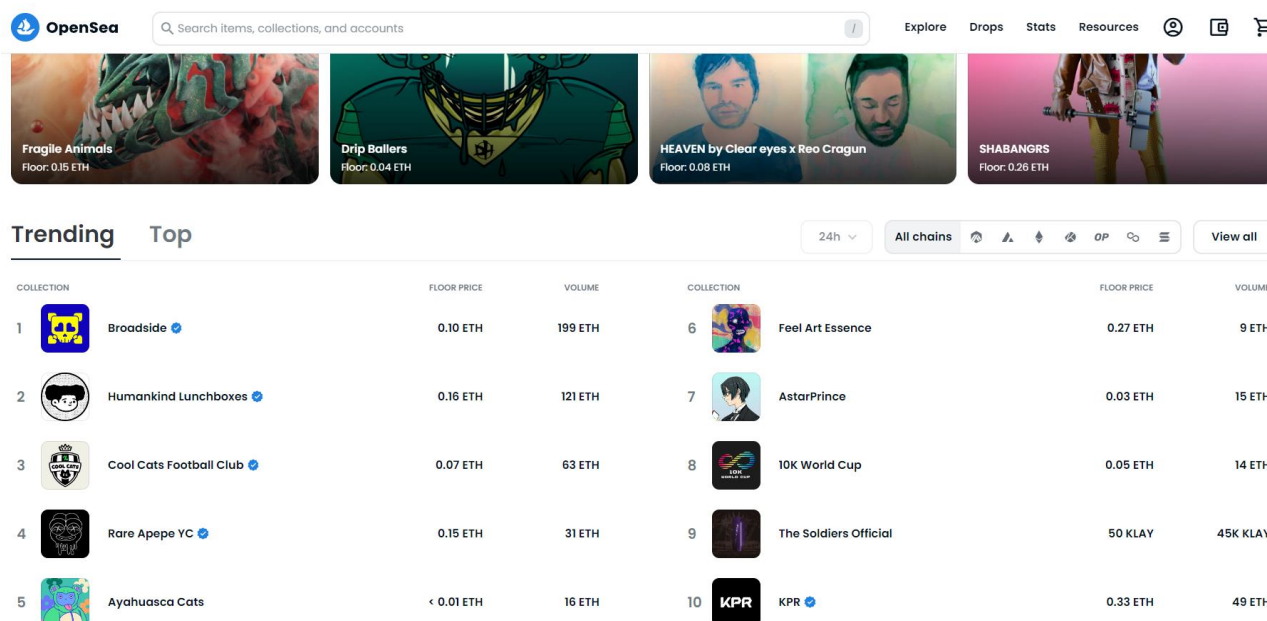


Рисунок 1.1- Графічний інтерфейс OpenSea платформи

1.6.2 Decentraland

Decentraland — це тривимірна браузерна платформа віртуального світу. Користувачі можуть купувати віртуальні земельні ділянки на платформі як NFT за допомогою криптовалюти MANA, яка використовує блокчейн Ethereum. Він був відкритий для публіки в лютому 2020 року і знаходиться під наглядом некомерційної організації Decentraland Foundation [5].

Дизайнери можуть створювати та продавати одяг та аксесуари для аватарів, які будуть використовуватися у віртуальному світі. На рисунку 1.2 зображений ігровий процес Decentraland.

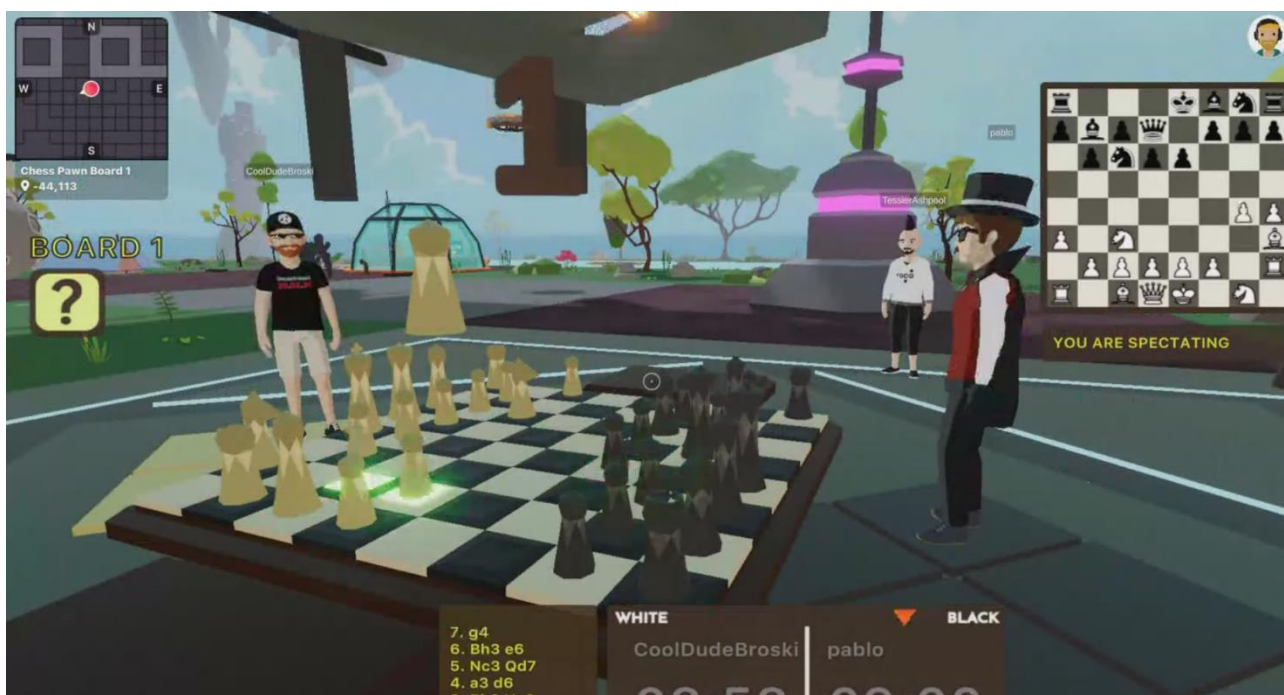


Рисунок 1.2 – Ігровий процес Decentraland

1.6.3 MetaMask

MetaMask — це програмний криптовалютний гаманець, який використовується для взаємодії з блокчейном Ethereum. Він дозволяє користувачам отримувати доступ до свого гаманця Ethereum використовуючи розширення у браузері або мобільний додаток, який потім можна використовувати для взаємодії з децентралізованими програмами [6].

MetaMask розроблено ConsenSys Software Inc, компанією, що розробляє блокчейн-програмне забезпечення, яке використовує інструменти та інфраструктуру Ethereum [6].

MetaMask дозволяє користувачам зберігати та керувати ключами облікових записів, транлювати транзакції, надсилати та отримувати криптовалюти та токени на основі Ethereum, а також безпечно підключатися до децентралізованих програм через сумісний веббраузер або браузер, вбудований у мобільну програму [6].

Вебсайти чи інші децентралізовані програми можуть підключатися, автентифікувати та/або інтегрувати інші функції смарт-контракту з гаманцем

користувача MetaMask (та будь-якими іншими подібними розширеннями для браузера гаманця блокчейну) за допомогою коду JavaScript, який дозволяє вебсайту надсилати підказки щодо дій, запити на підпис, або запити на проведення транзакцій через MetaMask як посередника [6].

Додаток містить інтегрований сервіс для обміну tokenів Ethereum шляхом об'єднання кількох децентралізованих бірж (DEX) для пошуку найкращого обмінного курсу. Ця функція під брендом MetaMask Swaps стягує комісію за обслуговування в розмірі 0,875% від суми транзакції [6].

За даними Bloomberg, станом на листопад 2021 року браузерне розширення MetaMask налічувало понад 21 мільйон активних користувачів.

На рисунку 1.3 зображений графічний інтерфейс MetaMask [6].

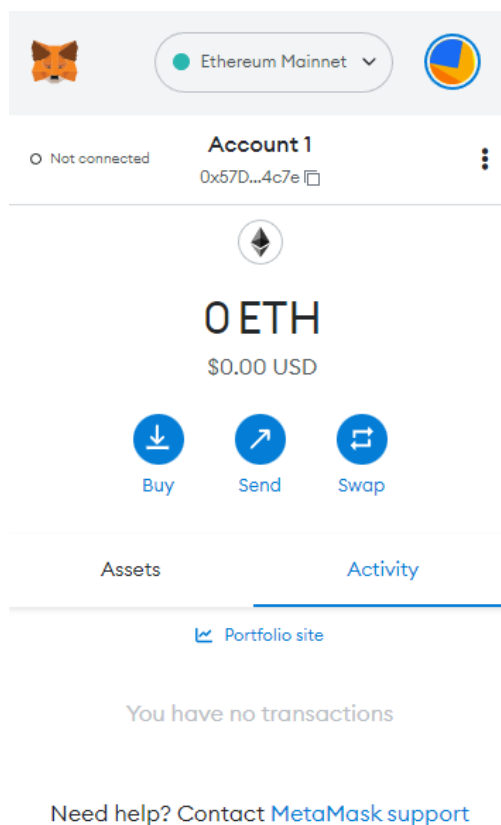


Рисунок 1.3 – Графічний інтерфейс MetaMask

1.7 Постановка задач дипломної роботи

В ході проведених досліджень була вивчена технологія блокчейн, її різновиди, сфери застосування та переги і недоліки.

Мета роботи – програмна реалізація ігрової програми play to earn з використанням технологій блокчейну.

Задачі роботи:

- аналіз вимог до системи;
- розробка архітектури системи;
- розробка ігрової механіки з використанням React, Node.js;
- розробка смарт контрактів на мові Solidity;
- завантаження в блокчейн;
- розробка інструкцій з використання програми.

1.8 Висновки до розділу 1

В даному розділі вирішено задачу аналізу предметної області. На його основі поставлено завдання до дипломної роботи.

Вирішено завдання розкрити функціонал технології блокчейну, а також історію його становлення. З'ясували, що різні типи блокчейнів: публічні, приватні та консорціумні. Розглянули приклади програм з використанням технологій блокчейну: OpenSea, Decentraland, MetaMask.

Вирішено завдання проаналізувати переваги та недоліки блокчейну. Побудовано таблицю переваг та недоліків блокчейну. Визначили, що blockchain технології мають широке використання в різних сферах життєдіяльності: банки та інвестиції, віртуальні ігрові платформи, державні послуги, віртуальні гаманці та інше.

Вирішено завдання поставити задачі до дипломної роботи на основі проведених досліджень.

В наступному розділі необхідно проаналізувати вимоги до системи, описати розробку архітектури системи, обрати фреймворк для розробки, обрати середовище програмування.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Даний розділ описує проєктування програмного забезпечення.

Розкриваються такі питання: аналіз та визначенню вимог до системи, опис розробки архітектури системи, аналіз та вибір ігрового рушія, аналіз та вибір блокчейну.

2.1 Аналіз вимог до системи

Розроблювана ігрова програма повинна бути оптимізованою для комп'ютерних браузерів, мати функціонал введення та виведення реальних коштів, бути зручною та цікавою для користувача.

Програмний продукт повинен містити наступний функціонал:

- користувальницький інтерфейс головного меню;
- підключення віртуального гаманця MetaMask;
- проведення транзакцій перед початком і кінцем гри;
- ігрова механіка стрільби по цілям;
- налаштування гри;

2.2 Розробка архітектури системи

Для реалізації проекту обраний шаблонний підхід MVC, який дозволяє розділити систему на 3 розділи:

- model;
- view;
- controller.

Модель надає дані і методи роботи з ними: запити до бази даних, перевірка на коректність. Модель не залежить від представлення (не знає як дані візуалізувати) і контролера (не має точок взаємодії з користувачем), просто

надаючи доступ до даних і управління ними. Модель будується таким чином, щоб відповідати на запити, змінюючи свій стан, при цьому може бути вбудовано повідомлення «спостерігачів». Модель, за рахунок незалежності від візуального представлення, може мати кілька різних представлень для однієї «моделі».

Представлення відповідає за отримання необхідних даних з моделі і відправляє їх користувачеві. Представлення не оброблює введені дані користувачем.

Контролер забезпечує «зв'язок» між користувачем і системою. Контролює і направляє дані від користувача до системи і навпаки. Використовує модель і представлення для реалізації необхідної дії [7].

Схема, яка демонструє принцип роботи MVC, зображена на рисунку 2.1.

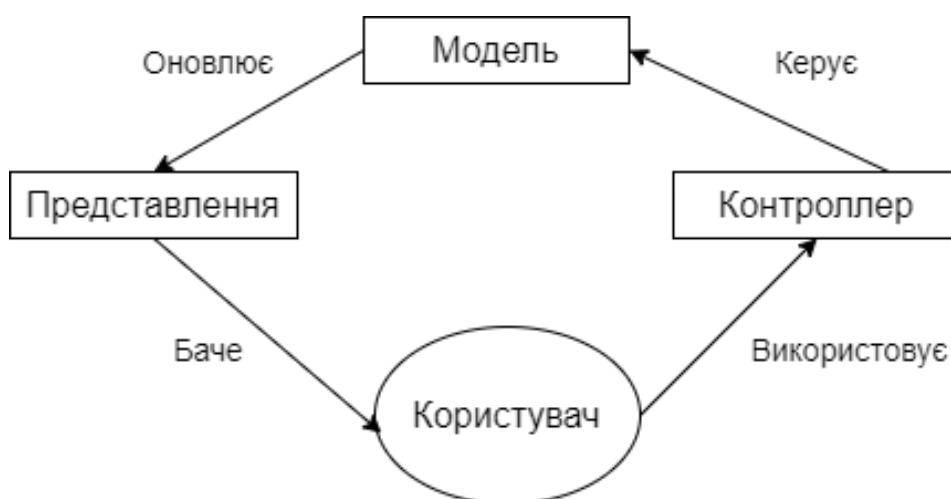


Рисунок 2.1 – Схема MVC

Також, для підтримки принципу єдиної відповідальності і можливості розбити систему на модулі, використаний архітектурний підхід Dependency Injection.

Dependency Injection – процес надання зовнішньої залежності програмного компоненту. Є специфічною формою «інверсії управління» (англ. Inversion of control, IoC), коли вона застосовується до управління залежностями

[8]. У повній відповідності з принципом єдиною відповідальності, об'єкт віддає піклування про побудову необхідних йому залежностей зовнішньому, спеціально призначеному для цього спільного механізму. В данному випадку, класу – GameRoot, який ініціалізує всі компоненти необхідними залежностями і даними.

Використовуючи впровадження залежності, об'єкт просто надає властивість, яка в змозі зберігати посилання на потрібний тип сервісу, і коли об'єкт створюється, посилання на реалізацію потрібного типу сервісу автоматично вставляється в це властивість (поле).

На рисунку 2.2 зазначена різниця між традиційним підходом і Dependency Injection.

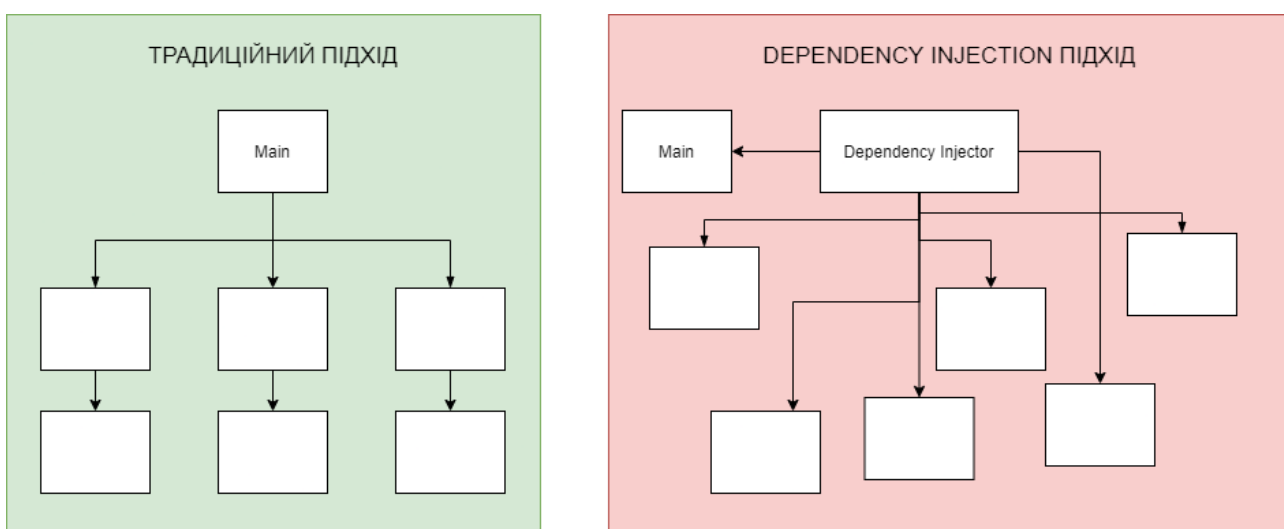


Рисунок 2.2 – Різниця між традиційним підходом і Dependency Injection.

Діаграма класів (рисунок 2.3) була розроблена за допомогою мови UML [9]. Відношення між класами відображають принцип роботи Dependency Injection.

2.3 Вибір фреймворку для розробки

У даному розділі проводиться аналіз існуючих фреймворків та ігрових рушіїв з метою вибору необхідних технологій для повноцінної розробки вебгри з використанням blockchain технологій.

Ігровий рушій (англ. Game engine) – це основний набір програмних компонентів і візуальних інструментів, який дозволяє створювати і запускати інтерактивні додатки з графічним забезпеченням, яке обробляється в реальному часі [10].

Фреймворк (англ. Framework) — це каркас програмної системи (чи підсистеми). Може включати: допоміжні програми, бібліотеки коду, мову сценаріїв та інше програмне забезпечення, що полегшує розробку та об'єднання різних компонентів великого програмного проекту. Зазвичай об'єднання відбувається рахунок використання єдиного API [11].

Приклади: вебфреймворки, як PHP-фреймворки Zend Framework, Laravel або Symfony або Django, написаний на Python.

2.3.1 Phaser

Phaser — це ігровий рушій 2D-ігор, який використовується для створення HTML5 додатків для комп'ютерів і мобільних пристроїв. Це безкоштовне програмне забезпечення, розроблене Photon Storm [12].

Phaser використовує рендери Canvas і WebGL і може автоматично перемикатися між ними залежно від підтримки браузера. Це забезпечує швидке графічне відтворення на комп'ютері та мобільному пристрої. Він використовує бібліотеку Pixi.js для візуалізації [12].

Ігри можна публікувати на iOS, Android і власних настільних програмах за допомогою сторонніх інструментів, таких як Apache Cordova та phonegap [12].

Річард Дейві оголосив про перший випуск Phaser у своєму блозі в квітні 2013 року. Версія 1.0 була випущена у вересні, включаючи бібліотеку Pixi.js для візуалізації [12].

Останньою офіційною версією Phaser 2 була 2.6.2, але щоб удосконалити стабільну версію під час роботи над Phaser 3, було створено новий репозиторій: Phaser CE (Community Edition). Таким чином, Phaser CE наразі є рекомендованою стабільною платформою для розробки [12].

Phaser 3.0.0 було випущено 13 лютого 2018 року, і розробка триває на GitHub. Більшість елементів і функцій інфраструктури було перебудовано з нуля з використанням повністю модульної структури та підходу, орієнтованого на дані. Phaser 3 містить абсолютно новий рендерер WebGL, розроблений для сучасних 2D-ігор. З тих пір більшість документації та прикладів для користувачів було завершено, а більшість функцій реалізовано [12].

Наразі розробляється Phaser 4, аносований 19 серпня 2019 року, який є спробою переписати Phaser 3 на TypeScript. Це не переписування API, а натомість буде зосереджено на перенесенні скриптів, які зараз є у Phaser 3, на TypeScript [12].

Phaser можна запускати в будь-якому веббраузері, який підтримує елемент canvas. Ігри, створені за допомогою Phaser, розробляються на JavaScript або TypeScript. Вебсервер потрібен для завантаження таких ресурсів, як зображення, аудіо та інші ігрові файли, оскільки браузери вимагають, щоб вебсторінки отримували доступ до файлів лише з того самого походження.

Візуалізація. Phaser можна візуалізувати як у WebGL, так і в елементі Canvas із можливістю використання WebGL, якщо браузер підтримує це, або, якщо пристрій не підтримує, він повернеться до Canvas [12].

Фізика. Phaser поставляється з трьома фізичними системами: Arcade Physics, Ninja Physics і P2.JS. Arcade Physics призначений лише для високошвидкісних зіткнень AABB. Ninja Physics дозволяє створювати складні тайли та схили, які підходять для пейзажу рівнів, а P2.JS — це повна система фізики, яка підтримує обмеження, пружини та багатокутники [12].

Анімація та аудіо. Анімацію можна виконати у Phaser, завантаживши спрайт-таблицю, атлас текстур і створивши послідовність анімації.

Вебаудіо та аудіо HTML5 можна використовувати для відтворення звуку.

Переваги:

- безкоштовна версія;
- наявність потужного і інтуїтивно зрозумілого редактора;
- візуалізація елементів редактора в реальному часі;
- кросплатформенна підтримка;
- модульна система компонентів.

Недоліком є обмеження візуального редактора при роботі з багатокомпонентними схемами;

На рисунку 2.4. зображено графічний інтерфейс Unity3d.

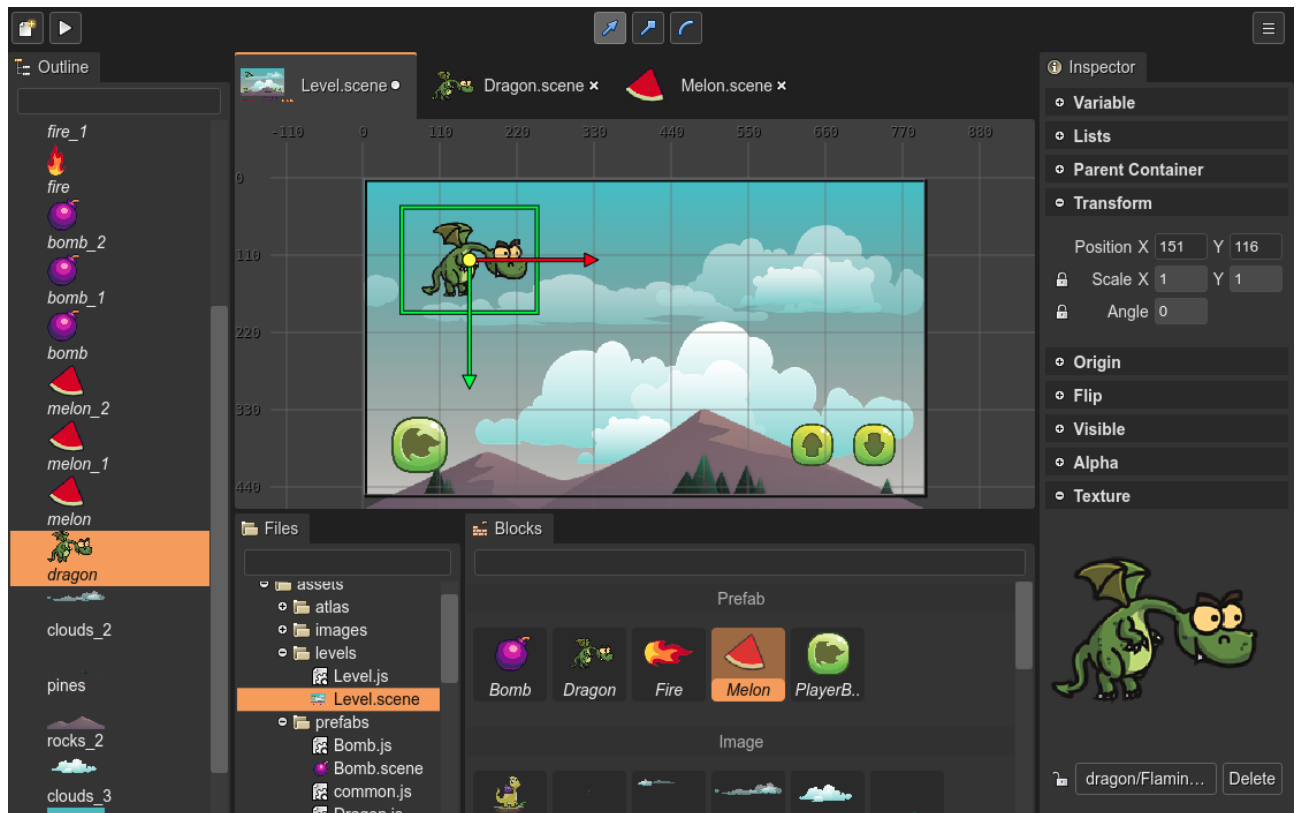


Рисунок 2.3 – Графічний інтерфейс Phaser

2.3.2 Node.js

Node або Node.js - програмна платформа, заснована на двигуні V8 (компілюючому JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованої мови на мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями вводу-виводу через свій API, написаний на C++, підключати інші зовнішні бібліотеки, написані різними мовами, забезпечуючи виклики до них із JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль вебсервера, але є можливість розробляти на Node.js та десктопні віконні програми (за допомогою NW.js, AppJS або Electron для Linux, Windows та macOS) і навіть програмувати мікроконтролери (наприклад, tessell, low.js та espruino). В основі Node.js лежить подієво-орієнтоване та асинхронне (або реактивне) програмування з неблокуючим введенням/виводом [13].

Переваги:

- легкість і швидкість написання;
- відносна простота (у порівнянні з java);
- node package manager (величезна кількість бібліотек які можуть бути встановлені в один рядок);
- кожна бібліотека потрапляє в дерево залежностей і все це робиться легко;
- постійний розвиток (зараз активно розвивається TypeScript (який привносить в JS типізацію, декоратори і використовується наприклад для Angular);
- відмінна якість графіки;
- універсальність коду для різних платформ;

Недоліки:

- треба постійно стежити за оновленнями, деякі речі виходять недостатньо протестованими;

– сильна залежність від NPM, якщо видалити, то перестань працювати багато інших програм;

2.3.3 Hardhat

Hardhat — це середовище, яке розробники використовують для тестування, компіляції, розгортання та налагодження dApps на основі блокчейну Ethereum. Таким чином, він допомагає програмістам і розробникам керувати багатьма завданнями, пов'язаними з розробкою dApps (децентралізовані додатки) і смарт-контрактів. Окрім надання розробникам належних інструментів для керування цим процесом, Hardhat також допомагає автоматизувати деякі з цих кроків і надає розробникам нові, корисні функції [14].

Hardhat постачається з попередньо побудованою локальною мережею Ethereum. Ця мережа зосереджена на налагодженні Solidity та трасування стека, повідомлення про збій транзакцій dApps тощо. Це надає розробникам необхідні інструменти для розуміння того, де програма виходить з ладу, і надає їм відповідь для їх вирішення.

Середовище характеризується плагінами, з яких походить багато функцій. Це означає, що розробники можуть вибрати, які саме плагіни вони хотіли б включити в процес розробки. Однак він має вбудовані параметри за замовчуванням, але їх усі можна змінити. Це означає, що програмному забезпеченню Hardhat байдуже, які інструменти використовують розробники.

2.3.4 Truffle

Truffle — це середовище розробки світового класу, платформа для тестування та конвеєр активів для блокчейнів, які використовують віртуальну машину Ethereum (EVM), покликане полегшити життя розробника.

Truffle вважається найпопулярнішим інструментом для розробки блокчейн-додатків із понад 1,5 мільйонами завантажень за весь час. Truffle

підтримує розробників протягом повного життєвого циклу їхніх проектів, незалежно від того, чи хочуть вони створювати на Ethereum, Hyperledger, Quorum або одній із постійно зростаючого списку інших підтримуваних платформ. У поєднанні з Ganache, персональним блокчейном, і Drizzle, інтерфейсним набором для розробки dApp, повний набір інструментів Truffle обіцяє стати повноцінною платформою для розробки dApp [15].

Особливості:

- вбудована компіляція смарт-контрактів, зв'язування, розгортання та керування двійковими файлами;
- автоматизоване тестування контракту для швидкої розробки;
- розширювана структура розгортання та міграції з можливістю сценаріїв;
- керування мережею для розгортання в будь-якій кількості публічних і приватних мереж;
- управління пакетами за допомогою EthPM і NPM за стандартом ERC190;
- інтерактивна консоль для прямого контрактного спілкування;
- настроюваний конвеєр збірки з підтримкою тісної інтеграції;
- зовнішній засіб виконання сценаріїв, який виконує сценарії в середовищі Truffle;

2.3.5 Порівняння Truffle і Hardhat

На таблиці 2.1 зображено порівняння Truffle і Hardhat на основі якого буде обраний той чи інший фреймворк для розробки з використанням blockchain технологій.

Таблиця 2.1 - Порівняння Truffle і Hardhat

Критерій	Hardhat	Truffle
Документація	Проста и зрозуміла документація з туторіалами	Комплексна документація і онлайн ресурси
Спільнота	Нове ком'юніті з розробниками і дискорд сервером	Велике ком'юніті з підтримкою
Бібліотеки та плагіни	Підтримка багатьох плагінів та бібліотек	Truffle Boxes та front-end бібліотеки в Drizzle
Тестування	Гнучке тестування з підтримкою смартконтрактів	Відсутність підтримки прямого контакту зі смартконтрактами під час тестування
Розгортання контрактів	Розробники можуть писати власні скрити та використовувати плагіни для додавання нових можливостей	Фіксований шаблон обмежує розгортання контрактів
Обробка помилок	Легкий доступ до управління помилками і інструменти тестування з трасуванням стеку	Доступ лише до помилок
Легкість розгалуження	Легке розвітвлення без додаткових інсталяцій	Потрібно встановити Ganache
Інтеграція TypeScript	Легкий процес інтеграції TypeScript	Тяжкий процес інтеграції TypeScript
Взаємодія зі смарт контрактами	Ethers.js бібліотека за замовчуванням для взаємодії зі смарт контрактами	Web3.js бібліотека за замовчуванням для взаємодії зі смарт контрактами
Компанії, які використовують фреймворк	Aave, Optimism, OpenZeppelin	JP Morgan Chase, BitGo, 1inch

Проведено аналіз, який показав, що найкращим рішенням буде використання фреймворка Hardhat для розробки blockchain функціоналу, оскільки він досить легкий у вивченні, наявності безкоштовна версія, велике ком'юніті з розробниками, якісна документація, а також багатий функціонал дебагінгу. Truffle уступає в данному виборі через громіздкість (велика кількість непотрібного функціонала для виконання мети проекту), не зручна документація, потрібно встановлювати Ganache для клонування проекту, не

велика кількість інструментів для дебагу (доступ лише до помилок). Отже Hardhat є найкращим вибором для реалізації мети проєкту.

Для створення механік ігрового процесу буде використовуватися Phaser та Node.js, оскільки інших альтернативних варіантів немає.

2.4 Обґрунтування вибору Hardhat в якості фреймворку для розробки blockchain функціоналу

Середовище розробки смарт-контрактів Hardhat пропонує розробникам відповідні інструменти для керування процесом розробки. Крім того, Hardhat також забезпечує ефективність, запроваджуючи автоматизацію на окремих етапах разом із новими та продуктивними функціями. Hardhat містить попередньо побудовану локальну мережу Ethereum, адаптовану для основної мети розвитку. Рівняння Hardhat Solidity чітко видно з фокусу фреймворку на налагодженні Solidity. У той же час Hardhat також має функцію трасування стека та повідомлень про збій dApp. У результаті розробники можуть мати під рукою необхідні інструменти для визначення екземпляра та причини збою програм. Він також може запропонувати необхідну відповідь для вирішення проблем, що лежать в основі збою програми.

Інша важлива особливість Hardhat відноситься до плагінів, які пропонують широкий спектр утиліт. Плагіни гарантують, що розробники можуть вибрати ті, які вони хочуть додати до процесу розробки. Навіть якщо ви знайдете вбудовані плагіни за замовчуванням, ви можете змінити їх відповідно до своїх потреб. Hardhat в основному не піклується про інструменти чи функції, які розробники хочуть мати у своїх програмах.

Найважливіша вимога для операцій розгортання Hardhat стосується Hardhat Runner. Це інтерфейс командного рядка або команда CLI, яка використовується для взаємодії з Hardhat. Hardhat Runner більше схожий на програму для запуску завдань, яка пропонує користувачам безліч альтернатив і

опцій. Розробники мають зауважити, що основна частина дизайну програми для виконання завдань фокусується на завданнях і плагінах. Конструкція засобу виконання завдань також передбачає, що розробники запускають завдання щоразу, коли вони запускають Hardhat із CLI. Приклад команди для запуску Hardhat з CLI буде таким: *npx hardhat compileCory*. Коли запускається команда в блокчейні Hardhat, вона запускає вбудований процес компіляції. Згодом одне завдання може викликати інше завдання, що спрощує розробку та визначення складних робочих процесів. Крім того, розробники також можуть використовувати Hardhat для перевизначення вже існуючих завдань, таким чином дозволяючи опції для розширення або налаштування робочих процесів.

Наступний важливий момент стосується мережі Hardhat. Згідно з визначенням Hardhat, фреймворк для розробки смарт-контрактів містить попередньо побудовану локальну мережу Ethereum. Локальна мережа Ethereum допомагає підтримувати тестування на Hardhat. Це невід’ємна вимога для розробників щодо роботи над тестуванням, запуском, налагодженням і розгортанням децентралізованого коду програми та смарт-контрактів.

Робота мережі Hardhat є ще одним багатообіцяючим моментом у розумінні структури розробки Hardhat Solidity. Одна з перших подробиць про функціонування мережі Hardhat вказує на майнінг блоку для кожної транзакції без будь-яких затримок. Крім того, мережа також покладається на підтримку впровадження EVM, подібного до того, що використовується на таких платформах, як Ganache, Remix і Ethereum Studio. Крім того, мережа також пропонує підтримку кількох хардфорків, таких як Петербург, Візантія та Константинополь.

Мережа Hardhat має можливість визначити випадок збою виклику чи транзакції. Крім того, він також може точно визначити причину збою транзакції або викликів. У результаті розробники можуть використовувати структуру розробки блокчейну Hardhat для простішого та ефективнішого налагодження. У разі невдачі транзакції без чітко визначеної причини мережа надає ясність, пропонуючи детальне пояснення помилки. Автоматичні повідомлення про

помилки відіграють важливу роль у процесі налагодження та можуть стикатися з різними ситуаціями. Нижче наведено деякі поширені випадки, коли мережа відображатиме автоматичні повідомлення про помилки:

- коли користувачі викликають функцію без вказівки правильних параметрів.
- використання неконтрактного облікового запису для виклику зовнішньої функції.
- некоректний виклик попередньо складеного договору.
- надсилання ефіру (ETH) до контракту без функції резервного повернення або отримання.
- розгортання контрактів у мережі Hardhat, які перевищують ліміт розміру бітового коду.
- надсилання недостатньої кількості ефіру або ETH.

Ефективність Hardhat у визначенні автоматичних повідомлень про помилки допомагає розробникам забезпечити значні цінні покращення. Найголовніше те, що розробники можуть отримати перевагу кращої швидкості процесу налагодження перед розгортанням коду

2.5 Вибір середовища програмування

2.5.1 Visual Studio Code

Visual Studio Code – це редактор вихідного коду. Він має багатомовний інтерфейс користувача та підтримує ряд мов програмування, підсвічування синтаксису, IntelliSense, рефакторинг, налагодження, навігацію за кодом, підтримку Git та інші можливості. Багато можливостей Visual Studio Code недоступні через графічний інтерфейс, часто вони використовуються через палітру команд або JSON-файли (наприклад, налаштування користувача). Палітра команд є подобою командного рядка, що викликається поєднанням клавіш [17].

VS Code також дозволяє замінювати кодову сторінку під час збереження документа, символи перекладу рядка та мову програмування поточного документа [17].

З 2018 з'явилося розширення Python для Visual Studio Code з відкритим вихідним кодом. Воно надає розробникам широкі можливості для редагування, налагодження та тестування коду [17].

Також VS Code підтримує редагування та виконання файлів типу Блокнот Jupyter (Jupyter Notebook (англ.)) безпосередньо з коробки без встановлення зовнішнього модуля в режимі візуального редагування і в режимі редагування вихідного коду [17].

На березень 2019 року за допомогою вбудованого в продукт інтерфейсу користувача можна завантажити і встановити кілька тисяч розширень тільки в категорії «programming languages» (мови програмування) [17].

Також розширення дозволяють отримати більш зручний доступ до програм, таких як Docker, Git та інші. У розширеннях можна знайти літери коду, теми для редактора та підтримку синтаксису окремих мов [17].

На таблиці 2.2 зображено, які мови програмування підтримує Visual Studio Code [17].

Таблиця 2.2 – Підтримувані мови програмування Visual Studio Code

Можливості	Мови програмування
Підсвітка синтаксису	Batch file C C# C++ CSS Clojure CoffeeScript Diff Dockerfile F# Git commit i rebase Go Groovy HLSL HTML Handlebars INI JSON Java JavaScript JavaScript ReactLESS (мова стилів) Lua Makefile Markdown Objective-C Objective-C++ PHP Perl Perl 6 PowerShell

Продовження таблиці 2.2

Можливості	Мови програмування
IntelliSense	CSS HTML JavaScript JSON Less Sass TypeScript
Рефакторинг	C# TypeScript
Отладка	• JavaScript и TypeScript для проєктів Node.js • C# і F# для проєктів .NET и Mono • C і C++ • Python (при встановленном плагіні Python) • PHP с XDebug (при установленном плагине PHP Debug) • Java (при встановленном наборі плагінів Java Extension Pack або окремо встановленном Debugger for Java) • Go (при встановленному плагіні Go)

2.5.2 JetBrains Rider

Rider - кроссплатформенне інтегроване середовище розробки програмного забезпечення для платформи .NET, що розробляється компанією JetBrains. Підтримуються мови програмування C#, VB.NET та F# [18].

Проект анонсовано у січні 2015 року. У його основі лежить інший продукт JetBrains - ReSharper. Середовище підтримує платформи .NET Framework, .NET та Mono. Працює на операційних системах Windows, MacOS, Linux [18].

Реліз відбувся 3 серпня 2017

Таблиця 2.3 – Порівняння IDE

Критерій	Rider	Visual Studio Code
Можливість роботи на різних платформах	+	+
Безкоштовна ліцензія	-	+
Якісна документація	+	+
Підтримка Node.js	-	+
Підтримка javascript	-	+
Система контролю версій	+	+
Приємний дизайн	+	+
Оптимізованість	+	+
Декомпіляція зовнішніх бібліотек	+	-

2.6 Висновки до розділу 2

Для комфортної роботи та швидкої розробки проекту, Visual Studio Code є найбільш оптимальною середою розробки, оскільки він має всі необхідні інструменти для зручного написання коду та чудово взаємодіє з Node.js, Hardhat, підтримує синтаксис javascript, а також надає більше інструментів для рефакторингу коду.

В даному розділі вирішено задачу з проектування програмного забезпечення. Завдяки проектуванню, визначено які архітектурні підходи та фреймворки використовувати в розробці.

Виконано завдання визначити функціональні вимоги до програмного продукту. Перечислені вимоги є необхідними для того, щоб продукт мав привабливість, користувачеві було приємно і зручно грати, а також щоб була можливість проводити транзакції на блокчейні Ethereum.

Виконано завдання проаналізувати архітектурні підходи, які допоможуть найоптимальнішим способом реалізувати зазначені функціональні вимоги. Розроблено схему (рис.2.1), яка демонструє суть патерну MVC (Model View Controller), який розділяє програму на 2 великі частини: дані та представлення.

Також розроблена схема, для більш наявного розуміння підходу Dependency Injection (рис 2.2), метою якого є – зробити явними всі залежності між програмними класами та об’єктами.

Виконано завдання розробити діаграму класів (рис.2.3), яка відображає використання двох архітектурних підходів. Згідно неї, були розроблені програмні класи.

Виконано завдання проаналізувати ігрові рушії та фреймворки для розробки смартконтрактів: Phaser, Truffle, Hardhat. Продемонстрована таблиця порівняння Truffle та Hardhat (рис. 2.1), яка дозволяє наявно побачити переваги та недоліки кожного з метою вибору оптимального варіанту. Обрано фреймворк Hardhat для написання смарт контрактів та розгортання програм у блокчейн по таким причинам: легкий у вивченні, в наявності безкоштовна версія, розширені інструменти дебагу, якісна докуменація, а також інтегрована віртуальна машина яка надає можливість тестувати програми без завантаження в блокчейн Ethereum.

Виконано завдання проаналізувати існуючі IDE (Integrated Development Environment). Побудована таблиця порівняння (табл. 2.2), яка демонструє переваги та недоліки кожного. Після чого, як середа програмування, на основі аналізу Visual Studio Code та JetBrains Rider, обраний Visual Studio Code, по таким причинам: підтримка синтаксису javascript, інтеграція Node.js, оптимізованість та безкоштовна ліцензія.

В наступному розділі необхідно описати конструювання програмного забезпечення: з чого складається ігровий процес, яка структура проекту, розробка смарт контрактів, інтеграція в блокчейн Ethereum.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В даному розділі описується програмна реалізація ігрової програми з використанням блокчейну Ethereum. Розділ присвячений наступним питанням: розробка ігрового процесу, опис структури проекту, опис розробки смарт контрактів, тестування на віртуальній машині Hardhat Ethereum.

3.1 Розробка ігрового процесу

Геймплей (gameplay) - ігровий процес у комп'ютерній грі [19].

Геймплей містить невід'ємний компонент комп'ютерних ігор, відсутній в інших видах мистецтва, і це компонент — інтерактивність. Геймплей гри є ступенем та природою інтерактивності, передбаченою грою. Тобто, як гравці можуть впливати на ігровий світ, і як ігровий світ реагує на дії, вчинені гравцями [19].

Поняття геймплея не включає графічне оформлення гри, використовуваний движок, сеттинг, в якому відбувається дія, або сюжетну складову. Подібні естетичні та змістовні елементи не є унікальними для комп'ютерних ігор і зустрічаються в інших засобах аудіовізуальної інформації (книги, кіно, телебачення). Геймплей же це те, що робить комп'ютерні ігри унікальним видом мистецтва [19].

3.2 Структура проекту

Проект складається з багатьох частин: скрипти (які і реалізують взаємодію між об'єктами на ігровому рівні), ігрова механіка, графічне відображення через спрайти, графічний інтерфейс (через який користувач взаємодіє з грою). Завдяки використанню архітектурного патерну MVC, вдалося

створити систему, в якій бізнес модель, тобто логіка програми, не залежить від візуальних компонентів. Це говорить про те, що можна підібрати будь-які графічні моделі та змінювати користувацький інтерфейс без редагування логіки програми. Завдяки використанню смартконтрактів на блокчейні Ethereum вдалося досягти підключення крипто гаманця та проведення транзакцій у випадку виграшу або програшу в процесі гри, завдяки чому вдалося реалізувати концепцію play to earn (грай, щоб заробляти).

Далі в таблиці 3.1 приведено опис скриптів, завдяки яким і запрограмований ігровий процес «Дія-Реакція-Відгук».

Таблиця 3.1 – Опис файлів

Назва	Опис
ground.js	Реалізує логіку анімації поверхності на якій знаходиться головний герой, за рахунок чого створюється імітації бігу головного героя.

Продовження таблиці 3.1

Назва	Опис
player.cs	Реалізує логіку переміщення головного героя, прискорення, оновлення та включення/виключення графічних ефектів. Містить функції зчитування доторкань, свайпів та інших маніпуляцій, які походять від користувача.
obstacle.cs	Реалізує появлення, видалення та оновлення стану у разі перетину з головним героєм.
nft.js	Реалізує появлення nft (non fungible token) у вигляді ігрових об'єктів, які герой може збирати і які потім конвертуються у криптовалюту, за рахунок чого реалізується концепція play to earn
RunToken.sol	Реалізує токен з протоколом ERC20
RunnerCollection.sol	Змінює ідентифікатор зібраних користувачем токенів та представляю собою колекцію ERC1155, які буде розгортатися на платформі OpenSea.
deployRunToken.js	Реалізує мінт токenu (інтеграції в блокчейн Ethereum), який потім відображається у віртуальному гаманці користувача як окрема криптовалюта
deployNFTCollection.js	Реалізує мінт колекції зібраних токенів на платформі OpenSea, які потім можна переглянути як колекцію зображень

На рисунку 3.1 зображена структура скриптів, які смартконтракти, які будуть працювати в блокчені Ethereum.

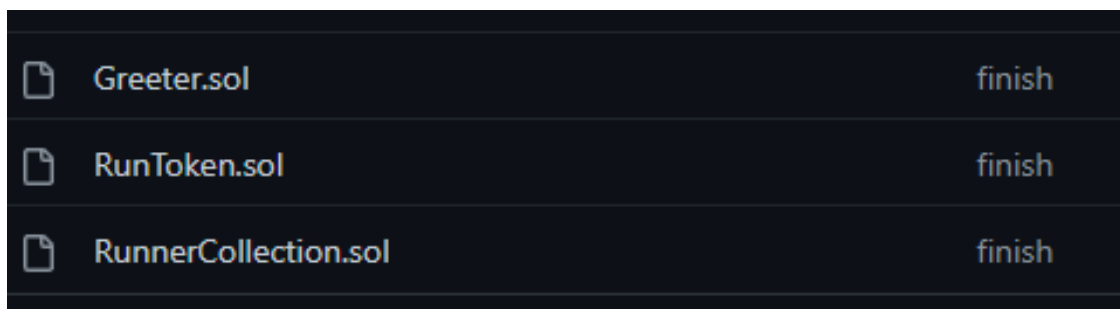


Рисунок 3.1 – Ієрархія об'єктів проекту

На рисунку 3.2 зображена структура скриптів, які реалізують ігрову механіку.

📁	imgs	finish
📄	.gitignore	finish
📄	blockchain.js	finish
📄	ground.js	finish
📄	index.html	finish
📄	nft.js	finish
📄	obstacle.js	finish
📄	player.js	finish
📄	script.js	finish
📄	styles.css	finish
📄	updateCustomProperty.js	finish

Рисунок 3.2 – Структура файлів скриптів ігрової механіки

3.3 Опис процесі мінтинга

Мінт – це процес отримання цифрового активу, який є перетвореним цифровим файлом, що зберігається на блокчейні.

Процес допомагає зробити NFT товаром, який можна купувати, продавати та переводити між гаманцями офіційно. Мінт – це народження NFT на блокчейні.

На рисунку 3.3 зображено, що рахунок віртуального гаманця MetaMask користувача збільшився в результаті збирання токенів під час проходження гри.

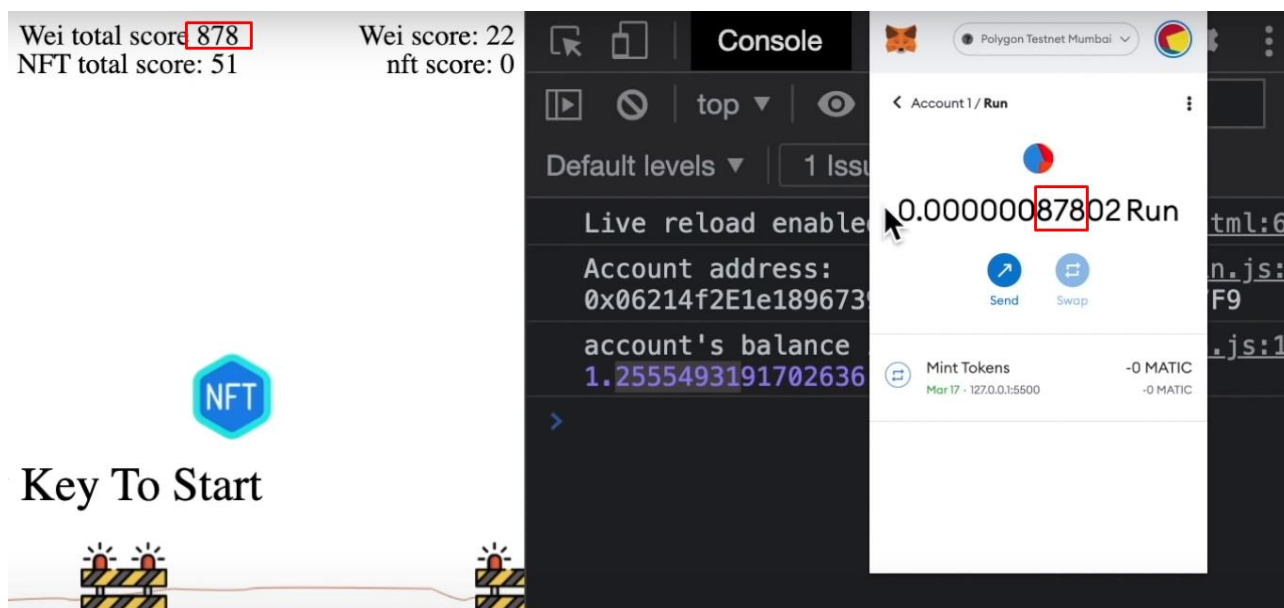


Рисунок 3.3 – Поповнення рахунку гаманця після проходження гри

3.4 Опис програмної реалізації графічного інтерфейсу

Для реалізації зручної системи керування ігровими екранами прийняте рішення створити єдиний контроллер, який буде мати посилання на всі ігрові екрани, задіяні в додатку, а також функції перевірок, які при виконанні заданих умов викликають функції інших класів та відображають необхідну інформацію.

Панелі, екрани, текст та спливаючі вікна реалізовані через canvas HTML5. Для керування графічними елементами був створений скрипт main.js, де і реалізовані умови та логіка відображення графічного інтерфейсу.

В таблиці 3.2 приведений опис функцій main.js.

Таблиця 3.2 – Опис функцій main.js

Метод	Опис
Update()	Оновлення графічних елементів кожен кадр
checkLose()	Перевірка, чи зіткнувся гравець з перешкодою
void HideAll()	Виключення всіх вікон одразу.
checkIfWeGotNft()	Перевірка, чи зібрали ми NFT

Продовження Таблиці 3.2

Метод	Опис
updateScore()	Оновлення рахунку
handleStart()	Створення графічних елементів гравця, рівня, перешкод, nft
handleLose()	Оновлення графічних елементів після програшу гравця

3.5 Висновки до розділу 3

Вирішена задача конструювання програмного забезпечення. Завдяки цьому стало зрозумілим що саме необхідно розробити, а також, на які етапи поділити розробку.

Вирішено завдання дослідження ігрового процесу. Визначено, які бувають етапи. Згідно цих етапів спроектований ігровий процес розроблюваного додатку.

Вирішено завдання розробки структури проекту. Структура складається з скриптів (які і реалізують взаємодію між об'єктами на ігровому рівні), зі спрайтів, які використовуються для відображення графічного інтерфейсу та візуалізації ігрової механіки, зі смартконтрактів, які представляють собою токени та за скриптів розгортання даних смартконтрактів в блокчейні Ethereum. Це допомогло визначитись з тим, які елементи програми необхідно розробити.

Вирішено завдання з програмної реалізації графічного інтерфейсу. Розроблено таблицю (рис. 3.2), яка описує кожну функцію відображення графічного інтерфейсу. Завдяки обраному підходу (створення єдиного контролера, керуючого усіма типами ігрових екранів), вдалося реалізувати зручний перехід між ігровими екранами, включення або виключення яких можливо з будь-якого класу-контролеру.

Розроблено смартконтракт з використанням протоколу ERC20, які представляють собою токени Run. Розроблено смартконтракт RunnerCollection,

який представляє собою колекцію nft. Розроблено скрипт мінтингу токenu Run в блокчейні Ethereum, що надало можливість зробити токен цифровим активом, який можна передавати, купувати та продавати, за рахунок чого і досягається мета створення play to earn гри. Розроблено скрипт розгортання зібраних користувачем токені на платформі OpenSea, які відображаються як спрайти і які можуть бути придбані іншими користувачами, що також підтверджує реалізацію мети створення play to earn гри.

В наступному розділі необхідно визначити системні вимоги, описати функціонал системи, інструкції з запуску системи, роботу системи.

4 ІНСТРУКЦІЇ З ВИКОРИСТАННЯ ПРОГРАМНОГО ПРОДУКТУ

В даному розділі описуються інструкції з використання програмного продукту. Розділ присвячений наступним питанням: системні вимоги, функціонал системи, інструкції з запуску системи, опис роботи системи.

4.1 Системні вимоги

Для роботи програми необхідний будь який браузер на ПК, підключений MetaMask або будь-який інший крипто гаманець.

4.2 Функціонал системи

Ігрова програма призначена для розваги користувачів та заробітку справжніх грошей через криптовалюту.

Програмний продукт надає наступний функціонал:

- а) запуск додатка;
- б) відображення головного меню з надписом «Press Any Key To Start», при натисканні будь-якої клавіші запускається ігровий процес
- в) безкінечний ігровий раунд, в якому користувач керує головним героєм, метою чого є – проходити рівень оминаючи перешкоди та збираючи nft; під час ігрового раунда відображаються такі складові ігрового процесу:
 - 1) поточний рахунок, який збільшується протягом проходження рівня;
 - 2) поточний рахунок зібраних nft;
 - 3) загальний рахунок;
 - 4) загальна кількість зібраних nft;
 - 5) кнопка «Connect MetaMask», при натиску на яку відбувається підключення віртуального гаманця MetaMask;

- 6) кнопка «Claim nft», при натиску на яку відбувається мінтинг зібраних токенів в блокчейн Ethereum;
- г) мінтинг зібраних nft, який відбувається через натискання на кнопки «Connect MetaMask» та «ClaimTokens»:
 - 1) поповнення рахунку криптогеманця токенами Run, які можна передавати, продавати, купувати та обмінювати на USDT;
 - 2) відображення колекції токенів на платформі OpenSea;

4.3 Інструкції з запуску системи

Для запуску програми, необхідно розгорнути власний вебсервер через використання Node.js та запустити вебсторінку index.html.

4.4 Опис роботи системи

Для використання ігрової програми, необхідно запустити вебсторінку. На рисунку 4.1 зображено головне меню перед початком гри.

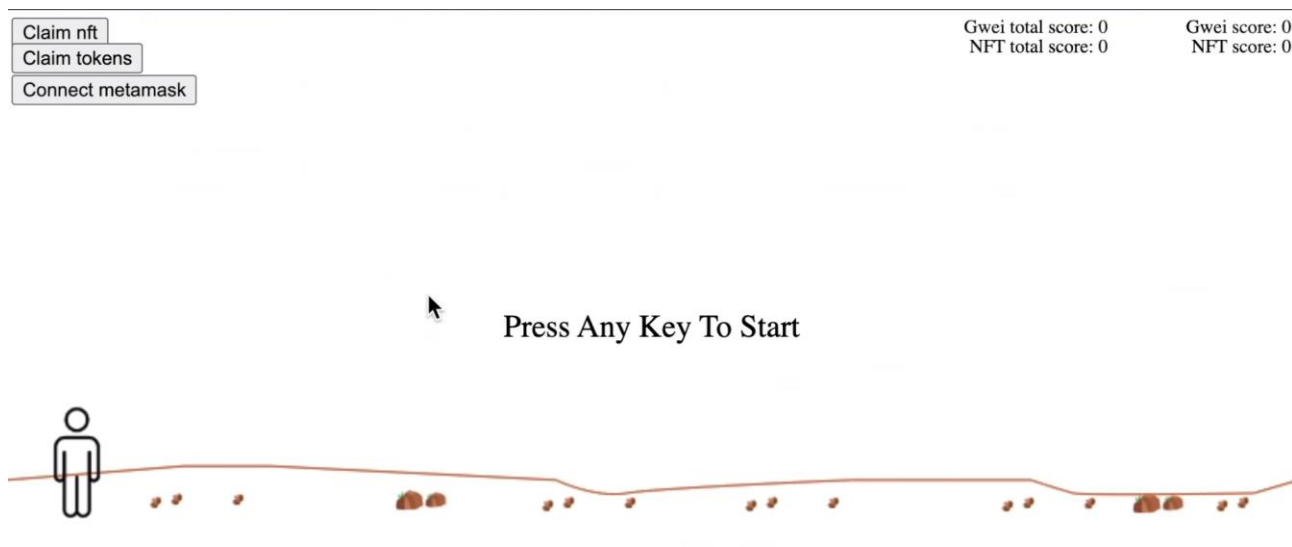


Рисунок 4.1 – Головне меню

Для того, щоб перевести токени на крипто гаманець необхідно спочатку підключити MetaMask до гри. Для підключення необхідно натиснути на кнопку «Connect to MetaMask», після чого з'явиться вікно програми MetaMask, де необхідно буде авторизуватися через введення пароля. На рисунку 4.2 зображено вікно авторизації в MetaMask.

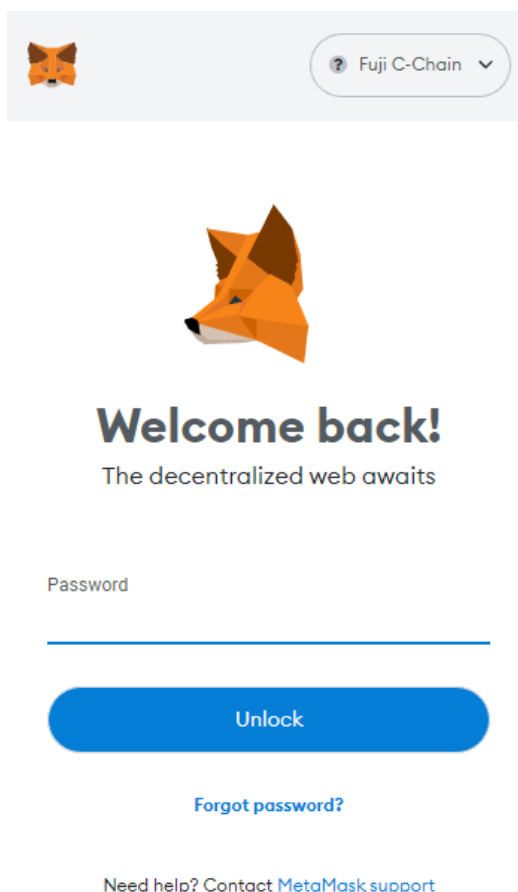


Рисунок 4.2 – Авторизація в MetaMask

Після програшу необхідно натиснути на кнопку «Claim tokens» для того, щоб зібрані бали замінити в блокчейні Ethereum і потім мати можливість обміняти їх на будь-яку іншу криптовалюту з метою виведення реальних грошей. На рисунку 4.3 зображено вікно підтвердження мінтингу токенів в блокчейн Ethereum.

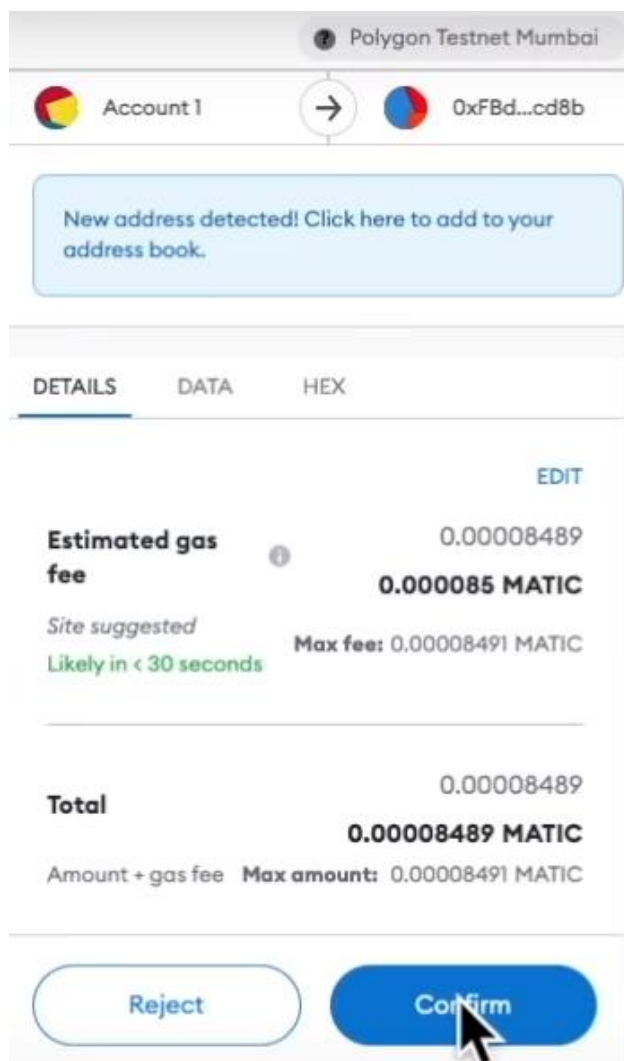
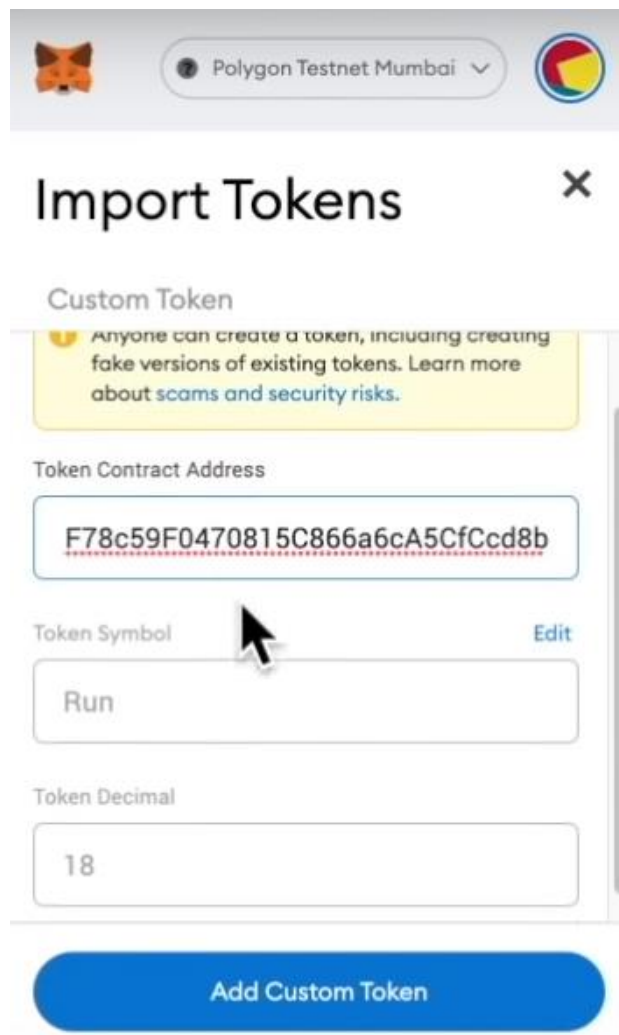


Рисунок 4.3 – Підтвердження мінтингу tokenів в блокчейн Ethereum

Після мінтингу необхідно імпортувати дані токени на свій рахунок крипто гаманця з рахунку смартконтракту гри. Це робиться натискання на кнопку «Import tokens» в вікні MetaMask. На рисунку 4.4. зображено вікно імпорту tokenів.



Polygon Testnet Mumbai

Import Tokens

Custom Token

Anyone can create a token, including creating fake versions of existing tokens. Learn more about [scams and security risks](#).

Token Contract Address

F78c59F0470815C866a6cA5CfCcd8b

Token Symbol [Edit](#)

Run

Token Decimal

18

Add Custom Token

Рисунок 4.4 – Вікно імпорту токенів з адреси смартконтракта гри

На рисунку 4.5 зображено рахунок крипто гаманця з токенами, які гравець заробив під час проходження гри.

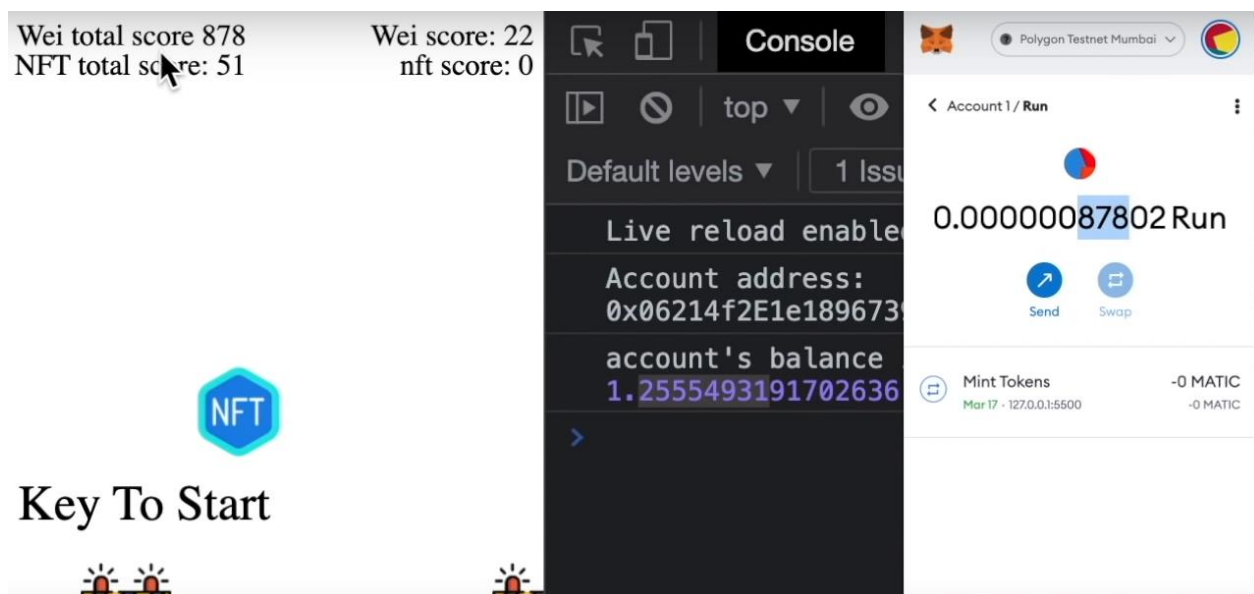


Рисунок 4.5 – Токени на рахунку крипто гаманця гравця

4.5 Висновки до розділу 4

Вирішена задача – розробити інструкції з використання програмного продукту. Це надає кінцевому користувачеві розуміння, яким чином взаємодіяти з ігровою програмою.

У розділі вирішено завдання визначити та описати системні вимоги для запуску та продуктивного функціонування ігрового додатку. Підходить будь-який веббраузер на ПК.

У розділі вирішено завдання детально описати функціонал програми, який повністю відповідає функціональним вимогам: проходження безкінечного рівня, користувацький інтерфейс для підключення та мінтингу tokenів, графічний інтерфейс відображення кількості tokenів та nft.

Описано інструкцію з запуску системи.

Детально описано як взаємодіяти з додатком.

ВИСНОВКИ

В результаті виконання даної дипломної роботи розглянуто технологію блокчейну. Проаналізовані, які бувають блокчейни, плюси та мінуси кожного. Було наведено приклади користування технологією блокчейн серед різних сфер життєдіяльності.

Були розглянуто та детально описано ігровий рушій на якому розроблюються браузерні ігри. Проаналізовано фреймворки, які використовуються для розробки та тестування смартконтрактів. Розглянуто переваги та недоліки кожного фреймворку і було обрано найкращий для реалізації поставленої мети.

Перевагами Hardhat є:

- якісна документація;
- інструменти для тестування та розгортання смартконтрактів в інтегрованій віртуальній машині.

Недоліками Truffle є:

- велика кількість непотрібного функціонала для виконання мети проекту;
- потрібно встановлювати Ganache для клонування проекту
- відсутність інтегрованої віртуальної машини для тестування смартконтрактів, є лише доступ до перевірки помилок
- не зручна документація

Після кінцевого аналізу по зазначеним критеріям обрано саме Hardhat, оскільки саме він має необхідний функціонал для тестування смартконтрактів та досить легкий у вивченні через зручну документацію. .

Були розглянуті сучасні середовища програмування (IDE – Integrated Development Kit) для написання програмного коду. В результаті був обраний Visual Studio Code, оскільки він підтримує синтаксис javascript та інтеграцію з Node.js API.

Були описані загальні складові ігрового процесу, такі як «Дія», «Реакція», «Відгук».

Описано структуру проекту та архітектурні шаблони, які використовувалися для проектування архітектури проекту. Серед зазначених шаблонів були: MVC (Model-View-Controller) та Dependency Injection.

Описано процес мінтингу токенів в блокчейн Ethereum.

Була описана система графічного користувацького інтерфейсу.

Розглянуто призначення та умови використання ігрового додатку. Детально описано функції системи. Наведено інструкцію для користувачів як встановлювати та як використовувати цей програмний продукт.

В результаті була створено ігрову програма з використанням blockchain технологій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Blockchain [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/Blockchain>
2. Public and private blockchains [Electronic resource]. – Access mode: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains>
3. Блокчейн [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Блокчейн>
4. OpenSea [Electronic resource]. – Access mode: <https://opensea.io/>
5. Decentraland [Electronic resource]. – Access mode: <https://decentraland.org/>
6. MetaMask [Electronic resource]. – Access mode: <https://metamask.io/>
7. Фриман А. Pro ASP.NET MVC 4, 4th edition / А. Фриман – М.: «Вильямс», 2013. – 688 с.
8. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship. — Pearson Education, 2008. — P. 157.
9. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования = Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. — 3-е изд. — М.: Вильямс, 2006. — 736 с.
10. Game engine [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Game_engine
11. Software framework [Electronic resource]. – Access mode: https://en.wikipedia.org/wiki/Software_framework
12. Phaser [Electronic resource]. – Access mode: <https://phaser.io/>
13. Node.js [Electronic resource]. – Access mode: <https://nodejs.org/en/>
14. Hardhat [Electronic resource]. – Access mode: <https://hardhat.org/>
15. Truffle [Electronic resource]. – Access mode: <https://trufflesuite.com/>
16. Hardhat vs Truffle [Electronic resource]. – Access mode: <https://101blockchains.com/hardhat-vs-truffle/>

17. Visual Studio Code [Electronic resource]. – Access mode:
<https://code.visualstudio.com/>

18. JetBrains Rider [Electronic resource]. – Access mode:
<https://www.jetbrains.com/>

19. Gameplay [Electronic resource]. – Access mode:
<https://en.wikipedia.org/wiki/Gameplay>

20. Игровой процесс [Электронный ресурс]. – Режим доступа:
https://uk.wikipedia.org/wiki/Игровой_процес

ДОДАТОК А
ТЕКСТ ПРОГРАМИ

A.1 blockchain.js

```

let provider = new ethers.providers.Web3Provider(window.ethereum)
let signer

async function connectMetamask() {
  await provider.send("eth_requestAccounts", []);
  signer = await provider.getSigner();
  console.log("Account address:", await signer.getAddress());

  const balance = await signer.getBalance()
  const convertToEth = 1e18;
  console.log("account's balance in ether:", balance.toString()
/ convertToEth);
}

async function claimTokens() {
  const runTokenContractAddress =
"0x00bcD49DcEaDbB4b9c7fF9E54a64CF2ad61ead61";
  const runTokenContractAbi = [
    "function mintTokens(address account, uint256 amount)
public",
  ];
  const runTokenContract = new
ethers.Contract(runTokenContractAddress, runTokenContractAbi,
provider);
  let convertToWei = 10000000000
  let amountToClaim = window.totalGweiScore * convertToWei
  await
runTokenContract.connect(signer).mintTokens(signer.getAddress(),
amountToClaim.toString())
}

async function claimNft() {
  const nftContractAddress =
"0x093298F529abdfCdA505e5C80A896478Dc32841a";
  const mintContractAbi = [
    "function mint(uint256 amount) public",
  ];
  const nftContract = new ethers.Contract(nftContractAddress,
mintContractAbi, provider);
  await
nftContract.connect(signer).mint(window.totalNFTScore.toString())
}

```

A.2 ground.js

```
import {
  getCustomProperty,
  incrementCustomProperty,
  setCustomProperty,
} from "../updateCustomProperty.js"

const SPEED = 0.05

const groundElems = document.querySelectorAll("[data-ground]")

export function setupGround() {
  setCustomProperty(groundElems[0], "--left", 0)
  setCustomProperty(groundElems[1], "--left", 300)
}

export function updateGround(delta, speedScale) {
  groundElems.forEach(ground => {
    incrementCustomProperty(ground, "--left", delta *
speedScale * SPEED * -1)
    // reloop - pak pravi loop za da ne run out of space
    if (getCustomProperty(ground, "--left") <= -300) {
      incrementCustomProperty(ground, "--left", 600)
    }
  })
}
```

A.3 index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <link rel="stylesheet" href="styles.css">
  <script src="script.js" type="module"></script>
</head>
<body>
  <button      class="button1"      onclick="claimNft()">Claim
nft</button>
  <button      class="button2"      onclick="claimTokens()">Claim
tokens</button>
  <button class="button3" onclick="connectMetamask()">Connect
metamask</button>
  <span class="nftTotalScore" data-nft-total-score>NFT total
score: 0</span>
  <span      class="gweiTotalScore"      data-wei-total-score>Gwei
total score: 0</span>

  <span class="score1" data-nft-score>NFT score: 0</span>
  <div class="score" data-score>Gwei score: 0</div>
<div class="game" data-game>
  <div class="start-screen" data-start-screen>Press Any Key
To Start</div>
  
  
  
</div>

```

```

        <script                                src="https://cdn.ethers.io/lib/ethers-
5.2.umd.min.js"
        type="application/javascript"></script>

        <script src="blockchain.js" ></script>
</body>
</html>

```

A.4 obstacle.js

```

import {
    setCustomProperty,
    incrementCustomProperty,
    getCustomProperty,
} from "../updateCustomProperty.js"

const SPEED = 0.05
const OBSTACLE_INTERVAL_MIN = 500
const OBSTACLE_INTERVAL_MAX = 2000
const gameElem = document.querySelector("[data-game]")

let nextNFTTime
export function setupNft() {
    // spawn an obstacle quickly once the game starts
    nextNFTTime = OBSTACLE_INTERVAL_MIN
    // remove all obstacles before the game starts again
    document.querySelectorAll("[data-nft]").forEach(nft => {
        nft.remove()
    })
}

export function updateNft(delta, speedScale) {
    document.querySelectorAll("[data-nft]").forEach(nft => {

```

```

        incrementCustomProperty(nft, "--left", delta *
speedScale * SPEED * -1)
        if (getCustomProperty(nft, "--left") <= -100) {
            nft.remove()
        }
    })

    if (nextNFTTime <= 0) {
        createNft()
        nextNFTTime =
            randomNumberBetween(OBSTACLE_INTERVAL_MIN,
OBSTACLE_INTERVAL_MAX) / speedScale
    }
    nextNFTTime -= delta
}

export function getNftRects() {
    return [...document.querySelectorAll("[data-
nft]")].map(nft => {
        return nft.getBoundingClientRect()
    })
}

function createNft() {
    const nft = document.createElement("img")
    nft.dataset.nft = true
    // todo: set img drugo nft
    nft.src = "imgs/nft.png"
    // todo: smeni klasa nft
    nft.classList.add("nft")
    setCustomProperty(nft, "--left", 100)
    gameElem.append(nft)
}

function randomNumberBetween(min, max) {

```

```

    return Math.floor(Math.random() * (max - min + 1) + min)
}

```

A.5 player.js

```

import {
  incrementCustomProperty,
  setCustomProperty,
  getCustomProperty,
} from "../updateCustomProperty.js"

const playerElem = document.querySelector("[data-player]")
const JUMP_SPEED = 0.45
const GRAVITY = 0.0015
const PLAYER_FRAME_COUNT = 2
const FRAME_TIME = 100

let isJumping
let playerFrame
let currentFrameTime
let yVelocity
export function setupPlayer() {
  isJumping = false
  playerFrame = 0
  currentFrameTime = 0
  yVelocity = 0
  setCustomProperty(playerElem, "--bottom", 0)
  document.removeEventListener("keydown", onJump)
  document.addEventListener("keydown", onJump)
}

export function updatePlayer(delta, speedScale) {
  handleRun(delta, speedScale)
  handleJump(delta)
}

```

```

export function getPlayerRect() {
    return playerElem.getBoundingClientRect()
}

export function setPlayerLose() {
    playerElem.src = "imgs/playerJumps.png"
}

function handleRun(delta, speedScale) {
    if (isJumping) {
        playerElem.src = `imgs/playerJumps.png`
        return
    }

    if (currentFrameTime >= FRAME_TIME) {
        playerFrame = (playerFrame + 1) % PLAYER_FRAME_COUNT
        playerElem.src = `imgs/playerRuns${playerFrame}.png`
        currentFrameTime -= FRAME_TIME
    }
    currentFrameTime += delta * speedScale
}

function handleJump(delta) {
    if (!isJumping) return

    incrementCustomProperty(playerElem, "--bottom", yVelocity *
delta)

    if (getCustomProperty(playerElem, "--bottom") <= 0) {
        setCustomProperty(playerElem, "--bottom", 0)
        isJumping = false
    }

    yVelocity -= GRAVITY * delta
}

```

```
function onJump(e) {
  if (e.code !== "Space" || isJumping) return

  yVelocity = JUMP_SPEED
  isJumping = true
}
```

A.6 main.js

```
import { updateGround, setupGround } from "../ground.js"
import { updatePlayer, setupPlayer, getPlayerRect,
setPlayerLose } from "../player.js"
import { updateObstacle, setupObstacle, getObstacleRects }
from "../obstacle.js"
import { updateNft, setupNft, getNftRects } from "../nft.js"

const GAME_WIDTH = 100
const GAME_HEIGHT = 30
const SPEED_SCALE_INCREASE = 0.00001

const gameElem = document.querySelector("[data-game]")
const scoreElem = document.querySelector("[data-score]")
const startScreenElem = document.querySelector("[data-start-
screen]")
const gweiTotalScoreElem = document.querySelector("[data-
wei-total-score]")
const nftTotalScoreElem = document.querySelector("[data-nft-
total-score]")

const nftScoreElem = document.querySelector("[data-nft-score
]")

setPixelToGameScale()
window.addEventListener("resize", setPixelToGameScale)
document.addEventListener("keydown", handleStart, { once:
true })
```

```

let lastTime
let speedScale
let score
let nftScore = 0

function update(time) {
  if (lastTime == null) {
    lastTime = time
    window.requestAnimationFrame(update)
    return
  }
  const delta = time - lastTime

  updateGround(delta, speedScale)
  updatePlayer(delta, speedScale)
  updateObstacle(delta, speedScale)
  updateNft(delta, speedScale)
  updateSpeedScale(delta)
  updateScore(delta)
  checkIfWeGotNft()
  // if there is a collision lose the game
  if (checkLose()) return handleLose()
  lastTime = time
  window.requestAnimationFrame(update)
}

function checkLose() {
  const playerRect = getPlayerRect()
  return getObstacleRects().some(rect => isCollision(rect,
playerRect))
}

```

```

function isCollision(rect1, rect2) {
  return (
    rect1.left < rect2.right &&
    rect1.top < rect2.bottom &&
    rect1.right > rect2.left &&
    rect1.bottom > rect2.top
  )
}

function checkIfWeGotNft() {
  const playerRect = getPlayerRect()
  if(getNftRects().some(rect => isCollision(rect,
playerRect))) {
    const nftToRemove = document.querySelectorAll("[data-
nft]")[0]
    nftToRemove.remove()
    nftScore += 1
    nftScoreElem.textContent = `nft score: ${nftScore}`
  }
  return getNftRects().some(rect => isCollision(rect,
playerRect))
}

function updateSpeedScale(delta) {
  speedScale += delta * SPEED_SCALE_INCREASE
}

function updateScore(delta) {
  // kolku score dobivas tuka se presmetue
  // console.log("delta u update score function = ", delta)
  score += delta * 0.01
  scoreElem.textContent = `Wei score: ${Math.floor(score)}`
}

```

```

// kaa kje pozne
function handleStart() {
  lastTime = null
  speedScale = 1
  score = 0
  setupGround()
  setupPlayer()
  setupObstacle()
  setupNft()
  startScreenElem.classList.add("hide")
  // call this onnly wehne screen refershes
  window.requestAnimationFrame(update)
}

window.totalNFTScore = 0
window.totalGweiScore = 0

function handleLose() {
  window.totalGweiScore += Math.floor(score)
  window.totalNFTScore += nftScore

  nftTotalScoreElem.textContent = `NFT total score:
${window.totalNFTScore}`
  gweiTotalScoreEleme.textContent = `Wei total score
${window.totalGweiScore}`

  nftScore = 0
  nftScoreElem.textContent = `nft score: ${nftScore}`
  setPlayerLose()
  // save
  setTimeout(() => {
    document.addEventListener("keydown", handleStart, { once:
true })

```

```

        startScreenElem.classList.remove("hide")
    }, 100)
}

function setPixelToGameScale() {
    let gameToPixelScale
    if (window.innerWidth / window.innerHeight < GAME_WIDTH /
GAME_HEIGHT) {
        gameToPixelScale = window.innerWidth / GAME_WIDTH
    } else {
        gameToPixelScale = window.innerHeight / GAME_HEIGHT
    }

    gameElem.style.width = `${GAME_WIDTH * gameToPixelScale}px`
    gameElem.style.height = `${GAME_HEIGHT *
gameToPixelScale}px`
}

```

A.7 styles.css

```

*, *::before, *::after {
    box-sizing: border-box;
    user-select: none;
}

body {
    margin: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    min-height: 100vh;
}

.game {

```

```
    overflow: hidden;
    position: relative;
}
```

```
button {
    position: fixed;
    margin-top: 1px;
}
```

```
.nftTotalScore {
    position: absolute;
    font-size: 3vmin;
    right: 30vmin;
    top: 4vmin;
}
```

```
.gweiTotalScore {
    position: absolute;
    font-size: 3vmin;
    right: 30vmin;
    top: 1vmin;
}
```

```
.score {
    position: absolute;
    font-size: 3vmin;
    right: 1vmin;
    top: 1vmin;
}
```

```
.button1 {
    position: absolute;
    font-size: 3vmin;
    left: 1vmin;
    top: 1vmin;
```

```
}

.button2 {
  position: absolute;
  font-size: 3vmin;
  left: 1vmin;
  top: 5vmin;
}

.button3 {
  position: absolute;
  font-size: 3vmin;
  left: 1vmin;
  top: 10vmin;
}

.score1 {
  position: absolute;
  font-size: 3vmin;
  right: 1vmin;
  top: 4vmin;
}

.start-screen {
  position: absolute;
  font-size: 5vmin;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

.hide {
  display: none;
}
```

```
}

.ground {
  --left: 0;
  position: absolute;
  width: 300%;
  bottom: 0;
  left: calc(var(--left) * 1%)
}

.player {
  --bottom: 0;
  position: absolute;
  left: 1%;
  height: 30%;
  bottom: calc(var(--bottom) * 1%);
}

.obstacle {
  position: absolute;
  left: calc(var(--left) * 1%);
  height: 30%;
  bottom: 0;
}

.nft {
  position: absolute;
  left: calc(var(--left) * 1%);
  height: 30%;
  bottom: 100px;
}
```

A.8 styles.css

```

// get css variable into js update it and back into css
export function getCustomProperty(elem, prop) {
  // getComputedStyle allows for getting css value
  // this returns string we will convert to float
  // the default is 0
  return
parseFloat(getComputedStyle(elem).getPropertyValue(prop)) || 0
}

export function setCustomProperty(elem, prop, value) {
  elem.style.setProperty(prop, value)
}

export function incrementCustomProperty(elem, prop, inc) {
  setCustomProperty(elem, prop, getCustomProperty(elem, prop)
+ inc)
}

```

A.9 deployNFTECollection.js

```

const hre = require("hardhat");

async function main() {
  // Hardhat always runs the compile task when running scripts
with its command
  // line interface.
  //
  // If this script is run directly using `node` you may want
to call compile
  // manually to make sure everything is compiled
  // await hre.run('compile');

  // We get the contract to deploy

```

```

const Greeter = await
hre.ethers.getContractFactory("RunnerCollection");
const greeter = await Greeter.deploy(
  "Runner",
  "RUN",

  "https://ipfs.io/ipfs/QmTCsMc4vjxTSx4vfBz8ZDaugvF3XGcdWuxcN8HN8Cob
J6/"
);

await greeter.deployed();
console.log("Greeter deployed to:", greeter.address);

await greeter.mint(10);
console.log("10 nfts minted")
}

// We recommend this pattern to be able to use async/await
everywhere
// and properly handle errors.
main()
  .then(() => process.exit(0))
  .catch((error) => {
    console.error(error);
    process.exit(1);
  });

```

A.10 deployRunToken.js

```
const hre = require("hardhat");
```

```

    async function main() {
        // Hardhat always runs the compile task when running scripts
with its command
        // line interface.
        //
        // If this script is run directly using `node` you may want
to call compile
        // manually to make sure everything is compiled
        // await hre.run('compile');

        // We get the contract to deploy
        const Greeter = await
hre.ethers.getContractFactory("RunToken");
        const greeter = await Greeter.deploy();

        await greeter.deployed();
        console.log("run token deployed to:", greeter.address);

        await
greeter.mintTokens("0x06214f2E1e1896739D92F3526Bd496DC028Bd7F9",
55);

        console.log("run tokens minted")
    }

    // We recommend this pattern to be able to use async/await
everywhere
    // and properly handle errors.
    main()
        .then(() => process.exit(0))
        .catch((error) => {
            console.error(error);
            process.exit(1);
        });

```

A.11 Greeter.sol

```

pragma solidity ^0.8.0;

import "hardhat/console.sol";

contract Greeter {
    string private greeting;

    constructor(string memory _greeting) {
        console.log("Deploying a Greeter with greeting:",
_greeting);
        greeting = _greeting;
    }

    function greet() public view returns (string memory) {
        return greeting;
    }

    function setGreeting(string memory _greeting) public {
        console.log("Changing greeting from '%s' to '%s'",
greeting, _greeting);
        greeting = _greeting;
    }
}

```

A.12 RunToken.sol

```

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract RunToken is ERC20 {
    constructor() ERC20("RunToken", "Run"){
        _mint(msg.sender, 20*10**6);
    }
}

```

```

    }

    function mintTokens(address account, uint256 amount)
public {
    _mint(account, amount);
}
}

```

A.13 RunToken.sol

```

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
import "@openzeppelin/contracts/utils/Strings.sol";

contract RunnerCollection is ERC1155 {

    string public name;
    string public symbol;
    uint256 public tokenCount;
    string public baseUri;

    constructor(string memory _name, string memory _symbol,
string memory _baseUri) ERC1155(_baseUri) {
        name = _name;
        symbol = _symbol;
        baseUri = _baseUri;
    }

    // we need to override the uri function so our nfts are
compatible with opensea
    function uri(uint256 _tokenId) override public view
returns(string memory) {
        return string (
            abi.encodePacked(

```

```
        baseUri,  
        Strings.toString(_tokenId),  
        ".json"  
    )  
    );  
}  
  
function mint(uint256 amount) public {  
    tokenCount += 1;  
    _mint(msg.sender, tokenCount, amount, "");  
}  
}
```

ДОДАТОК Б
СЛАЙДИ ПРЕЗЕНТАЦІЇ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Дипломна кваліфікаційна робота на тему:

ДОСЛІДЖЕННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТЕХНОЛОГІЙ
БЛОКЧЕЙНУ
RESEARCH AND DEVELOPMENT OF SOFTWARE FOR BLOCKCHAIN TECHNOLOGY

Виконав: ст. гр. КНТ-211м

Є.С. Лизя

Керівник: проф.

С.О Субботін

Рисунок Б.1 – Титульний слайд

МЕТА ТА ЗАВДАННЯ РОБОТИ

2

Мета роботи - програмна реалізація play to earn браузерної ігрової програми в жанрі платформеру з метою експертного дослідження технологій блокчейну Ethereum.

ЗАДАЧІ РОБОТИ

аналіз вимог до системи;

розробка архітектури системи;

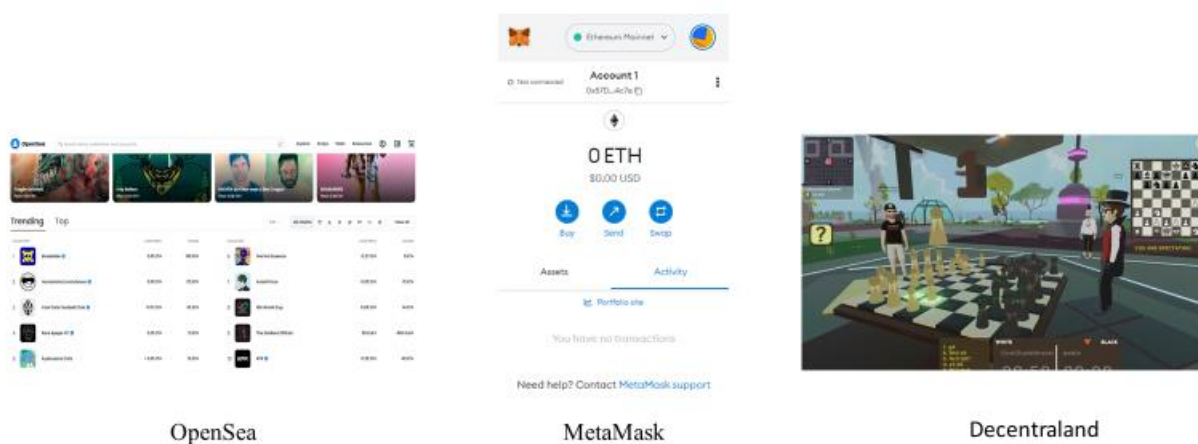
розробка дизайну клієнтської частини;

вибір інструментарію розробки;

програмна реалізація віртуального середовища;

розробка інструкцій з використання програми.

Рисунок Б.2 – Мета та завдання роботи



OpenSea

MetaMask

Decentraland

Рисунок Б.3 – Приклади використання

Розроблювана ігрова програма в жанрі платформеру повинна бути оптимізованою для мобільних девайсів, зручною у використанні та цікавою в процесі використання.

Програмний продукт повинен містити наступний функціонал:

користувальницький інтерфейс головного меню

підключення віртуального гаманця MetaMask

проведення транзакцій перед початком і кінцем гри

ігрова механіка стрільби по цілях

налаштування гри

Рисунок Б.4 – Аналіз функціональних вимог

ПОРІВНЯННЯ ФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ

5

Критерій	Bardot	Truffle
Документація	Проста и зрозуміла документація з туторіалами	Комплексна документація і онлайн ресурси
Спільнота	Нове ком'юніті з розробниками і дискорд сервером	Велике ком'юніті з підтримкою
Бібліотеки та плагіни	Підтримка багатьох плагінів та бібліотек	Truffle Boxes та front-end бібліотеки в Drizzle
Тестування	Гнучке тестування з підтримкою смартконтрактів	Відсутність підтримки прямого контакту зі смартконтрактами під час тестування
Розгортання контрактів	Розробники можуть писати власні скрити та використовувати плагіни для додавання нових можливостей	Фіксований шаблон обмежує розгортання контрактів
Обробка помилок	Легкий доступ до управління помилками і інструменти тестування з трасуванням стеку	Доступ лише до помилок
Легкість розгладження	Легке розгладження без додаткових інсталяцій	Потрібно встановити Ganache
Інтеграція TypeScript	Легкий процес інтеграції TypeScript	Тяжкий процес інтеграції TypeScript
Взаємодія зі смарт контрактами	Ethers.js бібліотека за замовчуванням для взаємодії зі смарт контрактами	Web3.js бібліотека за замовчуванням для взаємодії зі смарт контрактами
Компанії, які використовують фреймворк	Aave, Optimism, OpenZeppelin	JP Morgan Chase, BitGo, 1inch

Рисунок Б.5 – Порівняння фреймворків для розробки

ВИБІР СЕРЕДОВИЩА ПРОГРАМУВАННЯ

6

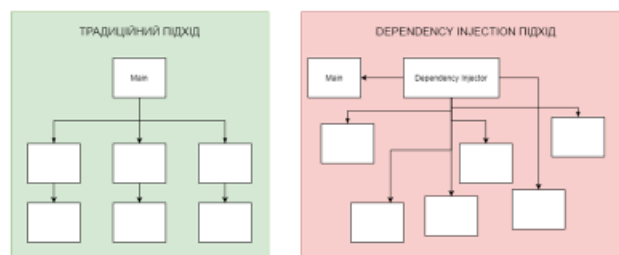
Порівняння IDE

Критерій	Rider	VisualStudio Code
Можливість роботи на різних платформах	+	+
Безкоштовна ліцензія	-	+
Якісна документація	+	+
Підтримка Node.js	-	+
Підтримка javascript	-	+
Система контролю версій	+	+
Присмний дизайн	+	+
Оптимізованість	+	+
Декомпіляція зовнішніх бібліотек	+	-

Рисунок Б.6 – Вибір середовища програмування



Схема MVC



Dependency Injection

Рисунок Б.7 – Архітектура системи

Опис файлів

ground.js	Реалізує логіку анімації поверхності на якій знаходиться головний герой, за рахунок чого створюється імітація бігу головного героя.
player.js	Реалізує логіку переміщення головного героя, прискорення, оновлення та включення/виключення графічних ефектів. Містить функції зчитування доторкань, свайпів та інших маніпуляцій, які походять від користувача.
obstacle.js	Реалізує появлення, видалення та оновлення стану у разі перетину з головним героєм.
nft.js	Реалізує появлення nft (non fungible token) у вигляді ігрових об'єктів, які герой може збирати і які потім конвертуються у криптовалюту, за рахунок чого реалізується концепція play to earn
RunToken.sol	Реалізує токен з протоколом ERC20
RunnerCollection.sol	Змінює ідентифікатор зібраних користувачем токенів та представляю собою колекцію ERC1155, які буде розгортатися на платформі OpenSea.
deployRunToken.js	Реалізує мінт токenu (інтеграції в блокчейн Ethereum), який потім відображається у віртуальному гаманці користувача як окрема криптовалюта
deployNFTCollection.js	Реалізує мінт колекції зібраних токенів на платформі OpenSea, які потім можна переглянути як колекцію зображень

Рисунок Б.8 – Опис файлів



Рисунок Б.9 – Взаємодія з програмним продуктом

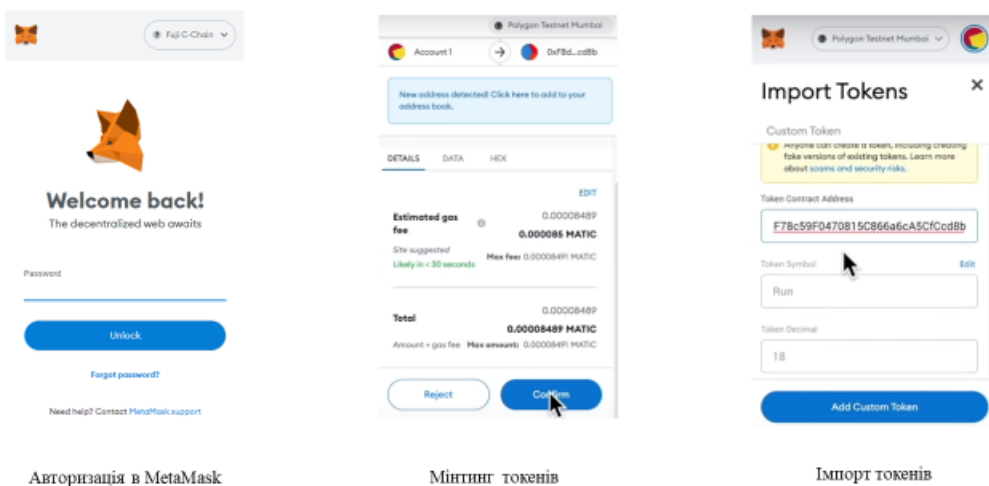
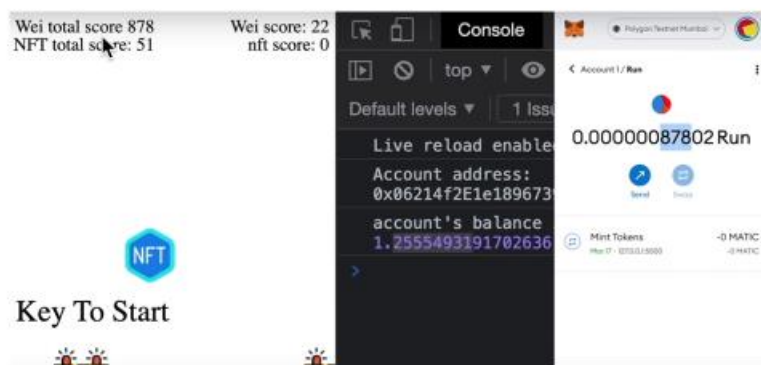


Рисунок Б.10 – Взаємодія з програмним продуктом



Зарахування крипто валюти на рахунок користувача

Рисунок Б.11 – Взаємодія з програмним продуктом

ВИСНОВКИ
Розглянуто технологію блокчейну
Проаналізовані, які бувають блокчейни, плюси та мінуси кожного
Наведено приклади користування технологією блокчейн серед різних сфер життєдіяльності
Проаналізовано фреймворки, які використовуються для розробки та тестування смартконтрактів. Розглянуто переваги та недоліки кожного фреймворку і було обрано найкращий для реалізації поставленої мети
Описано процес мінтингу токенів в блокчейн Ethereum.
Створено ігрову програму з використанням blockchain технологій

Рисунок Б.12 – Висновки