Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase»

Виконав: студент 4 курсу групи 2ПІ-20б спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Кучер М.В.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Кожемяко А.В.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ Романюк О.Н.
«10» червня 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н._____
«12» березня 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Кучеру Максиму Володимировичу

1. Тема роботи – «Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase».

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3 червня 2024 р.

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки TypeScript, фреймворк React, платформа розробки застосунків Firebase.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та постановка задачі; аналіз даних; розробка алгоритму гри, розробка алгоритму роботи штучного інтелекту, розробка методу формування рейтингу користувача; інтеграція сервісів Firebase, розробка серверної частини застосунку; аналіз методів тестування, тестування розробленого функціоналу; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд, актуальність теми, актуальність теми, мета, об'єкт та предмет дослідження, завдання, наукова новизна,

практична цінність, аналоги, порівняльний аналіз аналогів, використані техн
гії, алгоритм роботи ШІ (Minimax), алгоритм процесу гри, метод розрахунку
тингу користувача, тестування авторизації, тестування взаємодії з базою да
висновки, висновки, апробація результатів роботи і публікації, завершал
слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання ви-дав	завда прий
1-4	Майданюк В.П, к.т.н., доцент кафедри ПЗ	13.03.24	03.06.2

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	При
1	Аналіз стану алгоритмів додатків для розвитку логічного мислення	14.03.24 – 22.03.24	вик
2	Розробка алгоритмів роботи штучного інтелекту	13.03.24 – 22.03.24	вик
3	Аналіз і вибір мови програмування та середовища розробки	25.03.24 – 19.04.24	вик
4	Розробка серверної частини застосунку	22.04.24 – 10.05.24	вик
5	Тестування розробленого функціоналу	13.05.24 – 24.05.24	вик
6	Оформлення матеріалів до захисту БДР	27.05.24 – 30.05.24	вик

Студент _____ (підпис) Кучер М.Р (прізвище та ініці

Керівник бакалаврської дипломної роботи _____ (підпис) Майданюк (прізвище та ініці

АНОТАЦІЯ

УДК 004.42

Кучер М.В. «Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase»: бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 56 с.

На укр. мові. Бібліогр. : 18 назв; рис. : 27; табл. : 3.

У бакалаврській дипломній роботі проведено розробку алгоритмів гри web-застосунку для розвитку логічного мислення та інтеграцію сервісів Firebase. Виконано аналіз сучасних підходів до реалізації алгоритмів ігор для розвитку логічного мислення. Обґрунтовано необхідність розробки web-застосунку для розвитку логічного мислення з метою покращення можливостей розвитку логічного мислення людини.

Реалізовано алгоритм штучного інтелекту, який грає проти користувача, та розроблено серверну частину застосунку з використанням сервісу Firebase. Застосунок реалізовано з використанням фреймворку ReactJS мовою програмування TypeScript в середовищі розробки Visual Studio Code

Тестування підтвердило коректність інтеграції серверної частини з сервісом Firebase та коректну роботу алгоритму.

Отримані в бакалаврській дипломній роботі результати можна використати для інтеграції із інтерфейсом користувача WEB-застосунку, що дозволяє розвивати логічне мислення.

Ключові слова: web-застосунок, логічне мислення, Firebase, MiniMax.

ANNOTATION

UDC 004.42

Kucher M.V. "Development of a gaming WEB application for the development of logical thinking. Part 2. Implementation of the game algorithm and integration of the Firebase service: bachelor's thesis on the specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2024. 56 p.

In Ukrainian. Bibliography: 18 titles; Figures: 27; Tables: 3.

In the bachelor's thesis, the development of game algorithms of the web application for the development of logical thinking and the integration of Firebase services were carried out. An analysis of modern approaches to the implementation of game algorithms for the development of logical thinking was carried out. The need to develop a web application for the development of logical thinking in order to improve the possibilities of developing a person's logical thinking is substantiated.

An artificial intelligence algorithm that plays against the user was implemented, and the server part of the application was developed using the Firebase service. The application was implemented using the ReactJS framework in the TypeScript programming language in the Visual Studio Code development environment

The conducted testing confirmed the correctness of the integration of the server part with the Firebase service and the correct operation of the algorithm.

The results obtained in the bachelor thesis can be used for integration with the user interface of the WEB application, which allows for the development of logical thinking.

Keywords: web application, logical thinking, Firebase, MiniMax.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Розвиток логічного мислення є важливою властивістю людини у сучасному світі. Ця навичка допомагає приймати обдумані рішення, вирішувати завдання, розв'язувати проблеми та ефективно працювати з інформацією. Досягнення у логічному мисленні допомагають в різних сферах життя, включаючи навчання, роботу та особистий розвиток [1].

Додатки для розвитку логічного мислення стали популярним інструментом для тих, хто прагне удосконалити цю навичку. Такі додатки є актуальними з таких причин:

1. Застосунки для розвитку логічного мислення зазвичай використовують інтерактивні завдання, головоломки та ігри, які сприяють виробленню конкретних аспектів логічного мислення. Вони дозволяють користувачам вправлятися в цікавий спосіб та отримувати миттєвий зворотний зв'язок.

2. Ігри та головоломки вимагають від користувачів уваги до деталей та концентрації на завданні. Це сприяє розвитку навичок фокусування, що корисно у повсякденному житті. Для прикладу гра Tic-Tac-Toe сприяє розвитку критичного та стратегічного мислення, оскільки гравці повинні аналізувати хід суперника та передбачати його подальші дії, також необхідно планувати власні ходи, враховуючи можливі варіанти розвитку гри.

3. Застосунок може пропонувати різні рівні складності та індивідуальні налаштування, що дозволяє адаптувати гру до потреб та рівня розвитку кожного користувача. Можливість збереження процесу гри та аналізу помилок допомагає користувачам бачити свої успіхи та працювати над слабкими сторонами.

4. Більшість додатків для розвитку логічного мислення доступні на різних платформах, таких як смартфони, планшети та комп'ютери. Це робить їх доступними для використання в будь-якому місці та в будь-який час.

Тому, з обґрунтування та наведених вище аргументів можна зробити висновок, що тема роботи є актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є покращення можливостей розвитку логічного мислення людини за рахунок розробки алгоритмів та серверної частини web-застосунку, який містить функціонал гри для розвитку логічного мислення.

Основними задачами дослідження є:

- розробити ігрову логіку для WEB-застосунку для розвитку логічного мислення;
- реалізувати алгоритм MiniMax для можливості гри проти штучного інтелекту;
- інтегрувати сервіси Firebase для аутентифікації та для роботи з базою даних;
- провести тестування застосунку.

Об’єкт дослідження – процес розробки ігрового web-застосунку для розвитку логічного мислення.

Предмет дослідження – методи і програмні засоби реалізації алгоритмів ігрового web-застосунку для розвитку логічного мислення.

Методи дослідження. У процесі досліджень використовувались: теорія ігор, теорія алгоритмів, методи штучного інтелекту для реалізації алгоритму гри, документація сервісу Firebase та комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних значень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод формування рейтингу гравця, у якому, на відміну від існуючих, розрахунок кількості рейтингових балів

отриманих за гру враховує не тільки результат гри, але й рівень складності гри, що дозволяє більш точно формувати рейтинг користувачів.

2. Подальшого розвитку отримав функціонал авторизації у додатку, у якому, на відміну від існуючих реалізацій, сесія останньої авторизації зберігається на термін у 7 днів, що дозволяє значно покращити користувацький досвід взаємодії з додатком.

3. Подальшого розвитку отримала реалізація гри, у якій, на відміну від існуючих, додано можливість увімкнути обмеження тривалості ходу у 30 секунд, що дозволяє більш точно емулювати ситуацію прийняття рішення у короткий проміжок часу.

Практична цінність отриманих результатів. Практична цінність одержаних в бакалаврській дипломній роботі результатів полягає у можливості використання розроблених на мові програмування TypeScript алгоритмів та коду серверної частини, що реалізовує інтеграцію з сервісом Firebase, у web-застосунку для розвитку логічного мислення.

Особистий внесок здобувача. Усі результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій праці, опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, алгоритм роботи програми [2].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на «LIII Всеукраїнській науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2024)».

Публікації. Основні результати досліджень опубліковано в 1 науковій праці у матеріалах конференції.

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ РОЗВИТКУ ЛОГІЧНОГО МИСЛЕННЯ

1.1 Аналіз методів побудови алгоритмів веб-додатків для розвитку логічного мислення

Розвиток логічного мислення є важливою навичкою в сучасному динамічному світі. Здатність аналізувати інформацію, виявляти закономірності, робити обґрунтовані висновки та приймати рішення вкрай необхідна для успіху в багатьох сферах діяльності. Тому створення ефективних систем та засобів для тренування цієї навички є актуальним завданням.

WEB-додатки для розвитку логічного мислення набули значної популярності завдяки своїй доступності, інтерактивності та можливості тренуватися у захопливій ігровій формі. Проте, розробка таких додатків вимагає ретельного проектування та застосування ефективних алгоритмів на різних етапах.

Одним з ключових аспектів є генерація різноманітних завдань, головоломок та ігор, які стимулюють навички логічного мислення. Для цього можуть застосовуватися:

1. Готові набори завдань, закладені в базу даних додатку.

2. Процедурна генерація завдань на основі певних правил і випадкових чинників для забезпечення варіативності.

3. Методи машинного навчання для створення нових завдань на основі набору прикладів.

Важливим є також впровадження адаптивного рівня складності завдань, що підлаштовується під поточний рівень користувача. Це можна реалізувати за допомогою:

1. Алгоритмів аналізу ефективності користувача та динамічного коригування складності.

2. Методики поступового підвищення рівня завдань у міру просування.

3. Системи оцінювання для визначення поточного рівня користувача.

Для забезпечення зворотного зв'язку необхідно розробити ефективні алгоритми перевірки правильності відповідей користувача. Варіанти реалізації:

1. Порівняння з очікуваним результатом для завдань з відомим розв'язком.

2. Евристичні та оптимізаційні алгоритми для пошуку оптимальних рішень.

3. Застосування методів машинного навчання для перевірки відповідей.

Крім того, важливими компонентами успішних додатків є системи мотивації та заохочення (бали, рівні, досягнення, змагання), а також зручний інтуїтивний інтерфейс для полегшення взаємодії.

Крім цього, враховуючи, що додаток для розвитку логічного мислення є WEB-додатком, варто зазначити, що у такого типу додатків широко розповсюдженою практикою є використання авторизації/автентифікації та створення аккаунтів для кожного користувача. Це дозволяє додатку вести облік користувачів та прикріплювати до певного аккаунта ігровий прогрес, результати та досягнення, які будуть перенесенні при успішній авторизації користувачем з будь-якого пристрою.

В цілому, розробка якісних веб-додатків для розвитку логічного мислення потребує поєднання класичних підходів до створення ігрових та навчальних систем з новітніми методами штучного інтелекту. Це відкриває можливості для нових інноваційних рішень у даній сфері.

1.2 Порівняльний аналіз аналогів

Розглянемо програмні додатки, які спрямовані на розвиток логічного мислення. До таких входять:

1. CogniFit.
2. Lumosity.
3. Peak.

CogniFit [3] - це програма для тренування мозку, яка пропонує інтерактивні ігри та вправи, спрямовані на покращення різних когнітивних здібностей,

включаючи пам'ять, увагу, логічне мислення, зорово-просторову обізнаність та швидкість реакції. Програма використовує патентовані алгоритми для створення персоналізованих планів тренувань на основі початкової оцінки рівня користувача.

Ключові особливості CogniFit:

1. Адаптивний рівень складності, який змінюється відповідно до прогресу користувача.
2. Регулярні звіти та графіки для відстеження поліпшень.
3. Кілька режимів, включаючи групові тренування та змагання.
4. Можливість створювати персоналізовані ігрові плани для різних цілей.

Інтерфейс CogniFit продемонстровано на рисунку 1.1.

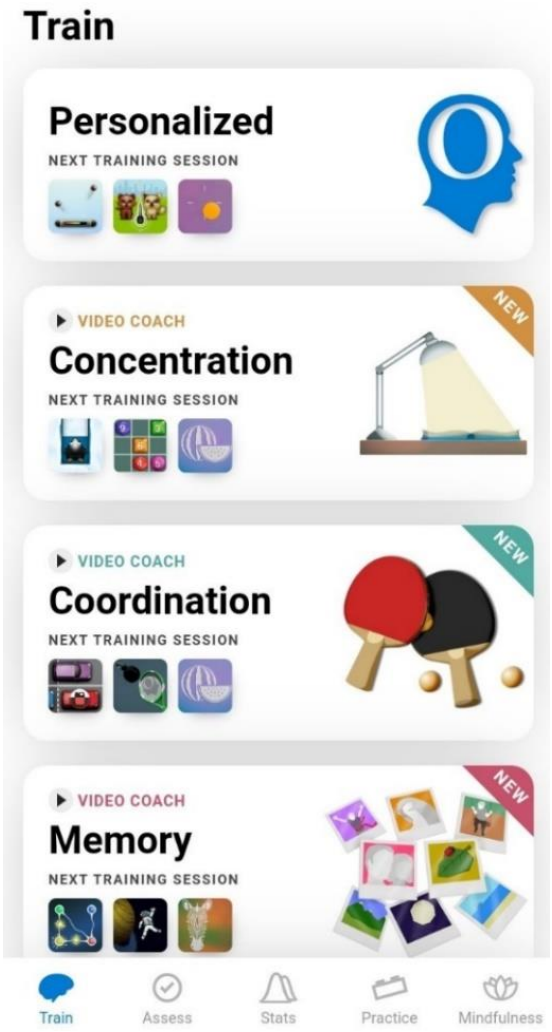


Рисунок 1.1 – Інтерфейс CogniFit

Lumosity [4] - це популярний додаток для розвитку когнітивних здібностей із понад 100 мільйонами користувачів у всьому світі. Він пропонує різноманітні ігрові вправи, що тренують такі навички, як пам'ять, увага, гнучкість мислення, швидкість обробки інформації та вирішення проблем. Lumosity використовує адаптивний підхід, регулярно змінюючи складність завдань відповідно до прогресу користувача.

Ключові особливості Lumosity:

1. Бібліотека ігор та головоломок.
2. Персоналізовані тренувальні програми відповідно до цілей користувача.
3. Щоденні нагадування для підтримки регулярності занять.
4. Оцінка когнітивних здібностей з порівнянням з іншими користувачами.
5. Деталізована статистика та аналітика для відстеження прогресу.

Зовнішній вигляд Lumosity наведено на рисунку 1.2.

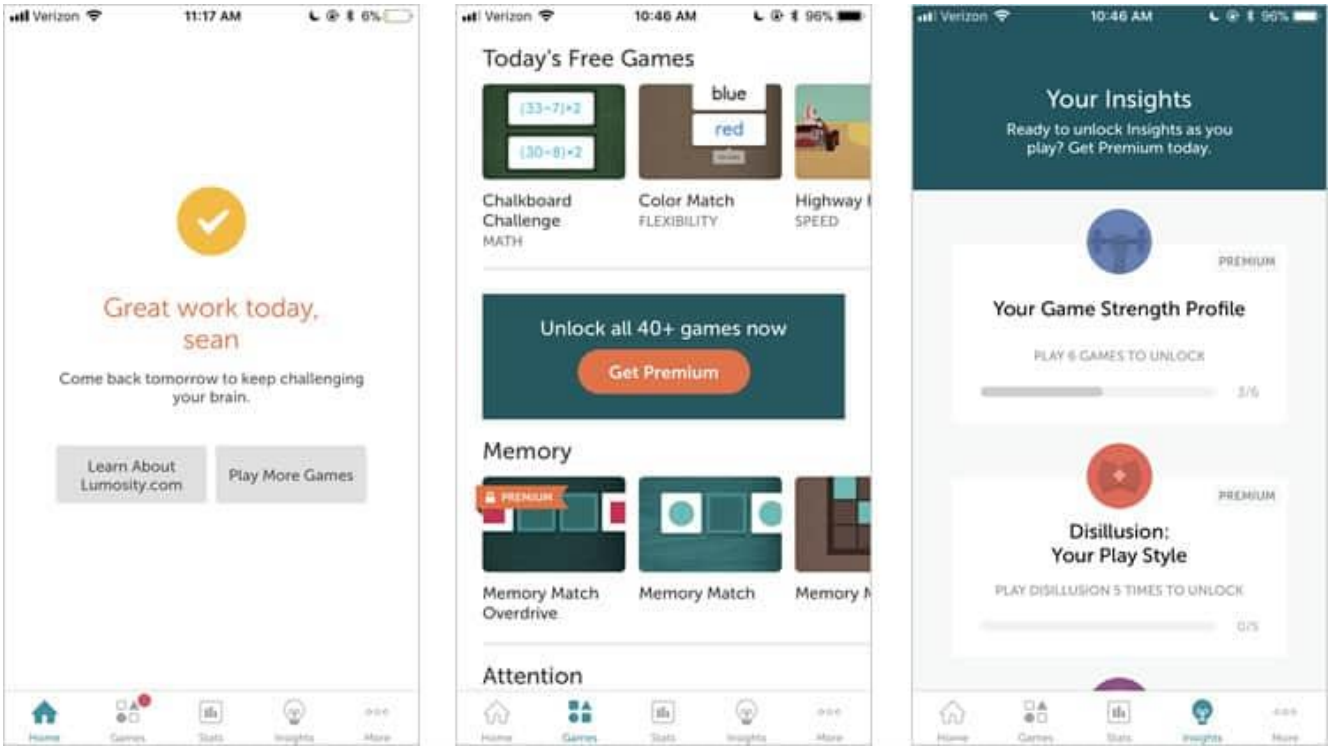


Рисунок 1.2 – Зовншіній вигляд Lumosity

Peak [5] - це мобільний додаток та веб-платформа, що пропонує структуровані програми тренувань для покращення різних аспектів когнітивних

здібностей, включаючи пам'ять, увагу, ментальну гнучкість, логічне мислення та вирішення проблем. Програма розроблена нейробіологами та враховує сучасні дослідження в галузі нейронауки.

Ключові особливості Peak:

1. Понад 40 ігор та вправ з різними рівнями складності.
2. Персоналізовані програми тренувань відповідно до цілей користувача.
3. Система мотивації з використанням ігрових механік.
4. Розширена статистика та поради для поліпшення результатів.
5. Науково обґрунтована методологія з використанням нейропсихологічних принципів.

Зовнішній вигляд Peak наведено на рисунку 1.3.

Рисунок 1.3 – Застосунок Peak

Для демонстрації відмінностей між цими додатками, була створена порівняльна таблиця (таблиця 1.1).

Таблиця 1.1 — Порівняння характеристик аналогів

	CogniFit	Lumosity	Peak	Власна розробка
Адаптивний рівень складності	+	+	+	+
Режим змагань	+	-	-	+
Кросплатформеність	+	+	-	+
Відстеження прогресу	+	+	+	+
Впровадження штучного інтелекту	-	-	-	+
Сумарний коефіцієнт	4	3	2	5

Таким чином, власна розробка має найвищий сумарний коефіцієнт порівняно з аналогами. Отже, власна розробка є актуальною.

1.3 Аналіз методів розв’язання задачі

Розв’язання задачі, яка полягає у імплементації алгоритму гри для web-додатка для розвитку логічного розвитку, може бути розв’язане кількома способами.

Порівняємо кілька алгоритмів роботи штучного інтелекту, який може бути використаний для гри проти гравця (користувача додатку).

Алгоритм MiniMax [6] - це рекурсивний алгоритм, який використовується для розв'язання задач на ігрових деревах з повною інформацією та кінцевим станом, таких як шахи, хрестики-нулики та інші ігри з двома учасниками. Мета

алгоритму - знайти оптимальний хід, який максимізує шанси на перемогу для комп’ютера (ШІ).

Основні кроки алгоритму MiniMax:

1. Створюється повне ігрове дерево, в якому кожен вузол представляє можливий стан гри, а ребра - доступні ходи.

2. Термінальні вузли - це стани гри, в яких гра завершується (перемога, програш або нічия). Для цих вузлів присвоюється деяке значення, що визначає результат гри.

3. Для нетермінальних вузлів (проміжних станів гри) застосовується рекурсивна функція, яка переглядає всі можливі ходи з цього стану і оцінює потенційні результати гри.

4. У вузлах, що відповідають ходу гравця, вибирається максимальна оцінка з усіх можливих ходів (максимізація). У вузлах, що відповідають ходу суперника, вибирається мінімальна оцінка (мінімізація).

5. Після перегляду всього дерева, у кореневому вузлі (поточний стан гри) вибирається хід, який відповідає максимальній оцінці для гравця.

Таким чином, алгоритм MiniMax прогнозує всі можливі варіанти гри і вибирає оптимальний хід, який максимізує шанси на перемогу для одного з гравців.

Алгоритм альфа-бета відсікання. Цей алгоритм є вдосконаленням MiniMax і дозволяє зменшити кількість обчислень за рахунок відкидання гілок, які не матимуть впливу на остаточний результат. Однак, MiniMax є більш простим у реалізації та зрозумілішим. Крім того, для ігор з невеликою кількістю можливих ходів та станів, альфа-бета відсікання може не давати суттєвого прискорення порівняно з MiniMax.

Алгоритм Монте-Карло [7] дерева пошуку є більш складним і потужним алгоритмом, який застосовується для пошуку рішень у складних іграх з великим простором станів. Проте, його реалізація є значно складнішою, ніж MiniMax, і

вимагає більше обчислювальних ресурсів. Для простих ігор з обмеженим числом ходів, MiniMax може бути достатнім і більш ефективним рішенням.

Алгоритм MegaMiniMax є модифікацією MiniMax, яка дозволяє зменшити обчислювальну складність за рахунок об'єднання функцій мінімізації та максимізації. Однак, для простих ігор з невеликою кількістю станів, ця оптимізація може не давати суттєвого прискорення. У такому випадку, MiniMax залишається більш простим і зрозумілим рішенням.

Алгоритми, засновані на машинному навчанні можуть бути дуже потужними для складних ігор, але вимагають значних обчислювальних ресурсів та великих наборів даних для навчання. Для простих ігор з невеликим простором станів, такі підходи можуть бути надмірно складними та ресурсномісткими порівняно з MiniMax.

Не зважаючи на існування більш просунутих алгоритмів, таких як альфа-бета відсікання, алгоритм Монте-Карло або підходи на основі машинного навчання, алгоритм MiniMax залишається найкращим вибором для реалізації штучного інтелекту в простих іграх з обмеженим числом ходів та станів. Його перевагами є:

1. Достатня ефективність для ігор з невеликим простором станів.

2. Не потребує значних обчислювальних ресурсів або великих наборів даних для навчання.

3. Забезпечує гарантовано оптимальне рішення (за умови достатньої глибини пошуку).

Таким чином, для розробки алгоритму гри для web-застосунку для розвитку логічного мислення найбільш оптимальним вибором є імплементація алгоритму MinMax.

Для реалізації авторизації у WEB-додаток існує кілька способів:

1. Форма входу зі збереженням сесій. Це найпоширеніший спосіб авторизації. Користувачі вводять свої ім'я користувача та пароль, щоб увійти до

системи. Сесійні куки або токени зберігають ідентифікаційні дані на стороні сервера або клієнта для автоматичного входу під час наступних відвідувань.

2. OAuth або OpenID Connect [8]. Це протоколи, які дозволяють користувачам увійти за допомогою свого облікового запису Google, Facebook, Twitter тощо. Вони надають додатку доступ до облікових даних користувача, забезпечуючи при цьому безпеку та зручність.

3. Firebase Authentication [9] - це служба автентифікації користувачів, яка надається в межах платформи Firebase. Ця служба дозволяє розробникам підключити та використовувати авторизаційні методи та провайдери Firebase з подальшим зберіганням авторизаційних даних користувачів та перевіркою сесій на стороні Firebase, забезпечуючи безпеку та зручність для користувачів, тому що вся логіка роботи з сесіями інкапсульована на стороні Firebase.

Розробка власної форми входу зі збереженням сесій вимагає повної розробки процесу забезпечення збереження даних та створення аккаунтів розробником самостійно. Хоч і це дозволяє повністю контролювати процес роботи функціоналу, однак, може бути занадто тривалим та, окрім цього, вимагає від розробника інтеграції хмарного сервісу для роботи з базами даних для збереження даних користувачів, або ж використання окремого фізичного сервера.

OAuth або OpenID Connect полегшують задачу розробки функціоналу авторизації, однак, все ще проблемним питанням залишається збереження даних користувачів.

Сервіси Firebase же, окрім вже готової реалізації автентифікації користувачів, надає також сервіс для роботи з базами даних, тобто, також забезпечує розробника готовим хмарним сховищем для збереження даних користувачів.

Таким чином, сервіси Firebase і насамперед Firebase Authentication є найбільш оптимальним варіантом імплементації авторизації використувача у додатку.

1.4 Постановка задач дослідження

В результаті аналізу стану питання розробки алгоритмів WEB-додатків, порівняння можливостей існуючих аналогів та аналізу методів розв’язання задачі розробки алгоритмів для WEB-застосунку для розвитку логічного мислення, були визначені завдання, які будуть реалізовані:

- розробка ігрової логіки у WEB-застосунку для розвитку логічного мислення;
- реалізація алгоритму MinMax для можливості гри проти штучного інтелекту;
- інтеграція з сервісами Firebase для аутентифікації та для роботи з базою даних.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

В цьому розділі було проведено аналіз методів побудови алгоритмів веб-додатків для розвитку логічного мислення. Було проведено аналіз аналогів розроблюваного WEB-додатка, продемонстровано переваги власної розробки. Було проведено аналіз методів розв’язання задачі з реалізації алгоритмів гри. Обґрунтовано вибір алгоритму MiniMax для гри штучного інтелекту та сервіси Firebase для авторизації користувача та бази даних для збереження даних користувачів. Проведено постановку задач дослідження.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМИ

2.1 Аналіз даних

Аналіз та робота з даними є невід'ємною частиною сучасних програмних застосунків, незалежно від їхньої сфери застосування. Вони відіграють вирішальну роль у отриманні цінної інформації, прийнятті обґрунтованих рішень та вдосконаленні функціональності цих застосунків.

Наприклад, у програмах для бізнесу дані можуть використовуватися для аналізу ринку, визначення тенденцій та прогнозування майбутніх результатів. У соціальних медіа дані про користувачів, їхні вподобання та поведінку використовуються для персоналізації контенту та реклами.

У контексті алгоритмів додатку для розвитку логічного мислення, дані також відіграють ключову роль. Наприклад, для алгоритму MiniMax, на якому базується гра, використовуються такий набір даних:

1. Поточний стан ігрового поля та інформація про гравця. Це включає дані про те, хто володіє поточним ходом, рахунок у грі, розташування фігур на полі та інші параметри, що характеризують стан гри.

3. Стан гри. Алгоритм враховує, чи є поточний стан гри кінцевим. Якщо так, то він присвоює цьому стану певне значення, яке відображає результат гри (виграш, програш або нічия).

4. Глибина обстеження дерева ігрових станів. Цей параметр означає, на якій глибині знаходиться вузол, у якому відбуваються обчислення. Глибина обстеження може варіюватися в залежності від рівня складності гри, обраного гравцем.

5. Прапори MAX та MIN. Ці дані вказують, чи є поточний вузол вузлом MAX або MIN.

Ці дані необхідні для збереження історії гри, аналізу результатів та покращення майбутніх ігрових сесій.

Таким чином, обробка та аналіз даних є невід'ємною частиною роботи алгоритмів додатку для розвитку логічного мислення, що дозволяє їм приймати оптимальні рішення та забезпечувати ефективну взаємодію з гравцем.

2.2 Розробка алгоритму гри

Алгоритм гри є всебічним підходом до реалізації гри в хрестики-нулики (або аналогічної ігрової механіки) з використанням штучного інтелекту. Він охоплює всі ключові етапи процесу гри, починаючи з відображення поточного ходу гравця і закінчуючи завершенням партії.

На стадії перебігу гри алгоритм відстежує, чий зараз хід, і встановлює таймер в 30 секунд на здійснення ходу. Якщо гравець не встигає за цей час зробити свій хід, алгоритм генерує випадковий хід замість нього. Це необхідно для того, щоб створити тиск на гравця, змусити його не просто обирати найкращий хід, а і робити це швидко. До того ж, це встановлює конкретні часові рамки тривалості гри, на які користувач не може сильно вплинути. Після ходу гравця ходить штучний інтелект.

Коли гра завершується, алгоритм виконує кілька важливих дій. По-перше, він зберігає результати гри. По-друге, оновлює рейтинг гравця та додає ідентифікатор гри, у список зіграних ігор користувача. По-третє, він показує модальне вікно з результатами гри. Наприкінці алгоритм очищає ігрове поле, готуючи його до наступної партії.

Ключовим елементом алгоритму є перевірка на перемогу, яка здійснюється за класичними 8 варіантами (діагональ, колонка, рядок). Залежно від рівня складності гри (легкий, середній, складний), алгоритм присвоює різну кількість балів за перемогу, нічию чи поразку.

Крім того, алгоритм підтримує можливість здатися, що призводить до завершення гри з статусом програної партії.

Блок-схема алгоритму гри наведена на рисунку 2.1.

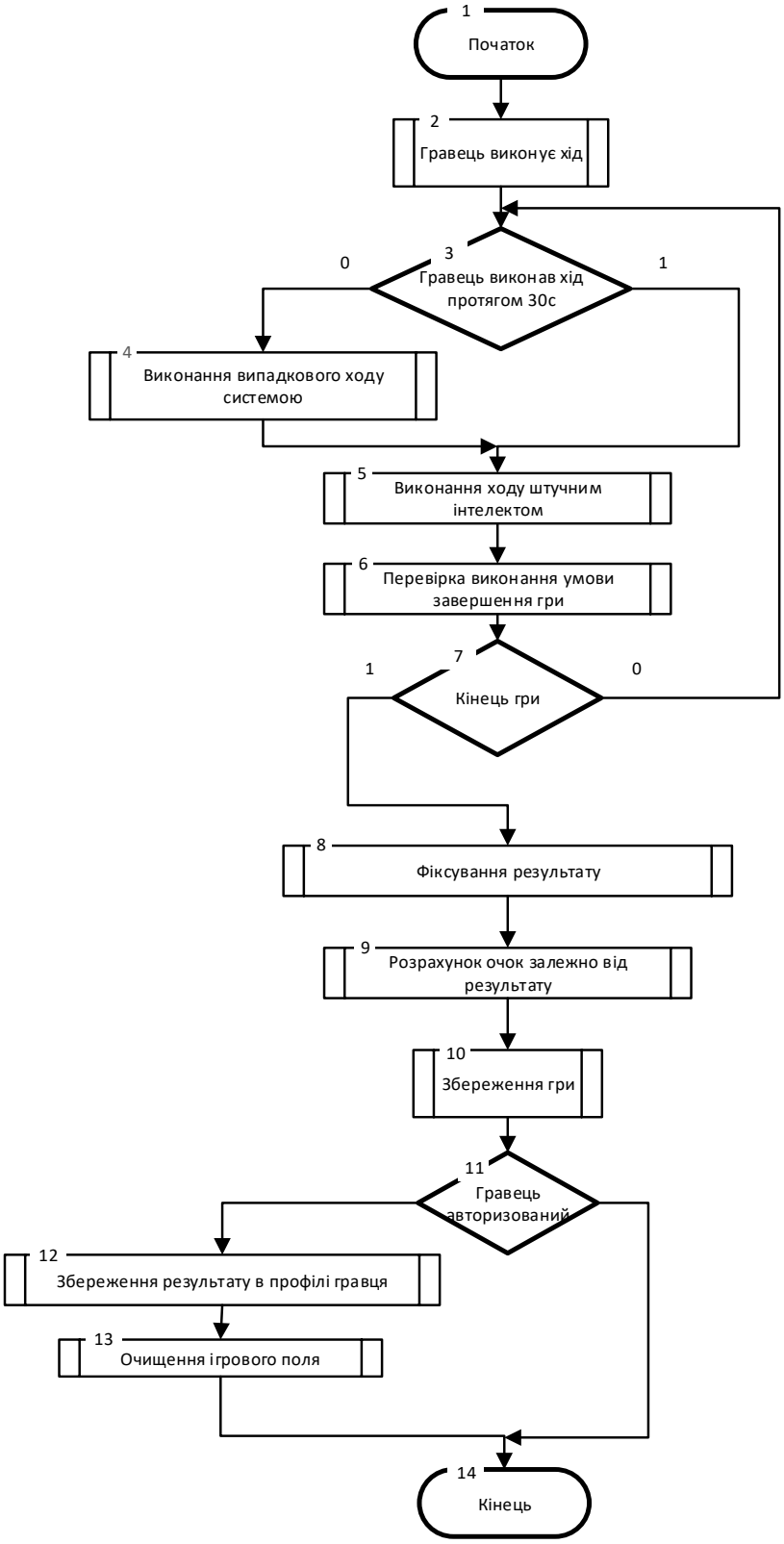


Рисунок 2.1 – Алгоритм гри проти штучного інтелекту

Розроблено також діаграму алгоритму процесу гри, блок-схему якого наведено на рисунку 2.2.

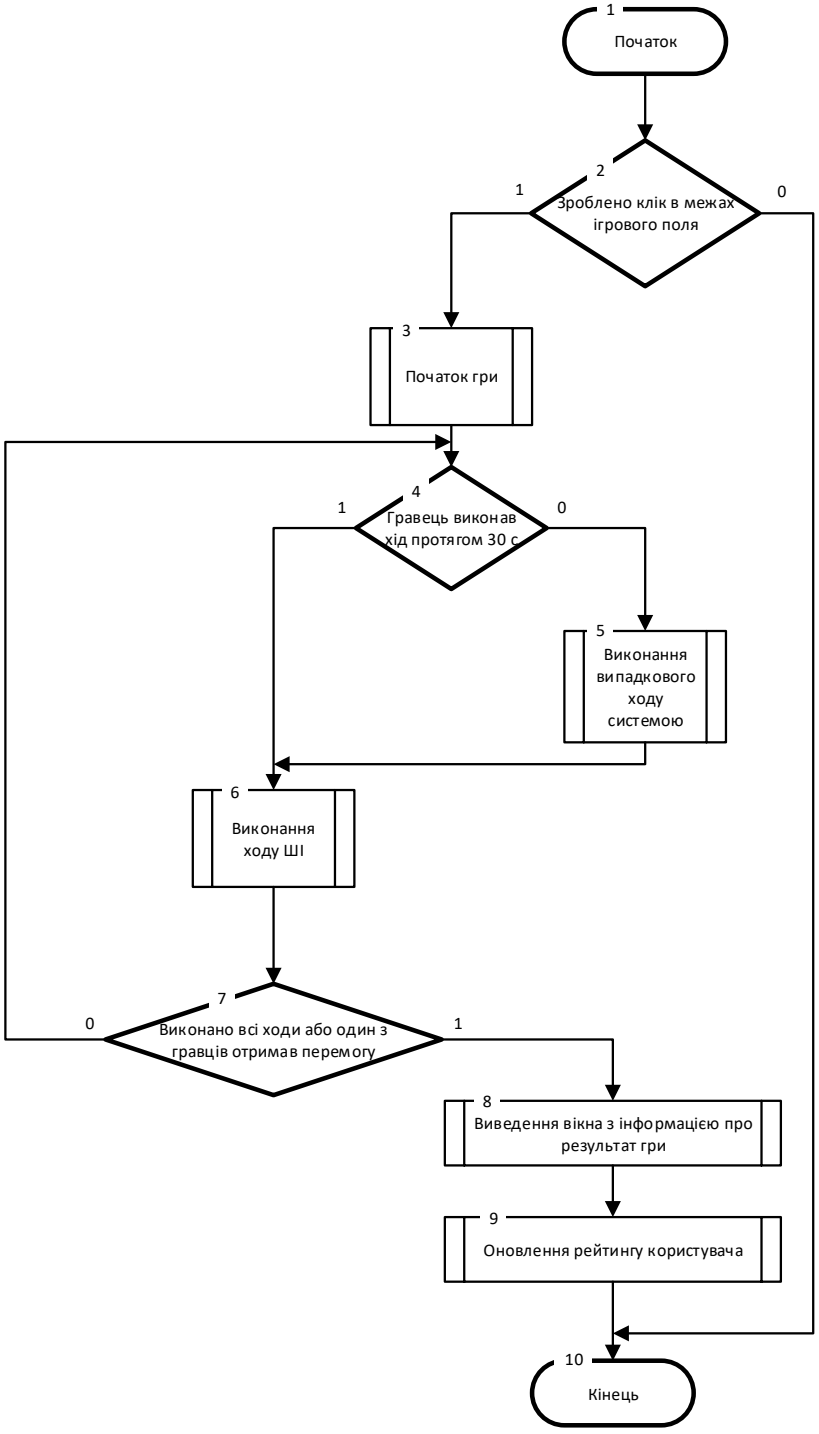


Рисунок 2.2 – Блок-схема алгориму процесу гри

Згідно з цього алгоритму, гравець розпочинає гру, натиснувши по ігровій області, після чого він здійснює перший хід. Якщо він не здійснив хід протягом 30 секунд, виконується автоматичний випадковий хід. Після цього, хід виконує штучний інтелект. Після завершення гри виводиться вікно з результатами гри. Відповідно до результатів, оновлюється рейтинг користувача.

2.3 Розробка алгоритму роботи штучного інтелекту

Алгоритм MiniMax є широко використовуваним алгоритмом прийняття рішень у теорії ігор та штучному інтелекті. Це особливо корисно для ігор з ідеальною інформацією, де всі гравці мають доступ до однакової інформації про стан гри. Алгоритм заснований на принципі мінімізації максимально можливого програшу (або максимізації мінімально можливого виграшу) для гравця, припускаючи, що суперник також грає оптимально.

Ключова ідея алгоритму MiniMax полягає в тому, щоб рекурсивно оцінити можливі ходи з поточного стану гри, а потім вибрати хід, який приведе до найкращого результату для ШІ, припускаючи, що гравець також зробить найкращі можливі ходи.

Алгоритм починається з перевірки поточної глибини дерева гри (чи глибина дорівнює 0). Якщо глибина дорівнює нулю, алгоритм повертає евристичне значення, пов’язане з поточним станом гри.

Якщо глибина не дорівнює нулю, алгоритм перевіряє, чи є поточний вузол кінцевим вузлом (тобто листовим вузлом у дереві гри). Якщо так, алгоритм повертає евристичне значення, пов’язане з поточним станом гри.

Якщо поточний вузол не є термінальним вузлом, алгоритм перевіряє, чи є він вузлом MAX, де гравець намагається максимізувати значення. Якщо так, алгоритм рекурсивно викликає алгоритм MiniMax, щоб знайти максимальне значення.

Якщо поточний вузол не є вузлом MAX, алгоритм припускає, що це вузол MIN, де опонент намагається мінімізувати значення. У цьому випадку алгоритм рекурсивно викликає алгоритм MiniMax, щоб знайти мінімальне значення.

Після знаходження мінімального або максимального значення алгоритм перевіряє, чи всі можливі ходи були виконані. Якщо ні, він відтворює можливий хід, знову викликає алгоритм MiniMax, отримує повернуте евристичне значення і скасовує зроблений хід. Потім алгоритм порівнює поточне евристичне значення

із повернутим евристичним значенням і відповідно оновлює евристичне значення.

Якщо поточний вузол є кореневим вузлом, алгоритм записує номер стовпця, який має евристичне значення. Ця інформація використовується для отримання найкращого можливого ходу у поточному стані гри.

Блок-схема алгоритму наведено на рисунку 2.3.

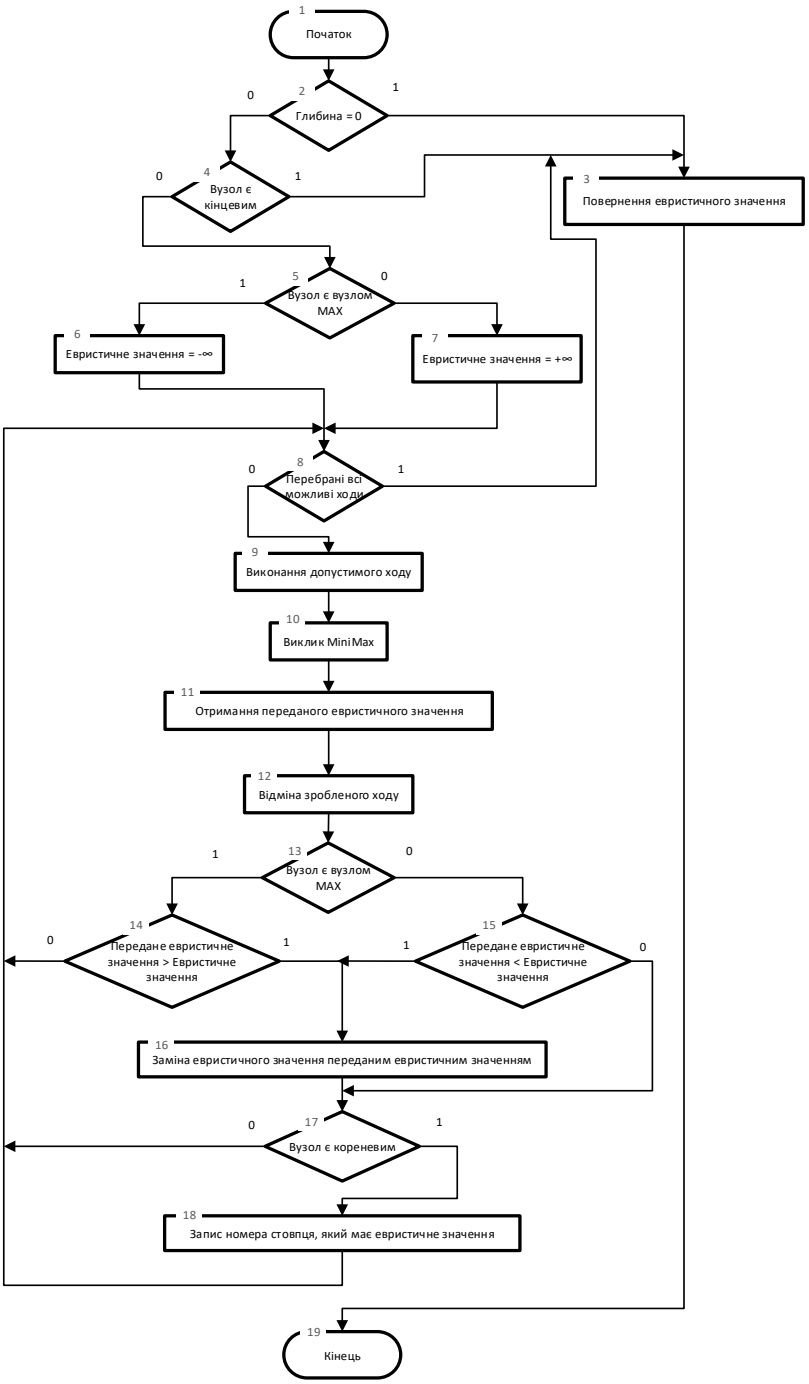


Рисунок 2.3 – Алгоритм MiniMax роботи штучного інтелекту

2.4 Розробка методу формування рейтингу користувача

Так як розробляються алгоритми гри, то необхідною є розробка системи рейтингу користувача та нарахування очок для заохочення гравців та з метою визначення успішності їхньої гри.

Для того, щоб формування рейтингу було об’єктивним, прозорим та не викликало суперечок, необхідно ретельно підійти до розробки правил формування рейтингу.

Нарахування очок залежить в першу чергу від того, як провів певну гру користувач (отримав перемогу, зіграв в нічию чи програв). Окрім цього, потрібно врахувати рівень складності, на якому він грає. На легшому рівні перемогти штучний інтелект простіше, з підвищенням складності ймовірність перемогти знижується, до того, необхідні кращі навики гри для того, аби досягти успіху.

Метод формування рейтингу користувача:

1. Визначити початковий рейтинг користувача (наприклад, 0).

2. Після завершення гри отримати такі вхідні дані:

- Результат гри (перемога, нічия, поразка).
- Рівень складності гри (легкий, середній, високий).

Згідно з отриманими вхідними даними виконати наступні дії:

3.1. Якщо результат гри - перемога:

3.1.1. Якщо рівень складності - легкий, додати 1 очко до рейтингу користувача.

3.1.2. Якщо рівень складності - середній, додати 3 очки до рейтингу користувача.

3.1.3. Якщо рівень складності - високий, додати 5 очків до рейтингу користувача.

3.2. Якщо результат гри - поразка:

3.2.1. Якщо рівень складності - легкий, відняти 1 очко від рейтингу користувача.

3.2.2. Якщо рівень складності - середній, відняти 2 очки від рейтингу користувача.

3.2.3. Якщо рівень складності - високий, відняти 1 очко від рейтингу користувача.

3.3. Якщо результат гри - нічия, не змінювати рейтинг користувача.

Оновити рейтинг користувача згідно із зміною, що відбулася на кроці 3.

Повторювати кроки 2-4 після кожної зіграної гри для постійного оновлення рейтингу користувача.

Така система очок пояснюється тим, що легкий рівень гри передбачає 33% використання алгоритму MiniMax, тобто, його доволі легко обіграти. Через це, винагорода за перемогу менша, аніж втрата очок внаслідок програшу.

На середньому рівні складності, використовується 66% алгоритму Mini-Max, що значно складніше, ніж на легкому рівні. Тому, програш очок в разі поразки такий же, однак, винагорода більша та рівна можливому програшу.

На високому рівні складності, використовується 100% алгоритму MiniMax, тобто, штучний інтелект грає на максимумі своїх можливостей. Він обирає найкращі варіанти продовження гри, тому винагорода за перемогу набагато вища – 5 очок, а от у разі програшу забирається лише 1 очко. Це пояснюється тим, що на цьому рівні складності перемога є малоймовірною, а от поразка – дуже ймовірна.

2.5 Висновки

В цьому розділі було проведено аналіз даних, що використовуються при роботі алгоритмів web-застосунку для розвитку логічного мислення, розроблено алгоритм гри, розроблено алгоритм роботи штучного інтелекту, розроблено метод формування рейтингу користувача.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

JavaScript [10], часто скорочено називається JS, є високорівневою, інтерпретованою мовою програмування, яка переважно використовується для створення вебсторінок разом з HTML та CSS. Він був створений Бренданом Ейхом у 1995 році під час його роботи в Netscape Communications Corporation.

JavaScript є прототипно-орієнтованою мовою з динамічною типізацією та підтримкою об'єктно-орієнтованого, імперативного та функціонального стилів програмування. Це означає, що ви можете використовувати JavaScript для створення складних об'єктів та функцій, які можуть взаємодіяти один з одним на вебсторінці.

Однією з ключових особливостей JavaScript є його здатність взаємодіяти з користувачем. Це дозволяє створювати динамічні вебсторінки, які реагують на дії користувача, такі як натискання кнопок, прокрутка сторінки або введення тексту. JavaScript також може взаємодіяти з HTML-елементами на сторінці, змінювати їх вміст, стилі та атрибути.

JavaScript виконується на клієнтській стороні, що означає, що код виконується безпосередньо в веббраузері користувача, а не на сервері. Це дозволяє зменшити навантаження на сервер та покращити продуктивність вебзастосунків. Однак JavaScript також може використовуватися на сервері за допомогою платформ, таких як Node.js.

JavaScript має багато бібліотек та фреймворків, які полегшують розробку вебзастосунків. Деякі з найпопулярніших включають jQuery, React, Angular та Vue.js. Ці інструменти надають готові рішення для загальних завдань, таких як маніпулювання DOM, обробка подій та управління станом застосунку.

TypeScript [11] є мовою програмування, яка була розроблена компанією Microsoft і представлена в 2012 році. Це розширення мови JavaScript, яке додає

статичну типізацію, класи, інтерфейси та інші особливості, необхідні для розробки великих і складних застосунків.

Однією з головних переваг TypeScript є статична типізація. Завдяки ній, розробник може вказати тип змінної при її оголошенні, і TypeScript перевірить, що змінна використовується правильно. Це допомагає уникнути помилок під час виконання коду і полегшує розуміння коду іншими розробниками. Крім того, TypeScript має інструменти для автозавершення коду, які допомагають швидше і точніше писати код.

TypeScript також підтримує об'єктно-орієнтоване програмування. Мова додає класи, інтерфейси, модифікатори доступу, абстрактні класи та інші особливості, які дозволяють створювати більш структурований і зрозумілий код. Завдяки цьому, розробники можуть створювати великі застосунки, які легко піддаються супроводу та розширенню.

Окрім цього, TypeScript має потужну систему модулів, яка дозволяє розбивати код на логічні блоки і легко використовувати їх в різних частинах застосунку. Це допомагає уникнути конфліктів імен і полегшує тестування та відладку коду.

TypeScript є суперсетом JavaScript, тому будь-який код, написаний на JavaScript, також буде працювати на TypeScript.

Однак, для використання всіх можливостей TypeScript, необхідно використовувати компілятор TypeScript, який перетворює код TypeScript в JavaScript. Це дозволяє використовувати нові особливості TypeScript в старих браузерах, які не підтримують нові версії JavaScript.

Отже, TypeScript є потужною мовою програмування, яка дозволяє розробникам створювати більш надійні, структуровані та зрозумілі застосунки. Завдяки статичній типізації, об'єктно-орієнтованому програмуванню та системі модулів, TypeScript є ідеальним вибором для розробки великих і складних застосунків.

Python [12] є інтерпретованою мовою програмування високого рівня, яка була створена Гвідо ван Россумом в 1991 році. Python відомий своєю простотою, читабельністю коду та широким спектром застосувань.

Однією з головних переваг Python є його простий і зрозумілий синтаксис. Python використовує відступи для групування інструкцій, що робить код більш читабельним і легким для розуміння. Крім того, Python має багату стандартну бібліотеку, яка включає в себе велику кількість модулів і функцій для роботи з файлами, мережею, базами даних та іншими ресурсами.

Python є мультипарадигмовою мовою, яка підтримує об'єктно-орієнтоване, процедурне та функціональне програмування. Це дозволяє розробникам вибирати найбільш підходящий підхід для вирішення конкретної задачі. Python також має динамічну типізацію, що означає, що тип змінної визначається автоматично під час виконання програми. Це дозволяє писати код швидше, але може призвести до помилок під час виконання програми.

Python широко використовується в багатьох сферах, включаючи веброзробку, наукові обчислення, машинне навчання, штучний інтелект, обробку даних та автоматизацію завдань. Python має велику кількість бібліотек та фреймворків, які полегшують розробку програмного забезпечення в цих сферах. Наприклад, Django та Flask використовуються для веброзробки, NumPy та Pandas використовуються для обробки даних, TensorFlow та PyTorch використовуються для машинного навчання.

Python також має велику та активну спільноту користувачів, яка створює та підтримує бібліотеки, фреймворки та інструменти для Python. Це означає, що розробники можуть знайти багато ресурсів та допомоги при роботі з Python.

Отже, Python є потужною та гнучкою мовою програмування, яка підходить для широкого спектра застосувань. Завдяки простому синтаксису, багатій стандартній бібліотеці та великій спільноті користувачів, Python є ідеальним вибором для початківців та досвідчених розробників. Проведемо порівняння цих

мов програмування, відштовхуючись від потреб розробки алгоритмів додатку для розвитку логічного мислення.

Порівняння характеристик мов програмування наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння мов програмування

Характеристика	JavaScript	TypeScript	Python
Статична типізація	-	+	+/-
Розробка веб-застосунків	+	+	+/-
Кросплатформеність	+	+	+
Керування пам’яттю	+/-	+/-	-
Багатопоточність	-	-	-
Сумарний бал	2.5	3.5	2

Таким чином, TypeScript є найкращим вибором для розробки алгоритмів веб-застосунку для розвитку логічного мислення.

Розглянемо також середовища розробки.

Visual Studio Code (VS Code) – це безкоштовний редактор коду з відкритим кодом, розроблений Microsoft. Він підходить для різних типів проектів, включаючи розробку веб-застосунків. VS Code пропонує зручний та легкий інтерфейс, який легко налаштувати відповідно до власних потреб. Він також має вбудовану підтримку популярних мов програмування, таких як JavaScript, TypeScript, CSS та HTML, що робить його ідеальним вибором для веб-розробників.

Одна з головних переваг VS Code – це його розширюваність завдяки великій кількості доступних розширень. Ці розширення додають нові функції та інструменти, покращуючи продуктивність розробки. Наприклад, розширення Emmet дозволяє швидко писати HTML та CSS код, а розширення Debugger for Chrome надає можливість налагоджувати JavaScript безпосередньо у браузері.

Крім того, є розширення для популярних фреймворків, таких як React, Angular та Vue.js.

VS Code має потужні функції для роботи з кодом, такі як інтелектуальне автодоповнення коду, рефакторинг, навігація по коду та підсвічування синтаксису. Він також підтримує вбудований термінал, що дозволяє легко запускати команди та інструменти збірки прямо з середовища розробки. Ще одна корисна функція – це вбудована підтримка Git, що полегшує роботу з системою контролю версій безпосередньо в VS Code.

Для веб-розробників VS Code пропонує вбудовану підтримку Live Server, що дозволяє швидко запускати та переглядати веб-сторінки в браузері. Крім того, він має вбудовану підтримку Node.js, що робить його ідеальним для розробки Node.js додатків. Розробники також можуть скористатися вбудованим інструментом для налагодження коду, який підтримує JavaScript, TypeScript та Node.js.

VS Code є крос-платформним рішенням, що означає, що його можна запускати на Windows, macOS та Linux. Він має синхронізацію налаштувань, що дозволяє легко переносити налаштування та розширення між різними комп'ютерами. Також VS Code підтримує інтеграцію з різними хмарними службами, такими як GitHub та Azure, що полегшує співпрацю та розгортання проектів.

Незважаючи на свою безкоштовність, VS Code пропонує великий набір функцій та можливостей, які роблять його потужним інструментом для веб-розробки. Завдяки активній спільноті розробників, VS Code постійно оновлюється та вдосконалюється, додаючи нові функції та покращуючи існуючі. Це робить його універсальним та гнучким середовищем розробки, яке можна налаштувати відповідно до будь-яких вимог веб-проекту.

WebStorm – це потужне інтегроване середовище розробки (IDE), розроблене компанією JetBrains спеціально для веб-розробників. Воно пропонує комплексний набір інструментів та функцій, які покривають весь цикл розробки

веб-застосунків, від написання коду до налагодження та розгортання. WebStorm підтримує широкий спектр веб-технологій, включаючи JavaScript, TypeScript, Angular, React, Vue.js, Node.js, HTML, CSS та багато інших.

Одна з головних переваг WebStorm – це його потужна система автодоповнення коду та рефакторингу. Інтелектуальне автодоповнення допомагає значно прискорити процес написання коду, пропонуючи відповідні варіанти у міру введення. Рефакторинг дозволяє легко реорганізовувати та оптимізувати код, економлячи час та зусилля. WebStorm також має вбудовані інструменти для перевірки коду на помилки та невідповідності стандартам кодування.

Середовище WebStorm пропонує зручний і настроюваний інтерфейс, який можна адаптувати відповідно до власних вподобань та потреб. Він підтримує різноманітні теми оформлення, налаштування гарячих клавіш та макросів, а також можливість встановлення плагінів для додавання нових функцій. Крім того, WebStorm має вбудовану підтримку популярних систем контролю версій, таких як Git, SVN та Mercurial.

Для веб-розробників WebStorm пропонує потужні інструменти для налагодження коду. Він дозволяє налагоджувати JavaScript, TypeScript та Node.js додатки, а також підтримує налагодження коду безпосередньо у браузері за допомогою вбудованого інструменту Debugger for Chrome. Крім того, WebStorm має вбудовану підтримку популярних фреймворків, таких як React, Angular та Vue.js, що полегшує роботу з ними.

Ще однією перевагою WebStorm є його інтеграція з інструментами збірки та автоматизації. Він підтримує Webpack, Gulp, Grunt та інші популярні інструменти, що дозволяє легко налаштовувати та керувати процесом збірки проектів. WebStorm також пропонує зручні інструменти для тестування, такі як вбудовані фреймворки для тестування JavaScript та можливість налагодження тестів.

Незважаючи на те, що WebStorm є платним рішенням, JetBrains пропонує безкоштовні ліцензії для студентів, викладачів та відкритих проектів з відкритим кодом. Крім того, компанія регулярно випускає оновлення та нові версії WebStorm, додаючи нові функції та покращуючи існуючі, що робить його надійним та сучасним середовищем розробки для веб-застосунків.

Atom – це безкоштовний редактор коду з відкритим кодом, розроблений GitHub. Він спеціально створений для веб-розробників і пропонує зручне та гнучке середовище для роботи з різними веб-технологіями, такими як HTML, CSS, JavaScript, TypeScript, Node.js та багато інших. Atom має простий та інтуїтивний інтерфейс, який легко налаштувати під свої потреби.

Одна з головних переваг Atom – це його система пакетів та плагінів, створена спільнотою розробників. Ця система дозволяє розширювати функціональність Atom відповідно до власних вимог, додаючи нові інструменти, теми оформлення, автодоповнення коду та багато іншого. Наприклад, є пакети для підтримки популярних фреймворків, таких як React, Angular та Vue.js, а також інструменти для налагодження коду та інтеграції з різними системами контролю версій.

Atom пропонує низку зручних функцій для веб-розробників, серед яких підсвічування синтаксису, автодоповнення коду, рефакторинг, навігація по коду та перегляд файлів у дереві проекту. Він також має вбудовану підтримку Git та можливість одночасної роботи з кількома вікнами та панелями, що полегшує роботу з великими проектами.

Для розробки веб-застосунків Atom пропонує вбудований пакет atom-live-server, який дозволяє швидко запускати та переглядати веб-сторінки прямо в середовищі редактора. Він також має пакети для налагодження JavaScript та Node.js додатків, що полегшує процес тестування та відлагодження коду.

Atom є крос-платформним рішенням, сумісним з операційними системами Windows, macOS та Linux.

Він підтримує синхронізацію налаштувань між різними пристроями, що дозволяє легко переносити налаштування та встановлені пакети на інші комп'ютери. Крім того, Atom має вбудовану підтримку різних кодувань символів, що корисно для роботи з міжнародними проектами.

Незважаючи на те, що Atom не такий потужний та функціональний, як деякі інші середовища розробки, він є гарним вибором для веб-розробників, які шукають простий, але водночас гнучкий і настроюваний редактор коду.

Сформуємо таблицю порівняння характеристик описаних середовищ розробки (таблиця 3.2).

Таблиця 3.2 – Порівняння характеристик середовищ розробки

Характеристика	Visual Studio Code	Webstorm	Atom
Безкоштовний доступ	+	-	+
Крос-платформеність	+	+	+
Вбудована підтримка Git	+	+	-
Автодоповнення коду	+	+	+
Розширюваність з використанням плагінів	+	+	+
Сумарний бал	5	4	4

Таким чином, Visual Studio Code випереджає за характеристиками Web-storm та Atom на 20% (100% - 4/5*100% = 20%). Саме тому було прийнято рішення використовувати для розробки саме Visual Studio Code.

3.2 Інтеграція сервісів Firebase

Сервіси Firebase [13] є потужною хмарною платформою, розробленою Google, яка допомагає створювати та масштабувати веб-додатки.

Firebase надає багато інструментів та функцій для різних аспектів розробки додатків, таких як бази даних, хостинг, авторизація, аналітика, повідомлення

тощо. Це дозволяє розробникам зосередитися на логіці додатка, а не на управлінні серверною інфраструктурою.

Одна з ключових функцій Firebase - це Firebase Realtime Database та Cloud Firestore. Це хмарні NoSQL бази даних, які забезпечують швидку синхронізацію даних у реальному часі між клієнтами та сервером. Вони пропонують високу доступність, масштабованість та автоматичне оновлення даних на всіх підключених клієнтах.

Firebase Authentication забезпечує легку інтеграцію різних методів автентифікації, таких як електронна пошта/пароль, Google, Facebook, Twitter, тощо. Це спрощує процес реєстрації та входу користувачів у ваш додаток.

Firebase Hosting [14] - це хостингова платформа для статичних веб-сайтів та односторінкових додатків (SPA). Вона забезпечує безпечне та швидке розгортання веб-додатків із глобальною CDN.

Firebase пропонує безкоштовний стартовий план із щедрими квотами, що дозволяє спочатку безкоштовно протестувати більшість функцій. Існують також платні плани для додатків із високим навантаженням.

Для інтеграції сервісів Firebase, спершу, зареєструємо на сайті Firebase проект, який згодом буде інтегровано із застосунком.

Спершу, налаштуємо сервіси реєстрації та авторизації. Firebase пропонує широкий набір варіантів реєстрації та авторизації (рисунок 3.1).

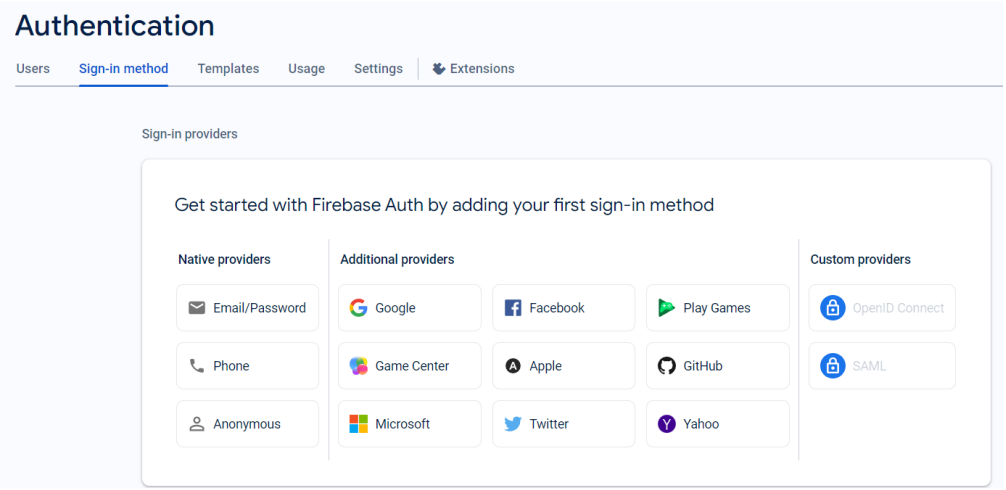


Рисунок 3.1 – Перелік методів реєстрації Firebase

Оберемо авторизацію через Google. Спершу, для цього необхідно додати назву проекту та електронну адресу розробника, яка буде закріплена за проектом (рисунок 3.2).

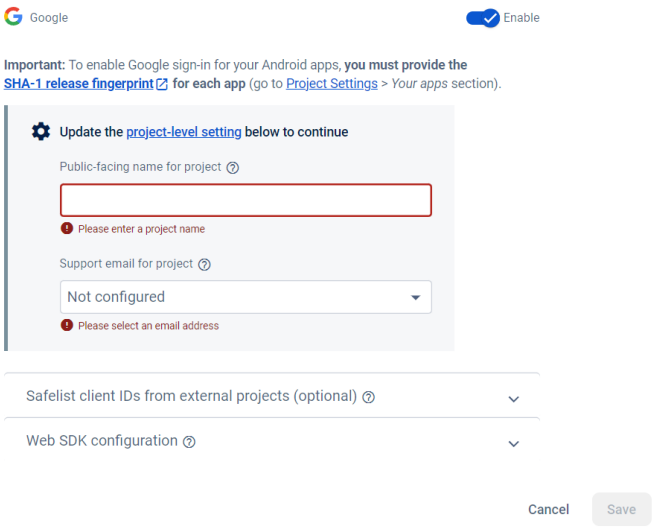


Рисунок 3.2 – Налаштування авторизації Google

Також додамо авторизацію з використанням електронної пошти та пароля. Це не потребує додаткових налаштувань.

Додані методи авторизації зображено на рисунку 3.3.

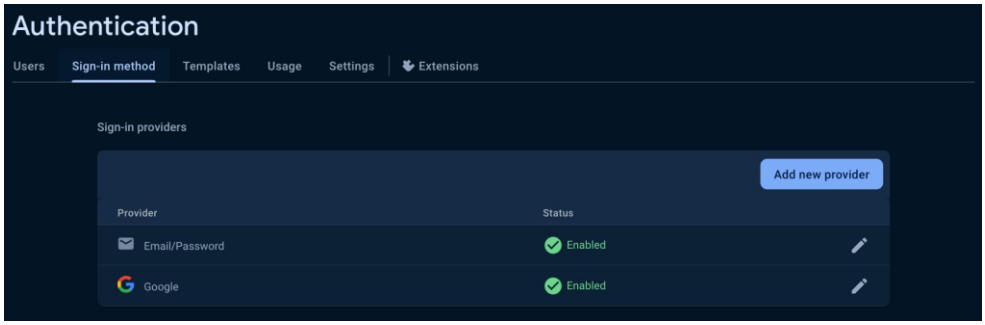


Рисунок 3.3 – Додані методи авторизації

Додамо також базу даних Firebase. Вікно реєстрації бази даних зображено на рисунку 3.4. Оберемо регіон europe-west3, сервери якого знаходяться

найближче до України з усіх можливих. Такий вибір забезпечує максимально швидке з’єднання та обмін даними.

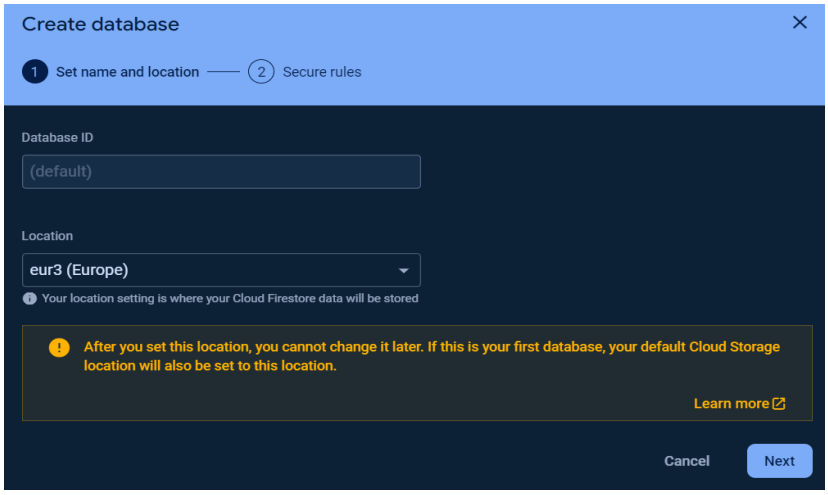


Рисунок 3.4 – Інтеграція бази даних Firebase

Firebase дозволяє переглядати історію релізів застосунка. Адмінська панель релізів зображена на рисунку 3.5.

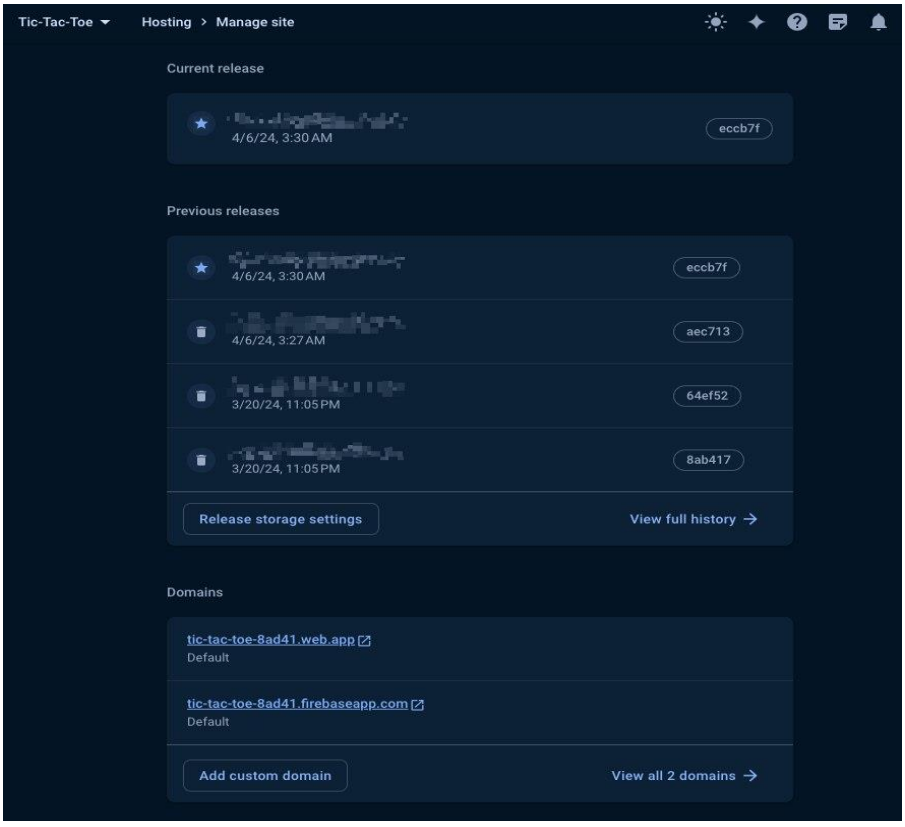


Рисунок 3.5 – Історія релізів застосунку в Firebase

Історія релізів відображає різні версії застосунка, що були опубліковані. Якщо в новому релізі виникають критичні помилки або проблеми, можна легко повернутися до попередньої стабільної версії, використовуючи цю інформацію. Це допомагає мінімізувати простої додатку та негативний вплив на користувачів.

Вона також дозволяє відстежити зміни між різними версіями, що полегшує процес аналізу та налагодження проблем. Це дозволяє порівняти коди різних релізів, щоб знайти походження проблеми. Це форма контролю версій і дозволяє відстежувати зміни в коді, ресурсах та конфігураціях з часом. Це корисно для розуміння, які зміни були внесені в певній версії та коли вони були додані.

Наявність такої історії випусків застосунка може полегшити комунікацію та співпрацю в команді розробників, надаючи інформацію про те, які функції були додані, які помилки були виправлені та які зміни були внесені в кожному релізі.Firebase містить вбудовану аналітику використання застосунка та його компонентів. На рисунку 3.6 продемонстровано аналітику активності користувачів застосунку.

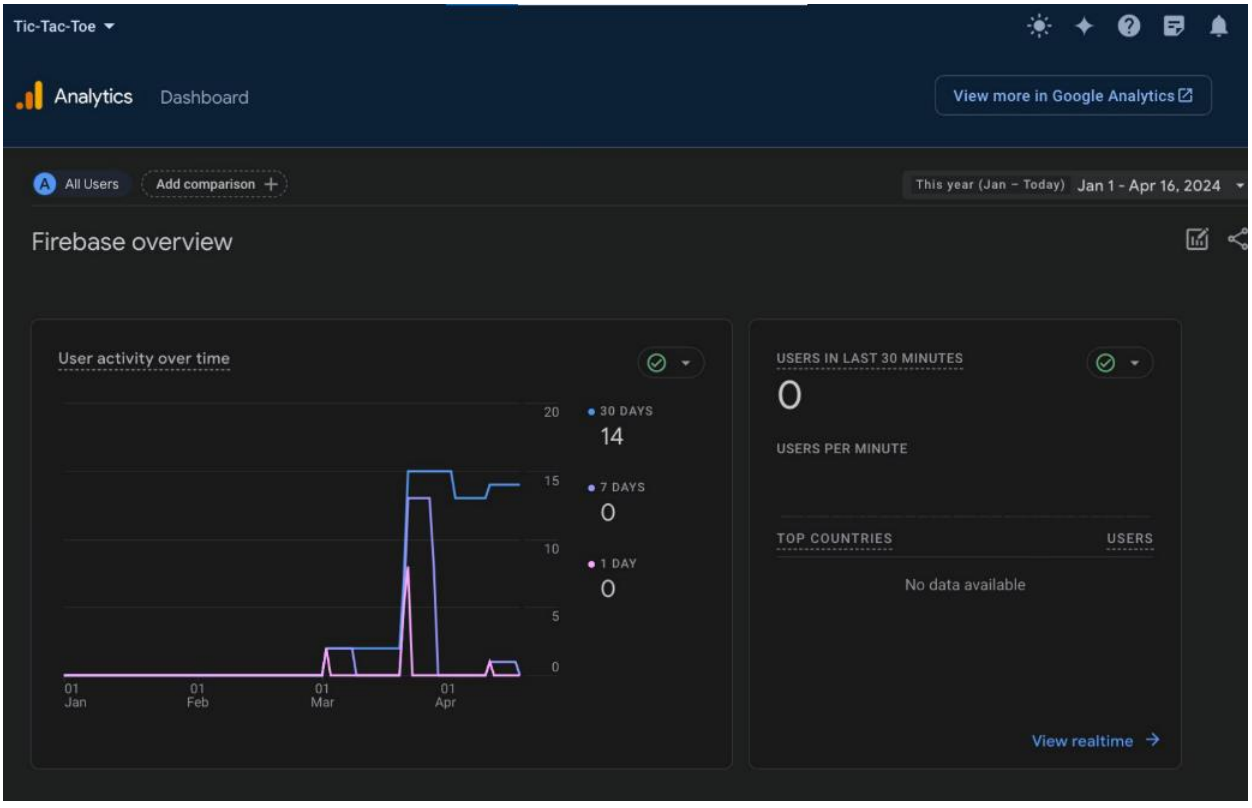


Рисунок 3.6 – Аналітика активності користувачів в застосунку

На рисунку 3.7 зображено статистику середнього часу перебування користувачів в застосунку. Середній час перебування користувача в застосунку складає 1 хвилину 3 секунди. Такі дані були отримані під час тестування додатку.

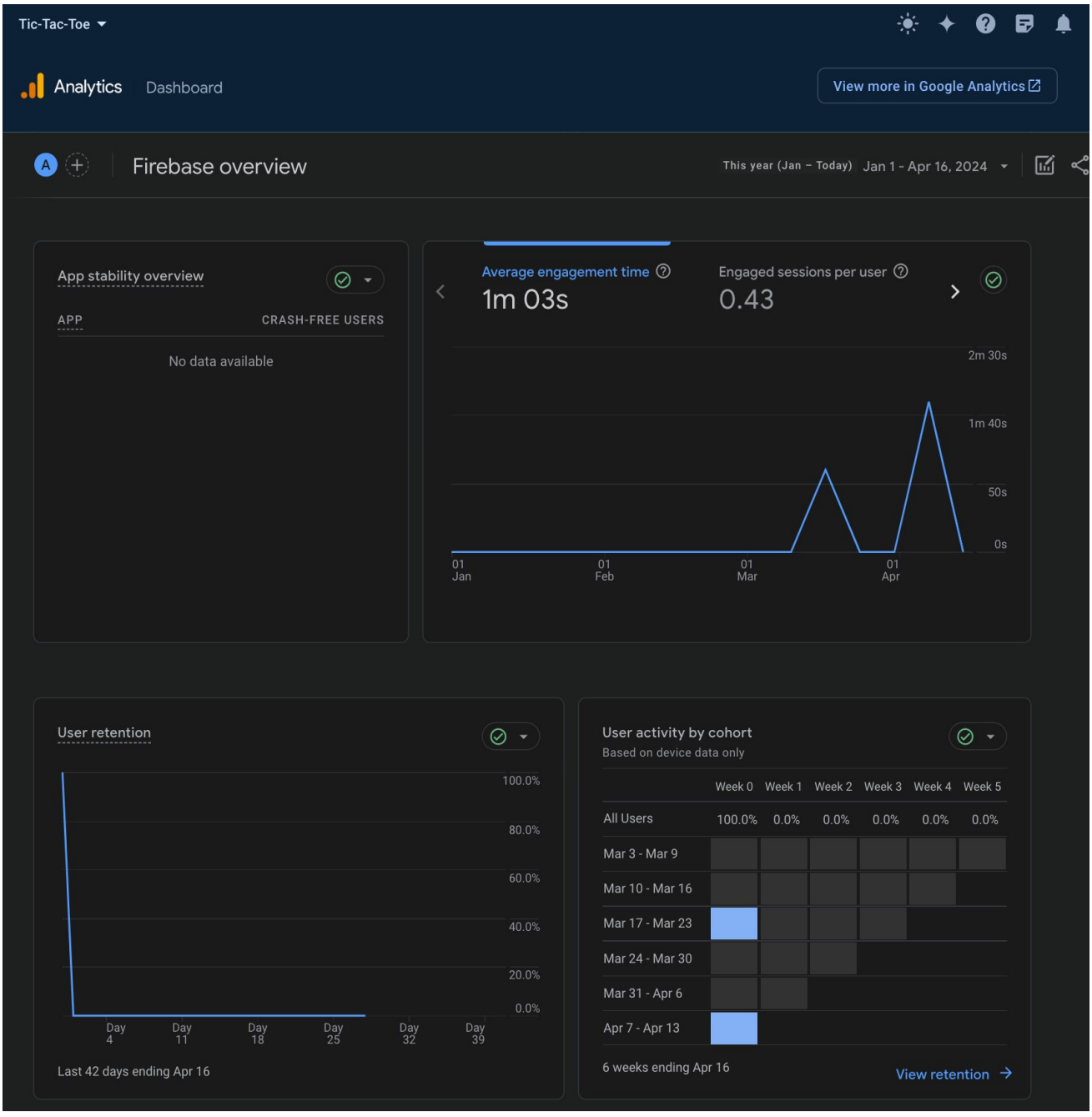


Рисунок 3.7 – Статистика перебування користувачів в застосунку

Аналітика по країнам користувачів затосунку продемонстрована на рисунку 3.8. Зафіксовано 14 користувачів з України та 1 з Великобританії.

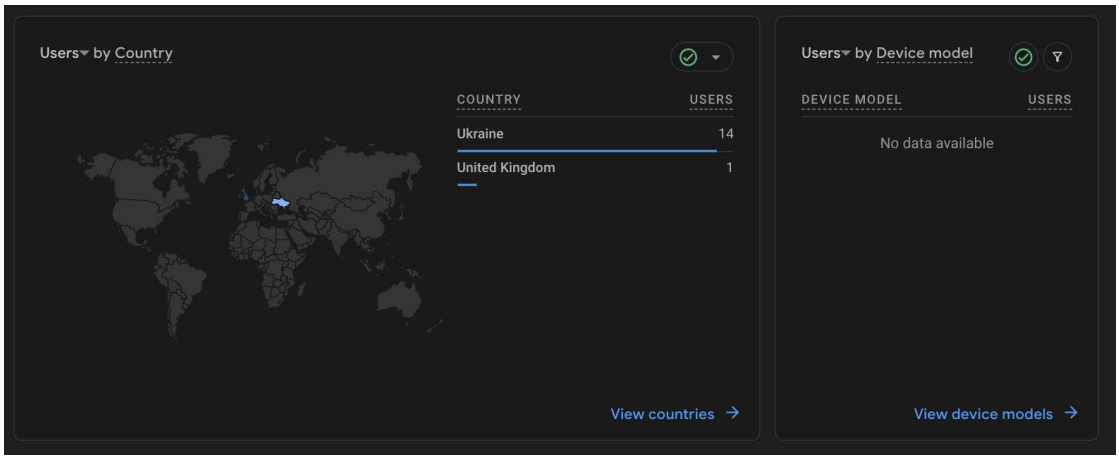


Рисунок 3.8 – Аналітика по країнам користувачів

Firebase також надає аналітику використання бази даних. Її подано на рисунку 3.9. На ній ми бачимо, що було здійснено 105 випадків читання бази даних та 29 випадків запису даних у неї.

Рисунок 3.9 – Аналітика використання бази даних

3.3 Розробка модуля взаємодії з базою даних

Було розроблено функціонал для взаємодії з базою даних. Діаграма потоку даних методу отримання користувача зображено на рисунку 3.10.

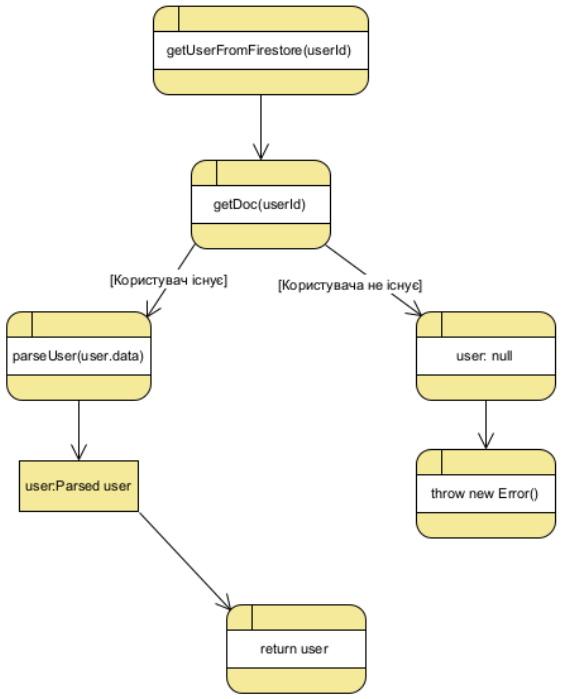


Рисунок 3.10 – Діаграма потоку даних для отримання користувача

Функція `getUserFromFirestore` отримує документ користувача з бази даних Firestore на основі наданого `userId`:

1. Вона імпортує необхідні функції (`getDoc`, `doc`) з бібліотеки `firebase/firestore` для взаємодії з Firestore.

2. Після цього, імпортує об'єкт `firestore` та `userSchema` з інших файлів у проекті.

3. Функція `getUserFromFirestore` є асинхронною функцією, яка приймає `userId` типу рядок як аргумент.

4. Всередині функції, спочатку вона намагається отримати документ користувача з колекції "users" у Firestore, використовуючи функції `getDoc` та `doc`, передаючи `userId` як параметр.

5. Потім вона перевіряє, чи існує отриманий документ користувача, використовуючи метод `exists()`.

6. Якщо документ користувача існує, вона обробляє дані користувача за допомогою функції `userSchema.parse` і повертає об'єкт, що містить `parsedUser`.

7. Якщо документ користувача не існує, вона повертає об'єкт з `user`, встановленим на `null`.

8. Якщо виникає помилка під час процесу, функція генерує виняток з повідомленням "Failed to get user from firestore".

Ця функція забезпечує спосіб отримати документ користувача з Firestore на основі наданого `userId`, обробити дані користувача за допомогою zod схеми та повернути оброблений об'єкт з даними користувача або обробити випадок, коли користувач не існує у базі даних або коли повернута з бази даних структура не пройшла перевірки схемою, тобто некоректна для роботи застосунку.

Розроблено також метод отримання ігор користувача. Діаграма потоку даних цього методу зображена на рисунку 3.11.

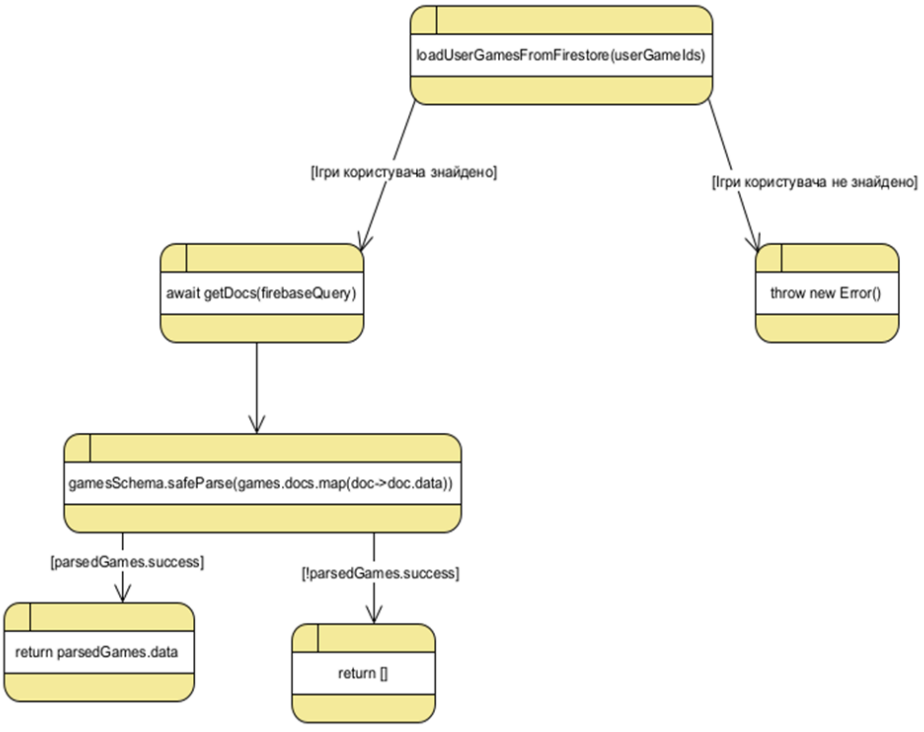


Рисунок 3.11 – Отримання ігор користувача

Функція loadUserGamesFromFirestore завантажує ігри користувача з бази даних Firestore на основі наданих userGamesIds:

1. Вона імпортує необхідні функції (query, collection, where, orderBy, getDocs) з бібліотеки firebase/firestore для взаємодії з Firestore.
2. Вона імпортує об'єкт firestore та gamesSchema з інших файлів у проекті.
3. Функція loadUserGamesFromFirestore є асинхронною функцією, яка приймає масив userGamesIds типу рядок як аргумент.
4. Всередині функції вона створює запит до Firestore, використовуючи функції query, collection, orderBy та where. Вона запитує колекцію "games" у Firestore. Сортує результати за спаданням поля "datetime". Фільтрує результати, де поле "id" знаходиться в переданому масиві userGamesIds.
5. Потім вона отримує документи, що відповідають запиту, використовуючи функцію getDocs.
6. Вона обробляє отримані документи за допомогою функції gamesSchema.safeParse, перетворюючи дані документів на відповідні об'єкти гри.
7. Якщо обробка успішна (parsedGames.success), вона повертає масив оброблених ігор (parsedGames.data).
8. Якщо обробка невдала, вона повертає порожній масив.
9. У разі виникнення помилки під час процесу вона виводить помилку в консоль і генерує виняток з повідомленням "Failed to get user games place from firestore".

Ця функція забезпечує спосіб завантажити ігри користувача з Firestore на основі переданих userGamesIds, відфільтрувати, відсортувати та обробити отримані дані за допомогою схеми gamesSchema.

Вона дозволяє отримувати ігри, пов'язані з конкретними ідентифікаторами, і обробляти їх відповідним чином.

Розроблено також функцію збереження гри в базу даних. Її діаграма потоку даних зображена на рисунку 3.12.

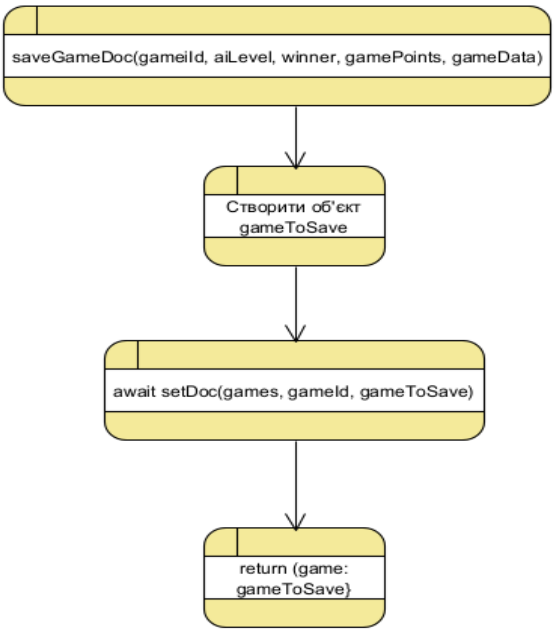


Рисунок 3.12 – Процес збереження гри в базу даних

Цей код є асинхронною функцією TypeScript saveGameDoc, яка зберігає дані гри в колекції "games" у Firestore. Ось опис того, що вона робить:

1. Вона імпортує необхідні функції (setDoc, doc) з бібліотеки firebase/firestore для взаємодії з Firestore.

2. Вона імпортує об'єкт firestore та різні типи даних (CellValueType, GameCellType, GameFullType, AiLevelType) з інших файлів проекту.

3. Функція saveGameDoc приймає об'єкт з такими властивостями: gameId, aiLevel, winner, gamePoints, gameData.

4. Всередині функції вона створює об'єкт gameToSave типу GameFullType, який містить всі необхідні дані для збереження гри, такі як id, datetime, aiLevel, winner, gamePoints та cells (дані гри).

5. Далі вона використовує функцію setDoc з Firestore для збереження gameToSave у колекції "games", використовуючи gameId як ідентифікатор.

6. Якщо операція успішна, вона повертає об'єкт, що містить збережену гру.

7. Якщо виникає помилка, вона генерує виняток з поточною помилкою.

Ця функція дозволяє зберігати повні дані гри в колекції "games" у Firestore. Вона приймає всі необхідні дані для гри, створює відповідний об'єкт

GameFullType, який містить ці дані, і зберігає його в Firestore з використанням унікального gameId як ідентифікатора документа. Вона також обробляє випадки успішного збереження та виникнення помилок під час операції.

Задля збереження гри в базу даних, окрім власне функціоналу збереження, потрібно також передбачити випадки, коли це потрібно робити і за яких умов.

Діаграма потоку даних функції збереження гри подана на рисунку 3.13.

Це функція зворотнього виклику onGameEnd, яка запускається після завершення гри. Вона виконує кілька завдань, пов'язаних із збереженням даних гри та оновленням інформації про користувача.

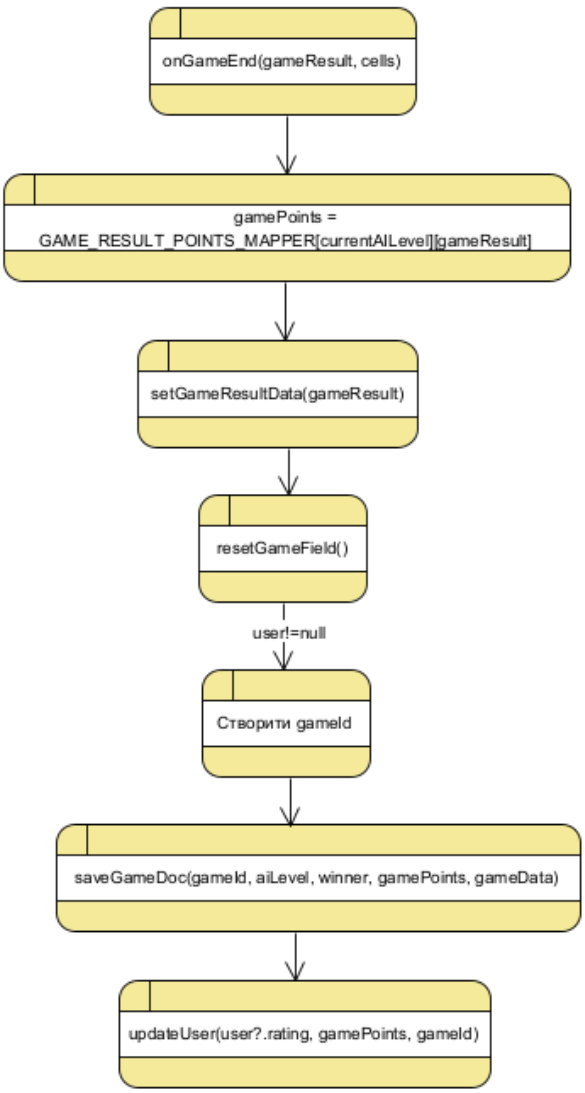


Рисунок 3.13 – Виклик процесу збереження гри

Вона приймає два аргументи: gameResult (типу GameResultType) і клітинки (масив GameCellType).

Він розраховує ігрові очки на основі поточного рівня штучного інтелекту currentAiLevel і gameResult за допомогою GAME_RESULT_POINTS_MAPPER.

Він викликає функцію setGameResultData з gameResult, щоб оновити дані результатів гри.

Він викликає функцію resetGameField, щоб скинути ігрове поле.

Якщо користувач не є нульовим (тобто користувач увійшов у систему), він генерує унікальний gameId за допомогою Math.random().toString(36).slice(2).

Він викликає функцію saveGameDoc із gameId, клітинками, currentAiLevel, gamePoints і переможцем (визначається на основі gameResult і того, виграно чи програно гру).

Він викликає функцію updateUser із рейтингом користувача (оновленим на основі GamePoints), масивом ігор користувача (додаючи gameId) та gameId.

Він повертає об’єкт, що містить savedGame і updatedUser.

Якщо під час процесу виникає помилка, вона реєструється з повідомленням «помилка збереження гри».

Код також імпортує кілька змінних і функцій з інших файлів, таких як currentAiLevel, resetGameField, setGameResultData, updateUser, user, games і rating.

Отже, цей код обробляє кінець гри, зберігає дані гри у Firestore, оновлює інформацію про користувача (рейтинг та історію гри) і повертає збережену гру й оновлені дані користувача. Під час виконання цих операцій враховується, чи ввійшов користувач у систему чи ні.

3.4 Висновки

В цьому розділі було проведено варіантний аналіз та здійснено вибір мови програмування розробки, аргументовано такий вибір. Було проведено інтеграцію сервісів Firebase в застосунок, розроблено алгоритми та функціонал гри, впроваджено базу даних.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз методів тестування

Тестування алгоритмів гри для розвитку логічного мислення. Методи тестування алгоритмів гри включають в себе перевірку правильності роботи алгоритму, тестування на різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках [17].

Перевірка правильності роботи алгоритму. Цей метод передбачає перевірку правильності роботи алгоритму на різних вхідних даних. Це може бути здійснено шляхом порівняння результатів роботи алгоритму з відомими правильними результатами.

Тестування на різних вхідних даних. Цей метод передбачає тестування алгоритму на різних вхідних даних, включаючи як типові, так і нетипові дані. Це дозволяє перевірити, чи працює алгоритм правильно в усіх випадках.

Аналіз часу виконання та пам'яті. Цей метод передбачає аналіз часу виконання та пам'яті, необхідної для роботи алгоритму. Це дозволяє визначити, чи працює алгоритм ефективно з точки зору використання ресурсів.

Тестування на граничних випадках. Цей метод передбачає тестування алгоритму на граничних випадках, тобто на вхідних даних, що знаходяться на межі допустимих значень. Це дозволяє перевірити, чи працює алгоритм правильно в крайніх випадках.

Тестування сервісів Firebase. Firebase - це платформа для розробки мобільних та вебдодатків від Google. Firebase пропонує широкий спектр сервісів, включаючи базу даних, аутентифікацію, зберігання файлів, відправку сповіщень та інше.

Тестування бази даних Firebase. База даних Firebase - це NoSQL база даних, яка підтримує синхронізацію даних в реальному часі. Тестування бази даних Firebase включає в себе перевірку правильності роботи бази даних, тестування на

різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках.

Тестування аутентифікації Firebase. Firebase Автентифікація - це сервіс, який дозволяє користувачам авторизуватися в додатку за допомогою різних провайдерів, таких як Google, Facebook, Twitter та інші. Тестування аутентифікації Firebase включає в себе перевірку правильності роботи сервісу, тестування на різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках.

Тестування зберігання файлів Firebase. Firebase Storage - це сервіс, який дозволяє зберігати та завантажувати файли в хмарному сховищі. Тестування зберігання файлів Firebase включає в себе перевірку правильності роботи сервісу, тестування на різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках.

Тестування інтеграції з базою даних. Тестування інтеграції з базою даних включає в себе перевірку правильності роботи інтеграції, тестування на різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках [18].

Перевірка правильності роботи інтеграції. Цей метод передбачає перевірку правильності роботи інтеграції з базою даних. Це може бути здійснено шляхом порівняння результатів роботи інтеграції з відомими правильними результатами.

Тестування на різних вхідних даних. Цей метод передбачає тестування інтеграції з базою даних на різних вхідних даних, включаючи як типові, так і нетипові дані. Це дозволяє перевірити, чи працює інтеграція правильно в усіх випадках.

Аналіз часу виконання та пам'яті. Цей метод передбачає аналіз часу виконання та пам'яті, необхідної для роботи інтеграції з базою даних. Це дозволяє визначити, чи працює інтеграція ефективно з точки зору використання ресурсів.

Тестування на граничних випадках. Цей метод передбачає тестування інтеграції з базою даних на граничних випадках, тобто на вхідних даних, що

знаходяться на межі допустимих значень. Це дозволяє перевірити, чи працює інтеграція правильно в крайніх випадках.

Висновок. Таким чином, тестування алгоритмів гри для розвитку логічного мислення, сервісів Firebase та інтеграції з базою даних є важливою частиною розробки додатків. Для проведення тестування необхідно використовувати різні методи, включаючи перевірку правильності роботи, тестування на різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках.

Переваги тестування. Тестування дозволяє виявити помилки та недоліки в роботі алгоритмів, сервісів та інтеграції з базою даних, а також перевірити, чи працюють вони ефективно з точки зору використання ресурсів. Це дозволяє поліпшити якість додатку та забезпечити кращий досвід користувача.

Недоліки тестування. Недоліками тестування є те, що воно може бути часозатратним та вимагати значних ресурсів. Крім того, тестування може не виявити всі помилки та недоліки, особливо якщо вони виникають в нетипових умовах.

Рівні тестування. Тестування може бути проведено на різних рівнях, включаючи модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування.

Модульне тестування. Модульне тестування передбачає тестування окремих модулів або компонентів додатку. Це дозволяє виявити помилки та недоліки в роботі окремих модулів та поліпшити їхню якість.

Інтеграційне тестування. Інтеграційне тестування передбачає тестування взаємодії між модулями або компонентами додатку. Це дозволяє виявити помилки та недоліки в роботі інтеграції між модулями та поліпшити їхню взаємодію.

Приймальне тестування. Приймальне тестування передбачає тестування додатку користувачем або замовником. Це дозволяє перевірити, чи відповідає

додаток вимогам користувача та чи працює він належним чином в умовах реального використання.

Автоматизоване тестування. Автоматизоване тестування передбачає використання спеціальних програмних засобів для автоматизації процесу тестування. Це дозволяє зберегти час та ресурси, а також підвищити ефективність тестування.

Ручне тестування. Ручне тестування передбачає виконання тестування вручну, без використання спеціальних програмних засобів. Це дозволяє перевірити додаток в умовах, які найбільш наближені до реального використання.

Регресійне тестування. Регресійне тестування передбачає повторне тестування додатку після внесення змін в нього. Це дозволяє перевірити, чи не виникли нові помилки та недоліки в роботі додатку в результаті внесених змін.

Навантажувальне тестування. Навантажувальне тестування передбачає тестування додатку під високим навантаженням. Це дозволяє перевірити, чи працює додаток належним чином в умовах високого навантаження та чи може він обробляти велику кількість даних.

Стресове тестування. Стресове тестування передбачає тестування додатку в умовах, які перевищують його нормальні робочі умови. Це дозволяє перевірити, чи працює додаток належним чином в умовах екстремального навантаження та чи може він витримати високі навантаження.

Тестування продуктивності. Тестування продуктивності передбачає тестування додатку з метою визначення його продуктивності. Це дозволяє визначити, чи працює додаток ефективно з точки зору використання ресурсів та чи може він обробляти велику кількість даних в короткий час.

Таким чином, тестування алгоритмів гри для розвитку логічного мислення, сервісів Firebase та інтеграції з базою даних є важливою частиною розробки додатків. Для проведення тестування необхідно використовувати різні методи, включаючи перевірку правильності роботи, тестування на різних вхідних даних, аналіз часу виконання та пам'яті, а також тестування на граничних випадках.

Тестування може бути проведено на різних рівнях, включаючи модульне тестування, інтеграційне тестування, системне тестування та приймальне тестування. Крім того, можна використовувати автоматизоване та ручне тестування, регресійне тестування, навантажувальне тестування, стресове тестування та тестування продуктивності.

4.2 Тестування розробленого функціоналу

Для тестування сервісів Firebase, перевіримо створення нового аккаунту, сесій авторизації та збереження ігор в базу даних.

База даних авторизацій користувачів продемонстрована на рисунку 4.1.

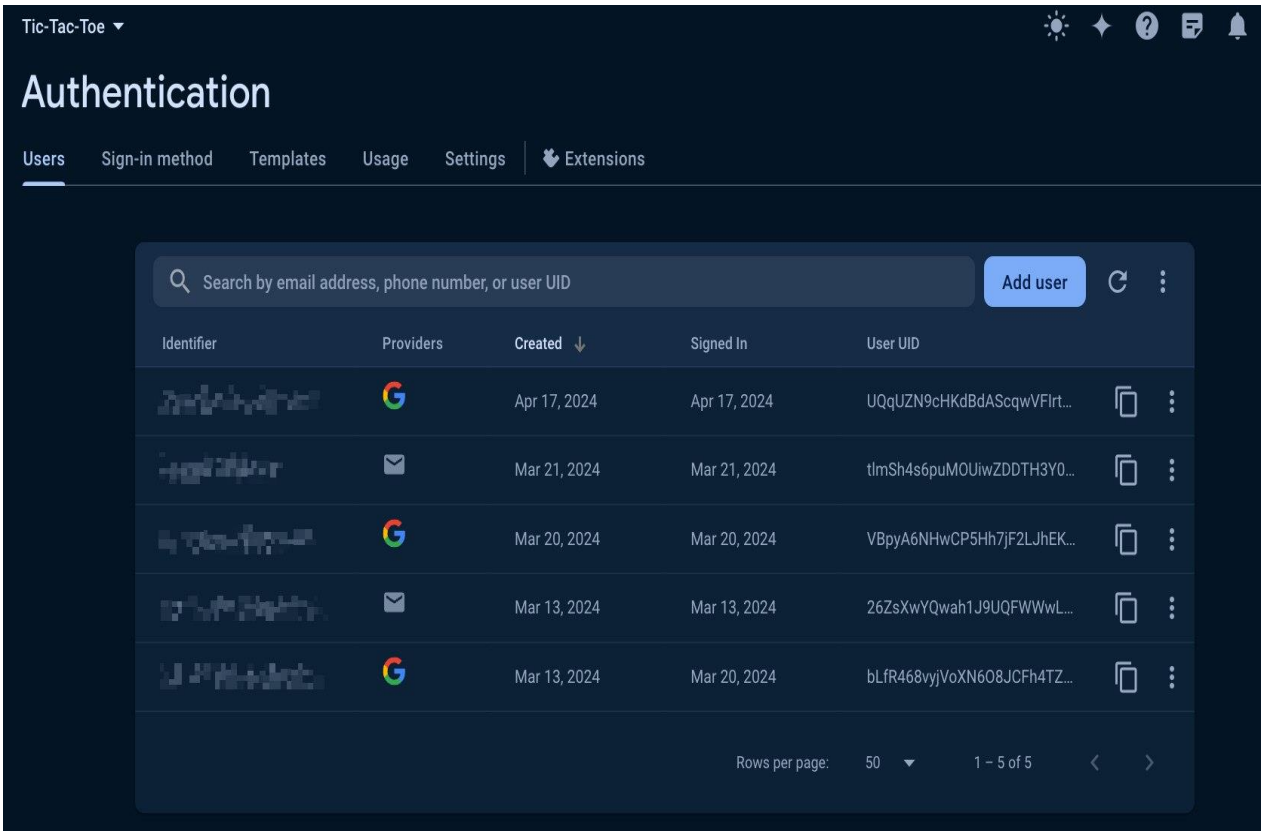


Рисунок 4.1 – База даних авторизацій користувачів

Після деякого часу використання застосунку користувачами, з’явилися 2 нові авторизації, які відмічені на рисунку 4.2.

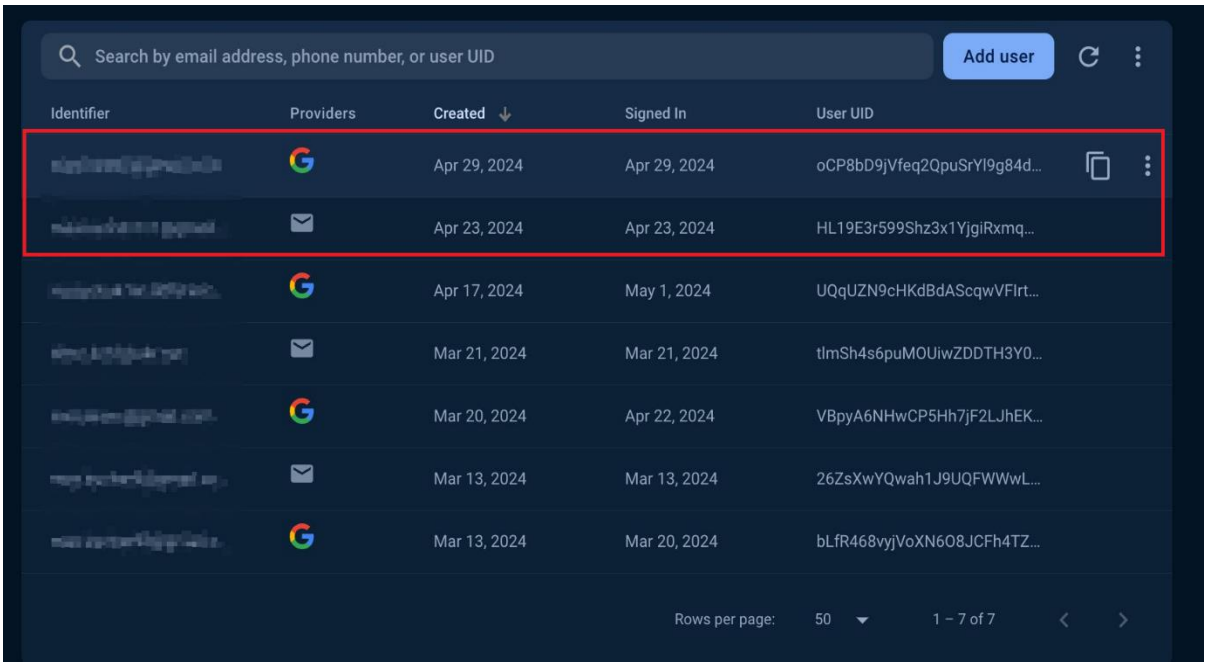


Рисунок 4.2 – Нові авторизації користувачів в базі даних

База даних з колекцією ігор зображена на рисунку 4.3.

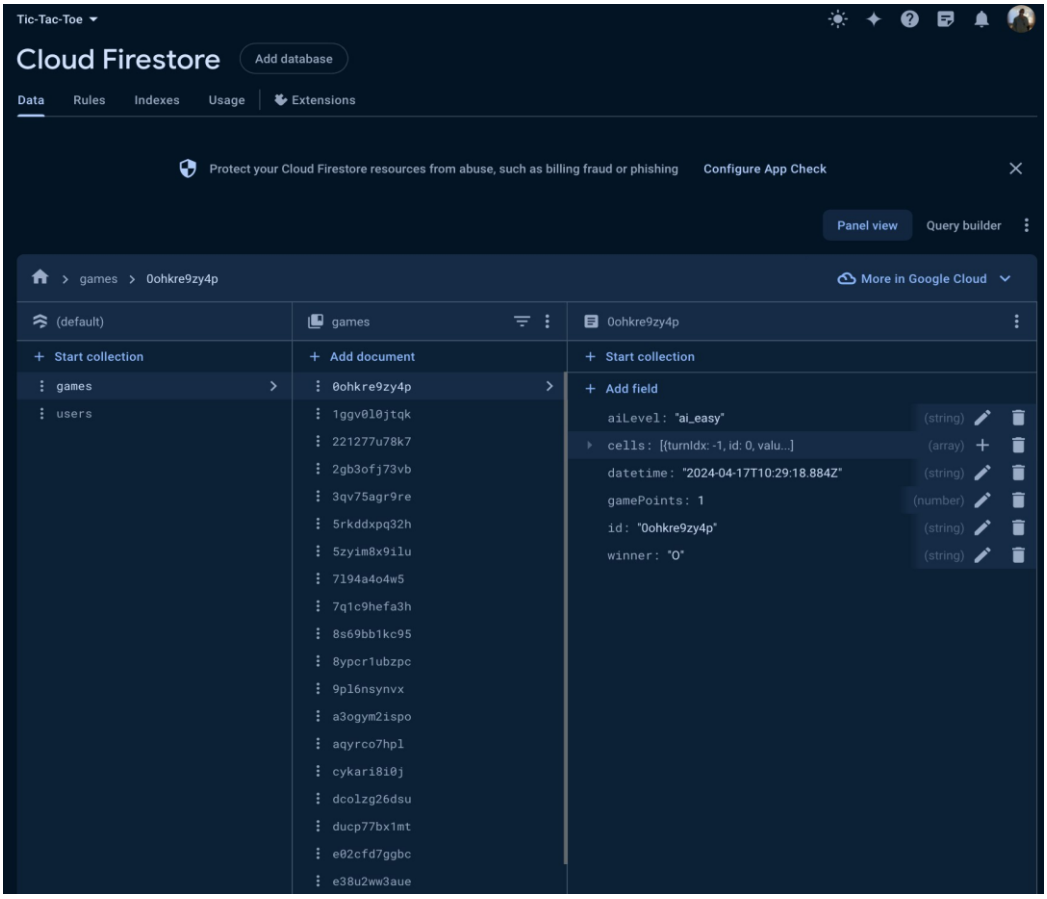


Рисунок 4.3 – База даних ігор

Проведемо 1 сесію гри, яка успішно завершилася. Про це свідчить відповідь сервера з кодом 200 OK, яка зображена на рисунку 4.4. Таким чином, було успішно додану в базу даних запис про нову гру.

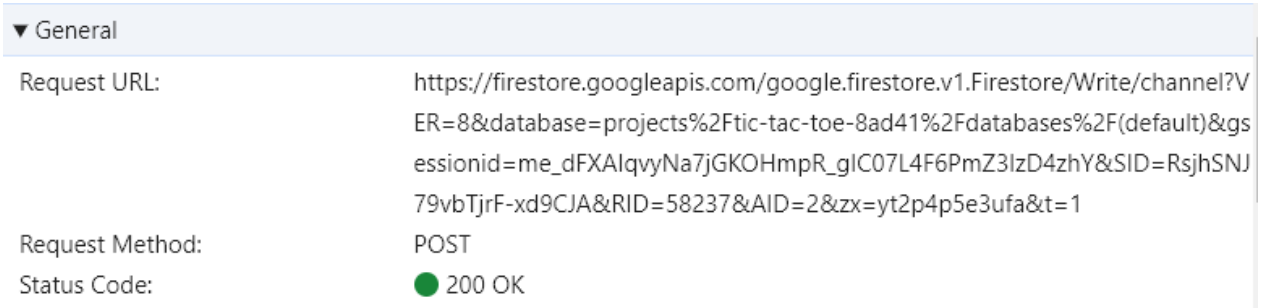


Рисунок 4.4 – Успішний запис гри в базу даних

Новий запис відмічено в базі даних (рисунок 4.5).

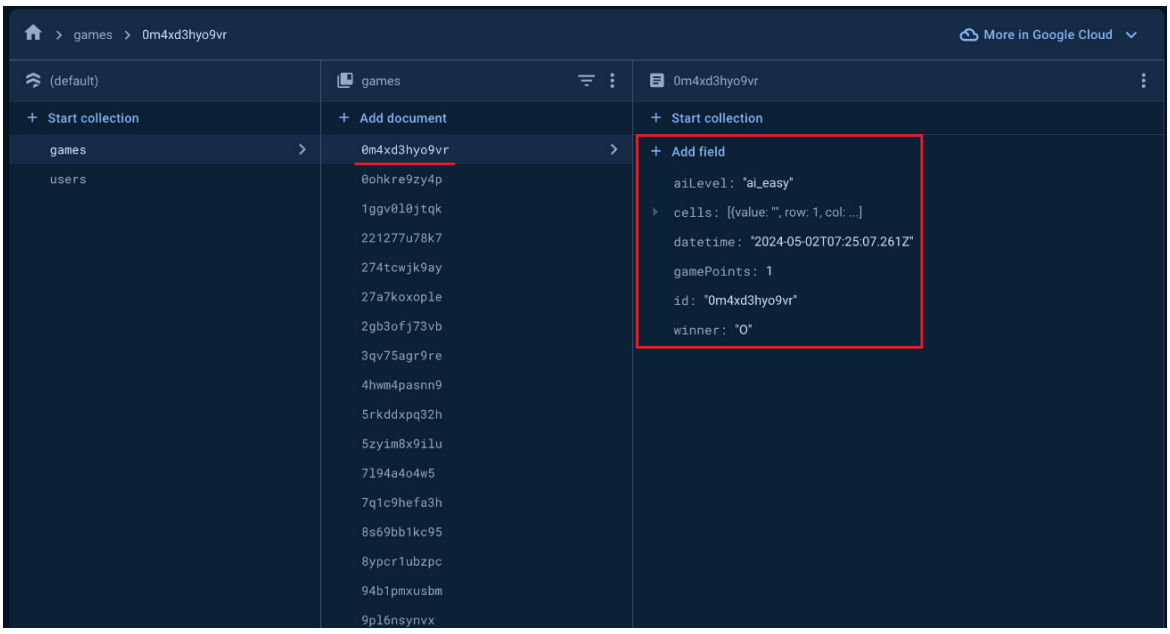


Рисунок 4.5 – Збережена в базу даних нова гра

В записі вказано складність ШІ, кількість набраних очок, переможця, та поле гри. Також записується час гри.

Створимо нового користувача в застосунку. База даних користувачів до реєстрації зображена на рисунку 4.6.

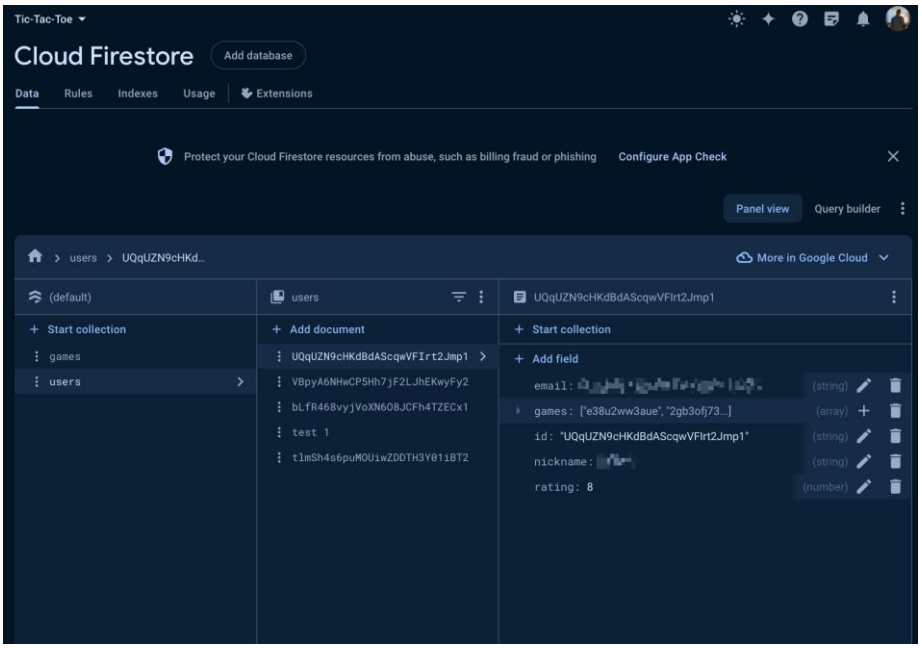


Рисунок 4.6 – База даних користувачів

Повідомлення про успішне створення нового користувача зображено на рисунку 4.7.

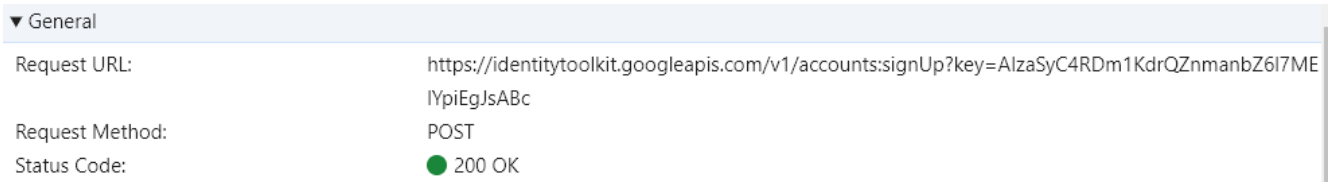


Рисунок 4.7 – Повідомлення про успішне створення користувача

Збережений в базі даних користувач зображений на рисунку 4.8.

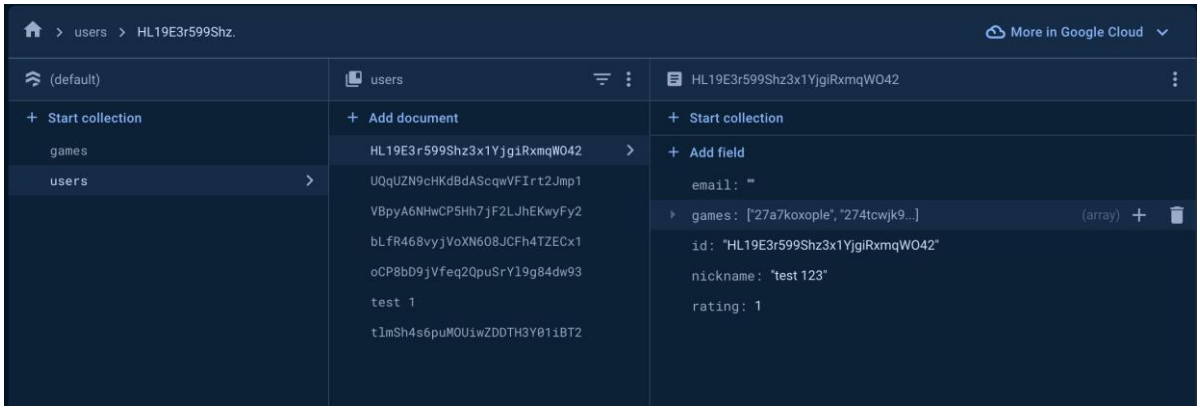


Рисунок 4.8 – Збережений в базі даних користувач

В результаті, проведене тестування демонструє якісну та коректну роботу розробленого функціоналу та впроваджених сервісів Firebase.

4.3 Висновки

У цьому розділі було проведено аналіз методів тестування веб-застосунків, алгоритмів гри, баз даних та сервісів Firebase. Було проведено тестування розробленого та впровадженого функціоналу, продемонстровано його коректну роботу. Розроблений функціонал виконує поставлені на початку розробки задачі.

ВИСНОВКИ

В результаті виконання бакалаврської дипломної роботи, було реалізовано алгоритми гри web-застосунку для розвитку логічного мислення та інтегровано сервіси Firebase у застосунок. Інтерфейс забезпечує взаємодію користувача із застосунком та його алгоритмами роботи. Він дозволяє грати у гру для розвитку логічного мислення, переглядати рейтинг гравців та власний рейтинг успішності гри.

Для розробки було використано середовище розробки Visual Studio Code, мову програмування TypeScript, фреймоворк React та сервіси Firebase.

У першому розділі, було проведено аналіз стану питання розробки алгоритмів застосунку та інтеграції сервісів Firebase, порівняльний аналіз аналогів, аналіз методів розв’язання задачі. На основі дослідження було доведено доцільність та актуальність власної розробки та поставлено задачі дослідження.

У другому розділі, було проведено аналіз даних, що використовуються та обробляються алгоритмами застосунку, розроблено алгоритм штучного інтелекту та алгоритм процесу гри, розроблено метод формування рейтингу користувача.

У третьому розділі, було проведено варіантний аналіз та обґрунтування вибору мови програмування розробки та проведено інтеграцію сервісів Firebase.

У четвертому розділі, було проведено аналіз методів тестування та виконано тестування розробленого функціоналу. Тестування довело працездатність та коректність роботи застосунку.

Таким чином можна зробити висновок, що поставлені задачі та цілі бакалаврської дипломної роботи виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Development of students logical thinking [Електронний ресурс] – Режим доступу до ресурсу: https://mental.ua/en/rozvytok-lohichnoho-myslennia-uchnia/

2. Кучер М.В., Мацедонський С.О. Розробка ігрового web-додатку для розвитку логічного мислення/М.В. Кучер, С.О. Мацедонський, В.П. Майданюк// Матеріали конференції «LIII Науково-технічна конференція підрозділів Вінницького національного технічного унівреситету (2024) », Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20790/17230

3. Cognifit [Електронний ресурс] – Режим доступу до ресурсу: https://www.cognifit.com/

4. Lumosity [Електронний ресурс] – Режим доступу до ресурсу: https://www.lumosity.com/en/

5. Peak [Електронний ресурс] – Режим доступу до ресурсу: https://www.peak.net/

6. MiniMax algorithm in game theory [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

7. Monte-Carlo algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/mathematics/monte-carlo-algorithm

8. How OpenIDConnect works [Електронний ресурс] – Режим доступу до ресурсу: https://openid.net/developers/how-connect-works/

9. Firebase Authentication [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/auth

10. What is JavaScript? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/javascript/

11. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

12. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

13. Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/

14. Firebase Hosting [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/hosting

15. Cloud functions for Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/functions

16. Firebase Cloud Messaging [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/cloud-messaging

17. Test algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/computer-science/test-algorithm

18. Database testing [Електронний ресурс] – Режим доступу до ресурсу: https://katalon.com/resources-center/blog/database-testing

ДОДАТОК А

(обов'язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
"12" березня 2024 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase»

за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

к.т.н., доц. В.П. Майданюк

«12» березня 2024 року.

Виконав:

студент гр. 2ПІ-20б М.В. Кучер

«12» березня 2024 року.

м. Вінниця – 2024 рік

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase».

Галузь застосування – web-технології, навчальний процес, програмне забезпечення.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ № 80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є покращення можливостей розвитку логічного мислення людини за рахунок розробки алгоритмів та серверної частини web-застосунку, який містить функціонал гри для розвитку логічного мислення.

Основними задачами дослідження є:

- розробити ігрову логіку для WEB-застосунку для розвитку логічного мислення;
- реалізувати алгоритм MiniMax для можливості гри проти штучного інтелекту;
- інтегрувати сервіси Firebase для аутентифікації та для роботи з базою даних;
- провести тестування застосунку.

4. Вихідні дані для проведення НДР

1. Development of students logical thinking [Електронний ресурс] – Режим доступу до ресурсу: https://mental.ua/en/rozvytok-lohichnoho-myslennia-uchnia/

2. MiniMax algorithm in game theory [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

3. Monte-Carlo algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/mathematics/monte-carlo-algorithm

4. How OpenIDConnect works [Електронний ресурс] – Режим доступу до ресурсу: https://openid.net/developers/how-connect-works/

5. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

6. Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://fire-base.google.com/

7. Firebase Hosting [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/hosting

8. Test algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/computer-science/test-algorithm

9. Database testing [Електронний ресурс] – Режим доступу до ресурсу: https://katalon.com/resources-center/blog/database-testing

5. Технічні вимоги

- середовище розробки – Microsoft Visual Studio Code;
- мова програмування – TypeScript;
- алгоритм ШІ – MiniMax;
- база даних – Firebase;

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до бакалаврської дипломної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану алгоритмів додатків для розвитку логічного мислення	14.03.24 –22.03.24
2	Розробка алгоритмів роботи штучного інтелекту	13.03.24 –22.03.24
3	Аналіз і вибір мови програмування та середовища розробки	25.03.24 –19.04.24
4	Розробка серверної частини застосунку	22.04.24 –10.05.24
5	Тестування розробленого функціоналу	13.05.24 – 24.05.24
6	Оформлення матеріалів до захисту БДР	27.05.24 – 30.05.24

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б
(обов'язковий)
ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase»

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Керівник БДР: к.т.н., доц. каф. ПЗ Майданюк В.П.

Оригінальність	94,4
Схожість	5,6

Аналіз звіту подібності

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Кучер М.В.

Керівник роботи _____ Майданюк В.П.

ДОДАТОК В

(довідниковий)

Текст коду серверної частини застосунку для розвитку логічного мислення

File: /src/shared/firebase.ts

```
import { initializeApp } from 'firebase/app';
import { getAnalytics } from 'firebase/analytics';
import { getFirestore } from '@firebase/firestore';
import { getAuth } from 'firebase/auth';

// Web app's Firebase configuration
const firebaseConfig = {
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID,
  measurementId: process.env.REACT_APP_FIREBASE_MEASUREMENT_ID,
};

// Initialize Firebase
const firebaseApp = initializeApp(firebaseConfig);
const analytics = getAnalytics(firebaseApp);
const firestore = getFirestore(firebaseApp);
const firebaseAuth = getAuth(firebaseApp);

export { firestore, analytics, firebaseAuth };
```

File: /src/shared/hooks/useRestoreUser.ts

```
import { useEffect, useState } from 'react';
import { useUserContext } from '../../providers/user/user.context';
import { firebaseAuth } from '../firebase';
import { getUserFromFirestore } from '../../providers/user/helpers/getUser';

export const useRestoreUser = () => {
  const { setUserState } = useUserContext();

  const [isLoading, setIsLoading] = useState(true);
```

```
  useEffect(() => {
    firebaseAuth.onAuthStateChanged(user => {
      if (!user) {
        setIsLoading(false);
        return;
      }

      setIsLoading(true);
      getUserFromFirestore(user.uid)
        .then(({ user }) => {
          setUserState(user);
        })
        .catch(error => {
          console.error('restore user error', { error, userId: user.uid });
        })
        .finally(() => setIsLoading(false));
    });
  }, []);

  return { isLoading };
};
```

File: /src/providers/user/user.provider.tsx

```
import React, { useCallback, useMemo, useState } from 'react';
import {
  signInWithPopup,
  GoogleAuthProvider,
  signInWithEmailAndPassword,
  createUserWithEmailAndPassword,
  setPersistence,
  browserLocalPersistence,
} from 'firebase/auth';

// types
import { UserContextType, UserSchemaType, UserType } from './user.types';

// helpers
import { firebaseAuth } from '../../shared/firebase';
import { getUserFromFirestore } from './helpers/getUser';
import { createUserInFirestore } from './helpers/createUser';
import { updateUserInFirestore } from './helpers/updateUser';

// context
```

```
import { userContext } from './user.context';

export const UserProvider = ({ children }: { children: React.ReactNode }) => {
 const [user, setUser] = useState<UserType>(null);

 const logout = useCallback(() => {
  try {
   firebaseAuth.signOut();
   setUser(null);
  } catch (error) {
   console.error('sign out error', {
    error,
   });
  }
 }, []);

 // ------------------- AUTHORIZATION METHODS
 const signInWithGoogle = useCallback(async () => {
  const provider = new GoogleAuthProvider();

  try {
   await setPersistence(firebaseAuth, browserLocalPersistence);

   const authResult = await signInWithPopup(firebaseAuth, provider);
   const userCredential = GoogleAuthProvider.credentialFromResult(authResult);

   if (!userCredential) throw new Error('auth by google credentials are empty');

   const userId = authResult.user.uid;

   const { user } = await getUserFromFirestore(userId);

   if (!user) {
    const { user } = await createUserInFirestore(userId, {
     nickname: authResult.user.displayName,
     email: authResult.user.email,
     rating: 0,
    });

    console.log('created user after google login', { user });
    setUser(user);
    return { user };
   } else {
```

```
      console.log('signed in user after google login, user exists', { user });
      setUser(user);
      return { user };
    }
  } catch (error) {
    console.error('auth by google error', {
      error,
    });
  }
}, []);

const createUserWithCredentials = useCallback(async (email: string, password: string) => {
  try {
    await setPersistence(firebaseAuth, browserLocalPersistence);

    const userCredential = await createUserWithEmailAndPassword(firebaseAuth, email, password);

    console.log('create user by credentials success', { userCredential });

    const { user } = await createUserInFirestore(userCredential.user.uid, {
      nickname: userCredential.user.displayName,
      email: '',
      rating: 0,
    });

    setUser(user);
    return { user };
  } catch (error) {
    console.error('create user by credentials error', { error });
  }
}, []);

const signInWithCredentials = useCallback(async (email: string, password: string) => {
  try {
    await setPersistence(firebaseAuth, browserLocalPersistence);

    const userCredential = await signInWithEmailAndPassword(firebaseAuth, email, password);

    console.log('log in user by credentials success', { userCredential });

    const { user } = await getUserFromFirestore(userCredential.user.uid);

    setUser(user);
```

```
    return { user };
  } catch (error) {
    console.error('log in user by credentials error', { error });
  }
 }, []);

 // ------------------- OTHER USER METHODS

 const updateUser = useCallback(
  async (userData: Partial<UserSchemaType>) => {
   try {
    if (!user) return;

    const userId = user.id;
    const { user: updatedUser } = await updateUserInFirestore(userId, { ...user, ...userData });

    setUser(updatedUser);

    return { user: updatedUser };
   } catch (error) {
    console.error('update user error', { error });
   }
  },
  [user],
 );

 const memoValue = useMemo<UserContextType>(
  () => ({
   user,
   logout,
   signInWithGoogle,
   signInWithCredentials,
   createUserWithCredentials,
   updateUser,
   setUserState: (user: UserType) => setUser(user),
  }),
  [createUserWithCredentials, logout, signInWithCredentials, signInWithGoogle, updateUser, user],
 );

 return <userContext.Provider value={memoValue}>{children}</userContext.Provider>;
};
```

File: /src/providers/user/user.context.ts

```
import React from 'react';

// types
import { UserContextType } from './user.types';

export const userContext = React.createContext<UserContextType>({
  user: null,
  logout: () => {},
  signInWithGoogle: () => {},
  createUserWithCredentials: () => {},
  signInWithCredentials: () => {},
  updateUser: () => {},
  setUserState: () => {},
});

export const useUserContext = () => {
  const context = React.useContext(userContext);

  if (!context) throw new Error('userContext should be used within UserProvider');

  return context;
};
```

File: /src/providers/user/helpers/createUser.ts

```
import { setDoc, doc } from 'firebase/firestore';

// consts
import { DEFAULT_USER } from '../user.const';

// types
import { UserType } from '../user.types';

// helpers
import { firestore } from '../../../shared/firebase';

export const createUserInFirestore = async (
  userId: string,
  userBody: Partial<Nullable<UserType>>,
) => {
  try {
    const userData: UserType = {
      ...DEFAULT_USER,
      ...userBody,
```

```
    nickname: userBody?.nickname || DEFAULT_USER.nickname,
    email: userBody?.email || DEFAULT_USER.email,
    rating: userBody?.rating || DEFAULT_USER.rating,
    games: userBody?.games || DEFAULT_USER.games,
    id: userId,
   };

   await setDoc(doc(firestore, 'users', userId), userData);

   return { user: userData };
  } catch (error) {
   throw new Error('Failed to create user');
  }
};
```

File: /src/providers/user/helpers/getUser.ts

```
import { getDoc, doc } from 'firebase/firestore';

// helpers
import { firestore } from '../../../shared/firebase';
import { userSchema } from './user.schema';

export const getUserFromFirestore = async (userId: string) => {
  try {
   const user = await getDoc(doc(firestore, 'users', userId));

   const isUserExists = user.exists();

   if (isUserExists) {
    const parsedUser = userSchema.parse(user.data());

    return { user: parsedUser };
   }

   return { user: null };
  } catch (error) {
   throw new Error('Failed to get user from firestore');
  }
};
```

File: /src/providers/user/helpers/getUserRatingPlace.ts

```
import {
  query,
```

```
  collection,
  where,
  orderBy,
  count,
  getAggregateFromServer,
} from 'firebase/firestore';

// helpers
import { firestore } from '../../../shared/firebase';
import { getUserFromFirestore } from './getUser';

export const getUserPlaceFromFirestore = async (userId: string) => {
  try {
    const { user } = await getUserFromFirestore(userId);

    if (!user) {
      console.error('user not found', { userId, user });
      return 0;
    }

    const firebaseQuery = query(
      collection(firestore, 'users'),
      orderBy('rating', 'desc'),
      where('rating', '>=', user.rating),
    );

    const querySnapshot = await getAggregateFromServer(firebaseQuery, {
      itemsCount: count(),
    });

    const userPlace = querySnapshot.data().itemsCount;

    return userPlace;
  } catch (error) {
    throw new Error('Failed to get user rating place from firestore');
  }
};
```

File: /src/providers/user/helpers/load-user-games.ts

```
import { query, collection, where, orderBy, getDocs } from 'firebase/firestore';

// helpers
import { firestore } from '../../../shared/firebase';
```

```
import { gamesSchema } from '../../game/game.schema';

export const loadUserGamesFromFirestore = async (userGamesIds: string[]) => {
 try {
  const firebaseQuery = query(
   collection(firestore, 'games'),
   orderBy('datetime', 'desc'),
   where('id', 'in', userGamesIds),
  );

  const games = await getDocs(firebaseQuery);

  const parsedGames = gamesSchema.safeParse(games.docs.map(doc => doc.data()));

  if (parsedGames.success) return parsedGames.data;

  return [];
 } catch (error) {
  console.error(error);
  throw new Error('Failed to get user games place from firestore');
 }
};
```

File: /src/providers/user/helpers/updateUser.ts

```
import { doc, updateDoc } from 'firebase/firestore';

// types
import { UserSchemaType } from '../user.types';

// helpers
import { firestore } from '../../../shared/firebase';

export const updateUserInFirestore = async (userId: string, userData: UserSchemaType) => {
 try {
  await updateDoc(doc(firestore, 'users', userId), userData);

  return { user: userData };
 } catch (error) {
  throw new Error('Failed to update user');
 }
};
```

File: src/providers/user/helpers/user.schema.ts

```
import { z } from 'zod';

export const userSchema = z.object({
  id: z.string(),
  nickname: z.string(),
  email: z.string(),
  rating: z.number(),
  games: z.array(z.string()),
});
```

File: src/providers/app/app.provider.tsx

```
import { useMemo, useState } from 'react';

// context
import { appContext } from './app.context';

// consts
import { AI_EASY } from './app.consts';

// types
import { AiLevelType, AppContextType, GameResultDataType } from './app.types';

export const AppProvider = ({ children }: { children: React.ReactNode }) => {
  const [aiLevel, setAiLevel] = useState<AiLevelType>(AI_EASY);
  const [gameResultData, setGameResultData] = useState<GameResultDataType>(null);
  const [isAuthPopupOpen, setIsAuthPopupOpen] = useState(false);

  const memoValue = useMemo<AppContextType>(
    () => ({
      // ai level
      currentAiLevel: aiLevel,
      toggleAiLevel: (aiLevel: AiLevelType) => setAiLevel(aiLevel),

      // game result data
      gameResultData,
      setGameResultData: (data: GameResultDataType) => setGameResultData(data),

      // auth popup
      isAuthPopupOpen,
      openAuthPopup: () => setIsAuthPopupOpen(true),
      closeAuthPopup: () => setIsAuthPopupOpen(false),
    }),
    [aiLevel, gameResultData, isAuthPopupOpen],
```

```
  );
  return <appContext.Provider value={memoValue}>{children}</appContext.Provider>;
};
```

File: src/providers/app/app.context.ts

```
import React from 'react';

// consts
import { AI_EASY } from './app.consts';

// types
import { AppContextType } from './app.types';

export const appContext = React.createContext<AppContextType>({
  // ai level
  currentAiLevel: AI_EASY,
  toggleAiLevel: () => {},

  // game result data
  gameResultData: null,
  setGameResultData: () => {},

  // auth popup
  isAuthPopupOpen: false,
  openAuthPopup: () => {},
  closeAuthPopup: () => {},
});

export const useAppContext = () => {
  const context = React.useContext(appContext);

  if (!context) throw new Error('appContext should be used within AppProvider');

  return context;
};
```

File: src/providers/app/app.consts.ts

```
export const AI_EASY = 'ai_easy';
export const AI_MEDIUM = 'ai_medium';
export const AI_HARD = 'ai_hard';

export const AI_LEVELS = [AI_EASY, AI_MEDIUM, AI_HARD] as const;
export const AI_LEVELS_NAMES = {
```

```
  [AI_EASY]: 'easy',
  [AI_MEDIUM]: 'medium',
  [AI_HARD]: 'hard',
} as const;
```

File: src/providers/game/game.provider.tsx

```
import { useCallback, useEffect, useMemo, useState } from 'react';

// consts
import {
  CELLS,
  CELL_EMPTY,
  CELL_O,
  CELL_X,
  GAME_IN_PROGRESS,
  GAME_LOST,
  GAME_NOT_STARTED,
  GAME_RESULT_POINTS_MAPPER,
  GAME_WON,
  INITIAL_GAME_STATE,
} from './game.conts';

// helpers
import { getMoveTimerTime } from './helpers/get-move-timer-time';
import { checkForWin } from './helpers/check-for-win';
import { getAiBestMove } from './helpers/get-best-move';
import { getGameStatus } from './helpers/get-game-status';
import { saveGameDoc } from './helpers/save-game';

// context
import { useAppContext } from '../app/app.context';
import { useToasterContext } from '../toaster/toaster.context';
import { useUserContext } from '../user/user.context';
import { gameContext } from './game.context';

// types
import {
  CellsValueType,
  GameContextType,
  GameCellType,
  GameStateType,
  GameResultType,
} from './game.types';
```

```
export const GameProvider = ({ children }: { children: React.ReactNode }) => {
  const { currentAiLevel, setGameResultData } = useAppContext();
  const { bug } = useToasterContext();
  const { updateUser, user } = useUserContext();

  const [gameCells, setGameCells] = useState<GameCellType[]>(CELLS);
  const [gameState, setGameState] = useState<GameStateType>(INITIAL_GAME_STATE);

  /**
   * effect to perofm an ai move, will run every time currentMove changes (so every move)
   */
  useEffect(() => {
    const moveTimeout = setTimeout(() => {
      if (CELL_O === gameState.currentMove) return;

      const aiMoveId = getAiBestMove({ cells: gameCells, aiLevel: currentAiLevel });

      if (aiMoveId === null) {
        bug('Не вдалось розрахувати відповідь бота');
        console.error('can not calculate ai move', {
          args: { cells: gameCells, aiLevel: currentAiLevel },
        });
        return;
      }

      makeMove(aiMoveId, gameState.currentMove);
    }, 750);

    return () => clearTimeout(moveTimeout);
  }, [gameState.currentMove]);

  /**
   * method to clear the game field and reset the game state
   */
  const resetGameField = useCallback(() => {
    setGameCells(CELLS);

    setGameState(INITIAL_GAME_STATE);
  }, []);

  /**
   * method to save game
```

```
 */
const onGameEnd = useCallback(
  async ({ gameResult, cells }: { gameResult: GameResultType; cells: GameCellType[] }) => {
    try {
      const gamePoints = GAME_RESULT_POINTS_MAPPER[currentAiLevel][gameResult];

      setGameResultData({
        gameResult,
      });

      resetGameField();

      // update user and save game only if user is logged in
      if (user !== null) {
        const gameId = String(Math.random().toString(36).slice(2));
        const savedGame = await saveGameDoc({
          gameId,
          gameData: cells,
          aiLevel: currentAiLevel,
          gamePoints,
          winner: gameResult === GAME_WON ? CELL_O : gameResult === GAME_LOST ? CELL_X : null,
        });
        const updatedUser = await updateUser({
          rating: Number(user?.rating) + gamePoints,
          games: [...(user?.games ?? []), gameId],
        });

        return { savedGame, updatedUser };
      }
    } catch (error) {
      console.error('save game error', { error });
    }
  },
  [currentAiLevel, resetGameField, setGameResultData, updateUser, user?.games, user?.rating],
);

/**
 * method to perform a move
 */
const makeMove = useCallback(
  (id: number, value: CellsValueType) => {
    const updatedCells = gameCells.map(cell =>
      cell.id === id ? { ...cell, value, turnIdx: gameState.currentMoveIdx } : cell,
```

```
    );

    setGameCells(updatedCells);

    const gameFinishStatus = checkForWin({ cells: updatedCells });
    const gameStatus = getGameStatus({
      checkingWinnerResult: gameFinishStatus,
    });

    if (gameStatus !== GAME_IN_PROGRESS && gameStatus !== GAME_NOT_STARTED) {
      onGameEnd({ gameResult: gameStatus, cells: updatedCells });

      return;
    }

    setGameState(prev => {
      const currentMove = prev.currentMove === CELL_O ? CELL_X : CELL_O;

      return {
        ...prev,
        currentMove,
        currentMoveEndsIn: getMoveTimerTime(),
        currentMoveIdx: prev.currentMoveIdx + 1,
        gameStatus,
      };
    });
  },
  [gameCells, gameState.currentMoveIdx, onGameEnd],
);

/**
 * method to surrender the game
 */
const surrender = useCallback(() => {
  onGameEnd({ gameResult: GAME_LOST, cells: gameCells });
}, [onGameEnd, gameCells]);

const memoValue = useMemo<GameContextType>(() => {
  const isGameStarted = gameCells.some(({ value }) => value !== CELL_EMPTY);

  return {
    // game cells
    cells: gameCells,
```

```
    makeMove,

    // game data
    currentMove: gameState.currentMove,
    currentMoveEndsIn: isGameStarted ? gameState.currentMoveEndsIn : 0,
    gameStatus: gameState.gameStatus,

    // game methods
    surrender,
    resetGameField,
   };
  }, [
   gameCells,
   makeMove,
   gameState.currentMove,
   gameState.currentMoveEndsIn,
   gameState.gameStatus,
   surrender,
   resetGameField,
  ]);

  return <gameContext.Provider value={memoValue}>{children}</gameContext.Provider>;
};
```

File: src/providers/game/game.context.ts

```
import React from 'react';

// consts
import { CELLS, CELL_O, GAME_NOT_STARTED } from './game.conts';

// types
import { GameContextType } from './game.types';

export const gameContext = React.createContext<GameContextType>({
  // game cells
  cells: CELLS,
  makeMove: () => {},

  // game data
  currentMove: CELL_O,
  currentMoveEndsIn: 0,
  gameStatus: GAME_NOT_STARTED,
```

```
 // game methods
 surrender: () => {},
 resetGameField: () => {},
});

export const useGameContext = () => {
 const context = React.useContext(gameContext);

 if (!context) throw new Error('gameContext should be used within GameProvider');

 return context;
};
```

File: src/providers/game/game.conts.ts

```
// consts
import { AI_EASY, AI_MEDIUM, AI_HARD } from '../app/app.consts';

// types
import { GameCellType, GameStateType } from './game.types';

const GAME_CELLS = 9;

export const CELL_X = 'X';
export const CELL_O = 'O';
export const CELL_EMPTY = '';

export const CELLS: GameCellType[] = Array.from({ length: GAME_CELLS }, (_, idx) => ({
 id: idx,
 value: CELL_EMPTY,
 turnIdx: -1,
 row: Math.ceil((idx + 1) / 3),
 col: (idx + 1) % 3 === 0 ? 3 : (idx + 1) % 3,
}));

// Game status
export const GAME_NOT_STARTED = 'not_started';
export const GAME_IN_PROGRESS = 'in_progress';
export const GAME_WON = 'won';
export const GAME_LOST = 'lost';
export const GAME_TIE = 'tie';

export const INITIAL_GAME_STATE: GameStateType = {
 currentMove: CELL_O,
```

```
  currentMoveEndsIn: 0,
  currentMoveIdx: 1,
  gameStatus: GAME_NOT_STARTED,
};

export const AI_LEVELS_EFFICIENT_MOVE_PERSENTAGES = {
  [AI_EASY]: 0.33,
  [AI_MEDIUM]: 0.66,
  [AI_HARD]: 1,
} as const;

export const GAME_RESULT_POINTS_MAPPER = {
  [AI_EASY]: {
    [GAME_WON]: 1,
    [GAME_LOST]: -1,
    [GAME_TIE]: 0,
  },
  [AI_MEDIUM]: {
    [GAME_WON]: 2,
    [GAME_LOST]: -1,
    [GAME_TIE]: 0,
  },
  [AI_HARD]: {
    [GAME_WON]: 5,
    [GAME_LOST]: -1,
    [GAME_TIE]: 0,
  },
} as const;
```

File: src/providers/game/helpers/check-for-win.ts

```
import { CELL_EMPTY, GAME_IN_PROGRESS, GAME_TIE } from '../game.conts';
import { CellValueType, GameCellType } from '../game.types';

// const with all winning conditions
const WIN_CONDITIONS = [
  // 3 in row win
  [
    { row: 1, col: 1, idx: 0 },
    { row: 1, col: 2, idx: 1 },
    { row: 1, col: 3, idx: 2 },
  ],
  [
    { row: 2, col: 1, idx: 3 },
```

```
    { row: 2, col: 2, idx: 4 },
    { row: 2, col: 3, idx: 5 },
  ],
  [
    { row: 3, col: 1, idx: 6 },
    { row: 3, col: 2, idx: 7 },
    { row: 3, col: 3, idx: 8 },
  ],

  // 3 in collumn win
  [
    { row: 1, col: 1, idx: 0 },
    { row: 2, col: 1, idx: 3 },
    { row: 3, col: 1, idx: 6 },
  ],
  [
    { row: 1, col: 2, idx: 1 },
    { row: 2, col: 2, idx: 4 },
    { row: 3, col: 2, idx: 7 },
  ],
  [
    { row: 1, col: 3, idx: 2 },
    { row: 2, col: 3, idx: 5 },
    { row: 3, col: 3, idx: 8 },
  ],

  // diagonal left-top to right-bottom win
  [
    { row: 1, col: 1, idx: 0 },
    { row: 2, col: 2, idx: 4 },
    { row: 3, col: 3, idx: 8 },
  ],

  // diagonal left-bottom to right-top win
  [
    { row: 1, col: 3, idx: 2 },
    { row: 2, col: 2, idx: 4 },
    { row: 3, col: 1, idx: 6 },
  ],
] as const;

type Args = { cells: GameCellType[] };
```

```
type ReturnType = typeof GAME_IN_PROGRESS | typeof GAME_TIE | CellValueType;

/**
 * method to check for someones win
 *
 * @param cells game cells
 */
export const checkForWin = ({ cells }: Args): ReturnType => {
  const { winner } = WIN_CONDITIONS.reduce<{ winner: CellValueType | null }>(
    (acc, winCondition) => {
      const [cellOne, cellTwo, cellThree] = winCondition;

      const cellOneValue = cells.at(cellOne.idx)?.value;
      const cellTwoValue = cells.at(cellTwo.idx)?.value;
      const cellThreeValue = cells.at(cellThree.idx)?.value;

      // if some of the cells are empty return current acc
      if (!cellOneValue || !cellTwoValue || !cellThreeValue) return acc;

      if (
        cellOneValue === cellTwoValue &&
        cellTwoValue === cellThreeValue &&
        cellOneValue === cellThreeValue
      ) {
        acc.winner = cellOneValue;
      }

      return acc;
    },
    {
      winner: null,
    },
  );

  if (!cells.find(({ value }) => value === CELL_EMPTY) && !winner) return GAME_TIE;

  return winner ?? GAME_IN_PROGRESS;
};
```

File: src/providers/game/helpers/get-best-move.ts

```
// types
import { CellValueType, GameCellType } from '../game.types';
```

```
// consts
import { AI_LEVELS_EFFICIENT_MOVE_PERSENTAGES, CELL_EMPTY, CELL_X } from '../game.conts';

// helpers
import { minimax } from './minimax';
import { AiLevelType } from '../../app/app.types';

type Args = {
  cells: GameCellType[];
  aiLevel: AiLevelType;
};

/**
 * method to get bot's move
 * as we have different levels of bot difficulty
 *  – for hard lvl bot it will use most efficient move
 *  – for medium lvl bot it will take efficient move in 66% (every 2nd)
 *  – for easy lvl bot it will take efficient move in 33% (every 3rd)
 *
 * @param cells – current board after player's move
 * @param aiLevel – selected bot level
 *
 * @returns id of the cell where bot will move
 */
export const getAiBestMove = ({ cells, aiLevel }: Args) => {
  const shoudlTakeMostEfficientMove =
    Math.random() <= AI_LEVELS_EFFICIENT_MOVE_PERSENTAGES[aiLevel];

  if (!shoudlTakeMostEfficientMove) {
    const emptyCells = cells.filter(({ value }) => value === CELL_EMPTY);
    const randomEmptyCellId = emptyCells.at(Math.floor(Math.random() * emptyCells.length))?.id;

    return randomEmptyCellId ?? null;
  }

  const { moveCellIdx } = cells.reduce<{
    bestMoveScore: number;
    moveCellIdx: number | null;
  }>(
    (acc, cell) => {
      const { value, id: currentCellId } = cell;

      // check for an empty cell
```

```
    if (value !== CELL_EMPTY) return acc;

    const updatedCells: GameCellType[] = cells.map(cell =>
      cell.id === currentCellId ? { ...cell, value: CELL_X } : cell,
    );

    const moveScore = minimax({
      cells: updatedCells,
      isMaximizing: false,
      checkingDepth: 0,
    });

    if (moveScore >= acc.bestMoveScore) {
      acc.bestMoveScore = moveScore;
      acc.moveCellIdx = currentCellId;
    }

    return acc;
   },
   {
    bestMoveScore: -Infinity,
    moveCellIdx: null,
   },
  );

  return moveCellIdx;
};
```

File: src/providers/game/helpers/get-game-status.ts

```
// types
import { GameStatusType } from '../game.types';

// helopers
import { checkForWin } from './check-for-win';

// consts
import { CELL_O, CELL_X, GAME_IN_PROGRESS, GAME_LOST, GAME_TIE, GAME_WON } from '../game.conts';

type Args = {
  checkingWinnerResult: ReturnType<typeof checkForWin>;
};

/**
```

```
 * method to get game status based on checkiing winner result
 */
export const getGameStatus = ({ checkingWinnerResult }: Args): GameStatusType => {
  // if tie return tie
  if (checkingWinnerResult === GAME_TIE) return GAME_TIE;

  // if someone won check whether winner is player if yes return game won othervise lost
  if (checkingWinnerResult === CELL_X || checkingWinnerResult === CELL_O) {
    return checkingWinnerResult === CELL_O ? GAME_WON : GAME_LOST;
  }

  // if all conds failed return game in progress status
  return GAME_IN_PROGRESS;
};
```

File: src/providers/game/helpers/get-move-score.ts

```
// consts
import { CELL_O, GAME_IN_PROGRESS, GAME_TIE } from '../game.conts';

// types
import { CellValueType } from '../game.types';

// helpers
import { checkForWin } from './check-for-win';

type Args = {
  gameResult: ReturnType<typeof checkForWin>;
};

export const getMoveScore = ({ gameResult }: Args) => {
  // if game finished with tie return 0 to that move
  if (gameResult === GAME_TIE) return 0;

  // if game is not finished return null to that move
  if (gameResult === GAME_IN_PROGRESS) return null;

  // if player won return -1
  if (gameResult === CELL_O) return -1;

  // if minimax won return 1
  return 1;
};
```

File: src/providers/game/helpers/get-move-timer-time.ts

```
export const getMoveTimerTime = () => new Date(Number(Date.now()) + 30 * 1000).getTime();
```

File: src/providers/game/helpers/minimax.ts

```
// types
import { CELL_EMPTY, CELL_O, CELL_X } from '../game.conts';
import { CellValueType, GameCellType } from '../game.types';
import { getMoveScore } from './get-move-score';

// helpers
import { checkForWin } from './check-for-win';

// helpers

type Args = {
  cells: GameCellType[];
  checkingDepth: number;
  isMaximizing: boolean;
};

/**
 * method to check how good a move is, it recursively calls all possible next moves,
 * if bot will move the cell updated before passing cells here
 *
 * @param cells – game field
 * @param checkingDepth – current depth
 * @isMaximizing – boolean value whether we check for bot move (true) or for players turn (false)
 * @returns score of the move into the current cell
 */
export const minimax = ({ cells, checkingDepth, isMaximizing }: Args) => {
  const gameResult = checkForWin({ cells });
  const moveScore = getMoveScore({ gameResult });

  // if game is over return move score
  if (moveScore !== null) return moveScore;

  // check move for bot
  if (isMaximizing) {
    return cells.reduce((acc, currentCell) => {
      const { value, id: currentCellId } = currentCell;

      // check for an empty cell
      if (value !== CELL_EMPTY) return acc;
```

```
    // update cell with bot's sign
    const updatedCells: GameCellType[] = cells.map(cell =>
      cell.id === currentCellId ? { ...cell, value: CELL_X } : cell,
    );

    // get score for the move above
    const nextMoveScore = minimax({
      cells: updatedCells,
      isMaximizing: false,
      checkingDepth: checkingDepth + 1,
    });

    // check whether it's better than current best move, stored in acc
    acc = Math.max(acc, nextMoveScore);

    return acc;
  }, -Infinity);
}

// check move for player
return cells.reduce((acc, currentCell) => {
  const { value, id: currentCellId } = currentCell;

  // check for an empty cell
  if (value !== CELL_EMPTY) return acc;

  // update cell with player's sign
  const updatedCells: GameCellType[] = cells.map(cell =>
    cell.id === currentCellId ? { ...cell, value: CELL_O } : cell,
  );

  // get score for the move above
  const nextMoveScore = minimax({
    cells: updatedCells,
    isMaximizing: true,
    checkingDepth: checkingDepth + 1,
  });

  // check whether it's better (for user, but worse for bot, so we use min here) than current best move, stored in acc
  acc = Math.min(acc, nextMoveScore);

  return acc;
```

```
  }, Infinity);
};
```

File: src/providers/game/helpers/save-game.ts

```
import { setDoc, doc } from 'firebase/firestore';
import { firestore } from '../../../shared/firebase';
import { CellValueType, GameCellType, GameFullType } from '../game.types';
import { AiLevelType } from '../../../app/app.types';

export const saveGameDoc = async ({
  gameId,
  aiLevel,
  winner,
  gamePoints,

  gameData,
}: {
  gameId: string;
  gamePoints: number;
  aiLevel: AiLevelType | null;
  winner: CellValueType | null;

  gameData: GameCellType[];
}) => {
  try {
    const gameToSave: GameFullType = {
      id: gameId,
      datetime: new Date().toISOString(),
      aiLevel,
      winner,
      gamePoints,

      cells: gameData,
    };

    await setDoc(doc(firestore, 'games', gameId), gameToSave);

    return { game: gameToSave };
  } catch (error) {
    throw error;
  }
};
```

File: src/pages/profile/profile.tsx

```
import { useEffect, useMemo, useState } from 'react';
import classNames from 'classnames';

// components
import { GamesSlider } from './components/games-slider/games-slider';
import { Plural } from '../../components/plural/plural';

// context
import { useUserContext } from '../../providers/user/user.context';
import { useToasterContext } from '../../providers/toaster/toaster.context';

// types
import { UserSchemaType } from '../../providers/user/user.types';

// consts
import { DEFAULT_USER } from '../../providers/user/user.const';

// helpers
import { getUserPlaceFromFirestore } from '../../providers/user/helpers/getUserRatingPlace';

// styles
import styles from './profile.module.css';
import { Loader } from '../../components/loader/loader';

export const ProfileScreen = () => {
  const { user, updateUser } = useUserContext();
  const { success } = useToasterContext();

  const [editableUser, setEditableUser] = useState<UserSchemaType>(DEFAULT_USER);
  const [userPlace, setUserPlace] = useState(0);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    void (async () => {
      try {
        setIsLoading(true);
        if (user) {
          const place = await getUserPlaceFromFirestore(user.id);

          setEditableUser(user);
          setUserPlace(place);
        }
```

```
    } catch (error) {
      console.error('get user data error', { error });
    } finally {
      setIsLoading(false);
    }
  })();
}, [user]);

const handleUpdateUser = async (event: React.FormEvent<HTMLFormElement>) => {
  event.preventDefault();

  try {
    await updateUser(editableUser);
    success('Ваш профіль було оновлено');
  } catch (error) {
    console.error('user update form error', { error });
  }
};

const handleNameChange = (event: React.ChangeEvent<HTMLInputElement>) => {
  const { value, name } = event.target;

  setEditableUser(prev => ({
    ...prev,
    [name]: value,
  }));
};

const isSaveUserDisabled = useMemo(() => {
  return !user || editableUser.nickname === user?.nickname;
}, [editableUser.nickname, user]);

return (
  <div className={classNames(styles['profile-page'])}>
    <div
      className={classNames(styles['loader-wrapper'], {
        [styles['active']]: isLoading,
      })}
    >
      <Loader />
    </div>

    <form onSubmit={handleUpdateUser} className={classNames(styles['user-data-form'])}>
```

```
      <div className={classNames(styles.input, styles['user-name'])}>
       <label htmlFor="nickname">Нікнейм</label>
       <input
        id="nickname"
        name="nickname"
        value={editableUser.nickname}
        onChange={handleNameChange}
        placeholder="..."
        required
       />
      </div>
      <div className={classNames(styles['rating'])}>
       Рейтинг:{' '}
       <b>
        {editableUser.rating} 
        <Plural count={editableUser.rating} many="очок" one="очко" other="очок" few="очки" />
       </b>
      </div>
      <div className={classNames(styles['placement'])}>
       Місце у рейтингу: <b>#{userPlace}</b>
      </div>

      <button
       type="submit"
       className={classNames(styles['save-user-btn'])}
       disabled={isSaveUserDisabled}
      >
       Зберегти дані
      </button>
     </form>

     <div className={classNames(styles['games-history-title'])}>Історія ігор</div>
     <GamesSlider />
    </div>
  );
};
```

File: src/pages/profile/profile.module.css

```
@value vars: "../../shared/styles/vars.module.css";
@value phone from vars;

.profile-page {
        min-width: 100vw;
```

```
min-height: 100vh;

display: flex;
flex-direction: column;
position: relative;

align-items: center;

padding: 16px;
padding-top: 76px;
row-gap: 24px;

background-color: var(--color-backgound);
box-sizing: border-box;

.loader-wrapper {
        position: absolute;
        z-index: 100;

        width: 100%;
        height: 75%;

        display: flex;
        justify-content: center;
        align-items: center;
        pointer-events: none;

        background-color: var(--color-backgound);
        opacity: 0;
        transition: opacity 500ms ease-out;

        &.active {
                cursor: not-allowed;
                opacity: 1;
                pointer-events: all;
        }

}

.user-data-form {
        display: grid;

        width: 100%;
```

```
height: 100%;
max-width: clamp(250px, 100%, 550px);

margin-inline: auto;

padding: 16px;

box-sizing: border-box;

align-items: center;

grid-template-columns: auto minmax(0, 300px);
grid-template-rows: 50px 50px;

grid-template-areas:
                "user-name placement"
                "rating save-user-btn";

gap: 12px;

border: 1px solid var(--color-border);

@media (max-width: phone) {
        padding: 30px 16px 16px 16px;

        grid-template-rows: unset;
        grid-template-columns: 1fr;
        grid-template-areas:
                "user-name"
                "rating"
                "placement"
                "save-user-btn";

        .user-name {
                input {
                        width: 100%;
                }
        }
}

.user-name {
        grid-area: user-name;
```

```
        position: relative;

        input {
            box-sizing: border-box;
            outline: none;
            padding: 4px 12px;

            color: var(--color-dark);
            font-size: 16px;

            background: transparent;

            border: 1px solid var(--color-border);
            border-radius: 4px;

            &::placeholder {
                color: var(--color-dark);
                font-size: 16px;
            }
        }

        label {
            position: absolute;
            top: -14px;
            left: 4px;

            color: var(--color-dark);
            font-size: 10px;
        }
    }

    .rating {
        grid-area: rating;

        font-size: 14px;
        color: var(--color-dark);

        b {
            font-weight: 700;
        }
    }

    .placement {
```

```
                grid-area: placement;

                font-size: 14px;
                color: var(--color-dark);

                b {
                        font-weight: 700;
                }
        }

        .save-user-btn {
                grid-area: save-user-btn;

                width: 100%;
                height: 50px;

                font-weight: 600;
                color: var(--color-dark);

                cursor: pointer;
                background-color: var(--color-accent);
                border: 1px solid var(--color-border);

                &:disabled {
                        opacity: 0.6;
                        cursor: not-allowed;
                }
        }
    }

    .games-history-title {
        text-align: center;
        color: var(--color-dark);
        font-size: 24px;
        font-weight: 700;

        margin-top: 24px;
    }
}

[data-theme='dark'] {
    .profile-page {
        .user-data-form {
```

```
                .user-name {
                    input {
                        color: var(--color-light);

                        &::placeholder {
                            color: var(--color-light);
                        }
                    }

                    label {
                        color: var(--color-light);
                    }
                }

                .rating {
                    color: var(--color-light);
                }

                .placement {
                    color: var(--color-light);
                }

                .save-user-btn {

                }
            }

            .games-history-title {
                color: var(--color-light);
            }
        }
}
```

File: src/pages/profile/components/games-slider/games-slider.tsx

```
import { useCallback, useEffect, useState } from 'react';
import classNames from 'classnames';
import useEmblaCarousel from 'embla-carousel-react';

// types
import type { EmblaOptionsType } from 'embla-carousel';
import { GameFullType, GamesFullType } from '../../../../providers/game/game.types';

// helpers
```

```
import { loadUserGamesFromFirestore } from '../../../../providers/user/helpers/load-user-games';

// context
import { useUserContext } from '../../../../providers/user/user.context';

// components
import { Plural } from '../../../../components/plural/plural';
import { GameReplayPopup } from '../game-replay-popup/game-replay-popup';

// consts
import { CELL_O, CELL_X } from '../../../../providers/game/game.conts';

// styles
import styles from './games-slider.module.css';
import { PrevButton, NextButton, usePrevNextButtons } from './components/arrows';
import { useDotButton, DotButton } from './components/dots';

const SLIDER_OPTIONS: Partial<EmblaOptionsType> = {
 align: 'start',
 skipSnaps: false,
};

export const GamesSlider = () => {
 const { user } = useUserContext();

 const [games, setGames] = useState<null | GamesFullType>(null);
 const [isGamesLoading, setIsGamesLoading] = useState(false);

 const [gameToReplay, setGameToReplay] = useState<null | GameFullType>(null);
 const [replayMoveIdx, setReplayMoveIdx] = useState<null | number>(null);

 // Start replay (show game 1st move)
 const startReplay = useCallback(() => {
  setReplayMoveIdx(1);
 }, []);

 // Increase replay move
 const replayNextMove = useCallback(() => {
  setReplayMoveIdx(prev => Math.min(Number(prev) + 1, 9));
 }, []);

 // Descrease replay move
 const replayPrevMove = useCallback(() => {
```

```
  setReplayMoveIdx(prev => Math.max(Number(prev) - 1, 1));
}, []);

// Close popup, reset game and move idx
const closeReplayPopup = useCallback(() => {
  setGameToReplay(null);
  setReplayMoveIdx(null);
}, []);

const [emblaRef, emblaApi] = useEmblaCarousel(SLIDER_OPTIONS);

const { prevBtnDisabled, nextBtnDisabled, onPrevButtonClick, onNextButtonClick } =
  usePrevNextButtons(emblaApi);

const { selectedIndex, scrollSnaps, onDotButtonClick } = useDotButton(emblaApi);

/**
 * effect to load games
 */
useEffect(() => {
  if (!user) return;

  void (async () => {
    try {
      setIsGamesLoading(true);

      const gamesFromFirestore = await loadUserGamesFromFirestore(user.games);

      setGames(gamesFromFirestore);
    } catch (error) {
      console.error('error while loading games', error);
    } finally {
      setIsGamesLoading(false);
    }
  })();
}, [user?.games?.join(',')]);

useEffect(() => {
  emblaApi?.reInit();
}, [games]);

return (
  <>
```

```
<div className={styles['games-slider-wrapper']}>
 <div ref={emblaRef} className={styles['slider-viewport']}>
  <div className={styles['slider-container']}>
   {games?.map(game => {
    const { winner, gamePoints } = game;
    const isWon = winner === CELL_O;
    const isLost = winner === CELL_X;

    return (
     <div
      key={game.id}
      className={classNames(styles['slide'], {
       [styles['won']]: isWon,
       [styles['lost']]: isLost,
      })}
      onClick={() => setGameToReplay(game)}
     >
      <span>{isWon ? 'Перемога' : isLost ? 'Програш' : 'Нічия'}</span>{' '}
      <span>
       {gamePoints >= 0 ? '+' : '–'} {Math.abs(gamePoints)}
        
       <Plural count={gamePoints} many="очок" one="очко" other="очок" few="очки" />
      </span>
     </div>
    );
   })}
  </div>
 </div>

 <div className={styles['embla__controls']}>
  <div className={styles['embla__buttons']}>
   <PrevButton onClick={onPrevButtonClick} disabled={prevBtnDisabled} />
   <NextButton onClick={onNextButtonClick} disabled={nextBtnDisabled} />
  </div>

  <div className={styles['embla__dots']}>
   {scrollSnaps.map((_, index) => (
    <DotButton
     key={index}
     onClick={() => onDotButtonClick(index)}
     className={classNames(styles['embla__dot'], {
      [styles['embla__dot--selected']]: index === selectedIndex,
     })}
```

```
            />
          ))}
        </div>
      </div>
    </div>

    <GameReplayPopup
      handleClose={closeReplayPopup}
      selectedGame={gameToReplay}
      currentMoveIdx={replayMoveIdx}
      startReplay={startReplay}
      nextMove={replayNextMove}
      prevMove={replayPrevMove}
    />
  </>
 );
};
```

File: src/pages/profile/components/games-slider/games-slider.module.css

```
.games-slider-wrapper {
        max-width: 650px;
        width: 90%;

        .slider-viewport {
                overflow: hidden;

                .slider-container {
                        display: flex;
                        column-gap: 16px;

                        backface-visibility: hidden;

                        .slide {
                                color: var(--color-gray-light);
                                white-space: nowrap;

                                &.won {
                                        color: var(--color-green);
                                }

                                &.lost {
                                        color: var(--color-red);
                                }
```

```
                                }
                            }
                        }
                }
}

.embla__controls {
  display: grid;
  grid-template-columns: auto 1fr;
  justify-content: space-between;
  gap: 1.2rem;
  margin-top: 1.8rem;
}
.embla__buttons {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 0.6rem;
  align-items: center;
}
.embla__button {
  -webkit-tap-highlight-color: rgba(var(--text-high-contrast-rgb-value), 0.5);
  -webkit-appearance: none;
  appearance: none;
  background-color: transparent;
  touch-action: manipulation;
  display: inline-flex;
  text-decoration: none;
  cursor: pointer;
  border: 0;
  padding: 0;
  margin: 0;
  box-shadow: inset 0 0 0 0.2rem var(--detail-medium-contrast);
  width: 32px;
  height: 32px;
  z-index: 1;
  border-radius: 50%;
  color: var(--text-body);
  display: flex;
  align-items: center;
  justify-content: center;
}
.embla__button:disabled {
  color: var(--detail-high-contrast);
}
```

```
.embla__button__svg {
  width: 35%;
  height: 35%;
}
.embla__dots {
  display: flex;
  flex-wrap: wrap;
  justify-content: flex-end;
  align-items: center;
              column-gap: 4px;
}
.embla__dot {
  -webkit-tap-highlight-color: rgba(var(--text-high-contrast-rgb-value), 0.5);
  -webkit-appearance: none;
  appearance: none;
  background-color: transparent;
  touch-action: manipulation;
  display: inline-flex;
  text-decoration: none;
  cursor: pointer;
  border: 0;
  padding: 0;
  margin: 0;
  width: 12px;
  height: 12px;
  display: flex;
  align-items: center;
  justify-content: center;
  border-radius: 50%;
}
.embla__dot:after {
  box-shadow: inset 0 0 0 2px var(--detail-medium-contrast);
  width: 12px;
  height: 12px;
  border-radius: 50%;
  display: flex;
  align-items: center;
  content: '';
}
.embla__dot--selected:after {
  box-shadow: inset 0 0 0 2px var(--text-body);
}
```

File: src/pages/profile/components/games-slider/components/dots.tsx

```
import React, { PropsWithChildren, useCallback, useEffect, useState } from 'react';
import { EmblaCarouselType } from 'embla-carousel';

type UseDotButtonType = {
  selectedIndex: number;
  scrollSnaps: number[];
  onDotButtonClick: (index: number) => void;
};

export const useDotButton = (emblaApi: EmblaCarouselType | undefined): UseDotButtonType => {
  const [selectedIndex, setSelectedIndex] = useState(0);
  const [scrollSnaps, setScrollSnaps] = useState<number[]>([]);

  const onDotButtonClick = useCallback(
    (index: number) => {
      if (!emblaApi) return;
      emblaApi.scrollTo(index);
    },
    [emblaApi],
  );

  const onInit = useCallback((emblaApi: EmblaCarouselType) => {
    setScrollSnaps(emblaApi.scrollSnapList());
  }, []);

  const onSelect = useCallback((emblaApi: EmblaCarouselType) => {
    setSelectedIndex(emblaApi.selectedScrollSnap());
  }, []);

  useEffect(() => {
    if (!emblaApi) return;

    onInit(emblaApi);
    onSelect(emblaApi);
    emblaApi.on('reInit', onInit);
    emblaApi.on('reInit', onSelect);
    emblaApi.on('select', onSelect);
  }, [emblaApi, onInit, onSelect]);

  return {
    selectedIndex,
    scrollSnaps,
```

```
    onDotButtonClick,
  };
};

type PropType = PropsWithChildren<
  React.DetailedHTMLProps<React.ButtonHTMLAttributes<HTMLButtonElement>, HTMLButtonElement>
>;

export const DotButton: React.FC<PropType> = props => {
  const { children, ...restProps } = props;

  return (
    <button type="button" {...restProps}>
      {children}
    </button>
  );
};
```

File: src/pages/profile/components/games-slider/components/arrows.tsx

```
import React, { PropsWithChildren, useCallback, useEffect, useState } from 'react';
import { EmblaCarouselType } from 'embla-carousel';

import styles from '../games-slider.module.css';
import classNames from 'classnames';

type UsePrevNextButtonsType = {
  prevBtnDisabled: boolean;
  nextBtnDisabled: boolean;
  onPrevButtonClick: () => void;
  onNextButtonClick: () => void;
};

export const usePrevNextButtons = (
  emblaApi: EmblaCarouselType | undefined,
): UsePrevNextButtonsType => {
  const [prevBtnDisabled, setPrevBtnDisabled] = useState(true);
  const [nextBtnDisabled, setNextBtnDisabled] = useState(true);

  const onPrevButtonClick = useCallback(() => {
    if (!emblaApi) return;
    emblaApi.scrollPrev();
  }, [emblaApi]);
```

```
  const onNextButtonClick = useCallback(() => {
    if (!emblaApi) return;
    emblaApi.scrollNext();
  }, [emblaApi]);

  const onSelect = useCallback((emblaApi: EmblaCarouselType) => {
    setPrevBtnDisabled(!emblaApi.canScrollPrev());
    setNextBtnDisabled(!emblaApi.canScrollNext());
  }, []);

  useEffect(() => {
    if (!emblaApi) return;

    onSelect(emblaApi);
    emblaApi.on('reInit', onSelect);
    emblaApi.on('select', onSelect);
  }, [emblaApi, onSelect]);

  return {
    prevBtnDisabled,
    nextBtnDisabled,
    onPrevButtonClick,
    onNextButtonClick,
  };
};

type PropType = PropsWithChildren<
  React.DetailedHTMLProps<React.ButtonHTMLAttributes<HTMLButtonElement>, HTMLButtonElement>
>;

export const PrevButton: React.FC<PropType> = props => {
  const { children, ...restProps } = props;

  return (
    <button
      className={classNames(styles['embla__button'], styles['embla__button--prev'])}
      type="button"
      {...restProps}
    >
      <svg className={styles['embla__button__svg']} viewBox="0 0 532 532">
        <path
          fill="currentColor"
```

```
        d="M355.66 11.354c13.793-13.805 36.208-13.805 50.001 0 13.785 13.804 13.785 36.238 0 50.034L201.22 266l204.442
204.61c13.785 13.805 13.785 36.239 0 50.044-13.793 13.796-36.208 13.796-50.002 0a5994246.277 5994246.277 0 0 0-229.332-
229.454 35.065 35.065 0 0 1-10.326-25.126c0-9.2 3.393-18.26 10.326-25.2C172.192 194.973 332.731 34.31 355.66 11.354Z"
       />
     </svg>
     {children}
   </button>
 );
};

export const NextButton: React.FC<PropType> = props => {
 const { children, ...restProps } = props;

 return (
   <button
     className={classNames(styles['embla__button'], styles['embla__button--next'])}
     type="button"
     {...restProps}
   >
     <svg className={styles['embla__button__svg']} viewBox="0 0 532 532">
       <path
         fill="currentColor"
         d="M176.34 520.646c-13.793 13.805-36.208 13.805-50.001 0-13.785-13.804-13.785-36.238 0-50.034L330.78 266 126.34
61.391c-13.785-13.805-13.785-36.239 0-50.044 13.793-13.796 36.208-13.796 50.002 0 22.928 22.947 206.395 206.507 229.332
229.454a35.065 35.065 0 0 1 10.326 25.126c0 9.2-3.393 18.26-10.326 25.2-45.865 45.901-206.404 206.564-229.332 229.52Z"
       />
     </svg>
     {children}
   </button>
 );
};
```

File: database.rules.json

```
{
 "rules": {
   ".read": "auth != null",
   ".write": "auth != null"
 }
}
```

File: firebase.json

```
{
 "hosting": {
```

```
  "public": "build",
  "ignore": [
    "firebase.json",
    "**/.*",
    "**/node_modules/**"
  ],
  "rewrites": [ {
    "source": "**",
    "destination": "/index.html"
  }]
 }
}
```

File: package.json

```
{
 "name": "tic-tac-toe",
 "version": "0.1.0",
 "private": true,
 "dependencies": {
   "@types/node": "^16.18.79",
   "@types/react": "^18.2.55",
   "@types/react-dom": "^18.2.19",
   "classnames": "^2.5.1",
   "embla-carousel-react": "^8.0.0",
   "firebase": "^10.11.1",
   "react": "^18.2.0",
   "react-dom": "^18.2.0",
   "react-pure-modal": "^2.2.5",
   "react-router": "^6.22.0",
   "react-router-dom": "^6.22.0",
   "react-scripts": "5.0.1",
   "typescript": "^4.9.5",
   "zod": "^3.22.4"
 },
 "scripts": {
   "start": "react-scripts start",
   "build": "react-scripts build",
 },
 "eslintConfig": {
   "extends": [
     "react-app",
     "react-app/jest"
   ]
```

```
 },
 "browserslist": {
  "production": [
   ">0.2%",
   "not dead",
   "not op_mini all"
  ],
  "development": [
   "last 1 chrome version",
   "last 1 firefox version",
   "last 1 safari version"
  ]
 }
}
```

File: tsconfig.json

```
{
 "compilerOptions": {
  "target": "es5",
  "lib": [
   "dom",
   "dom.iterable",
   "esnext"
  ],
  "allowJs": true,
  "skipLibCheck": true,
  "esModuleInterop": true,
  "allowSyntheticDefaultImports": true,
  "strict": true,
  "forceConsistentCasingInFileNames": true,
  "noFallthroughCasesInSwitch": true,
  "module": "esnext",
  "moduleResolution": "node",
  "resolveJsonModule": true,
  "isolatedModules": true,
  "noEmit": true,
  "jsx": "react-jsx"
 },
 "include": [
  "src"
 ]
}
```

ДОДАТОК Г

(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«Розробка ігрового Web-застосунку для розвитку логічного мислення. Частина 2. Реалізація алгоритму гри та інтеграція з сервісом Firebase»

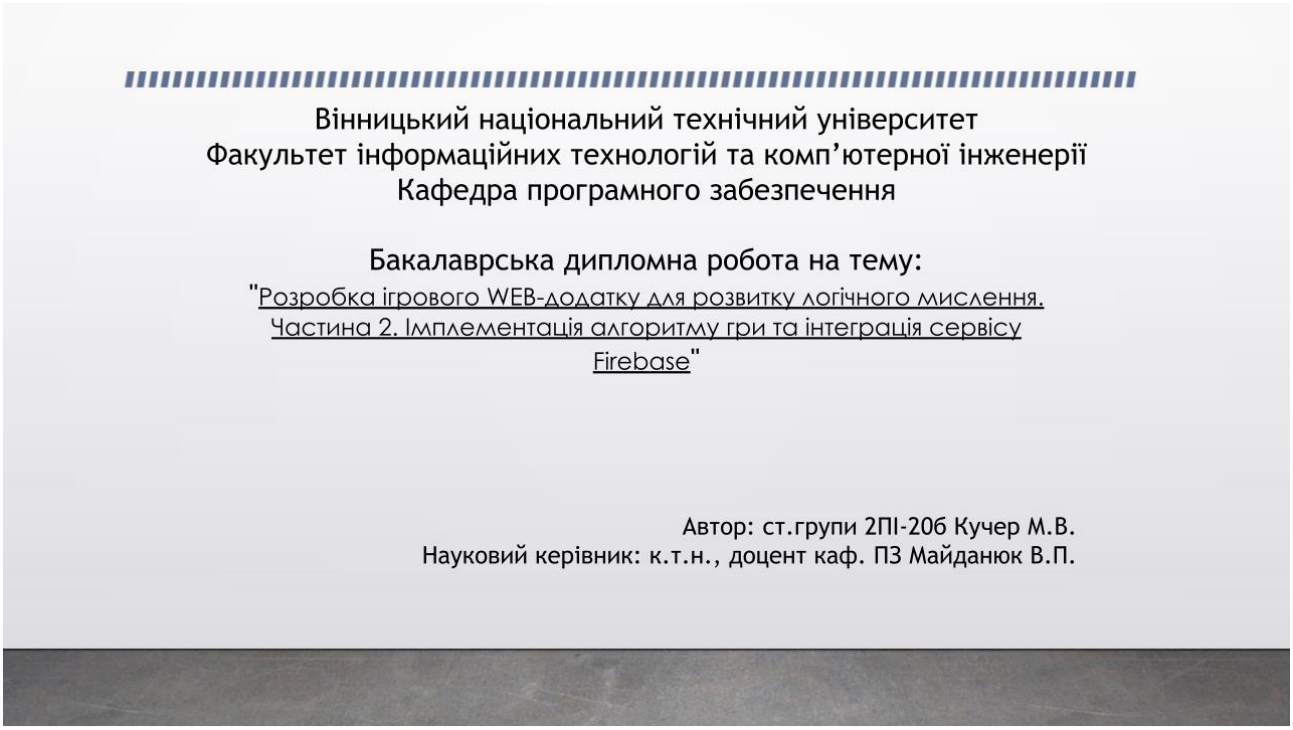


Рисунок Г.1 – Титульний слайд

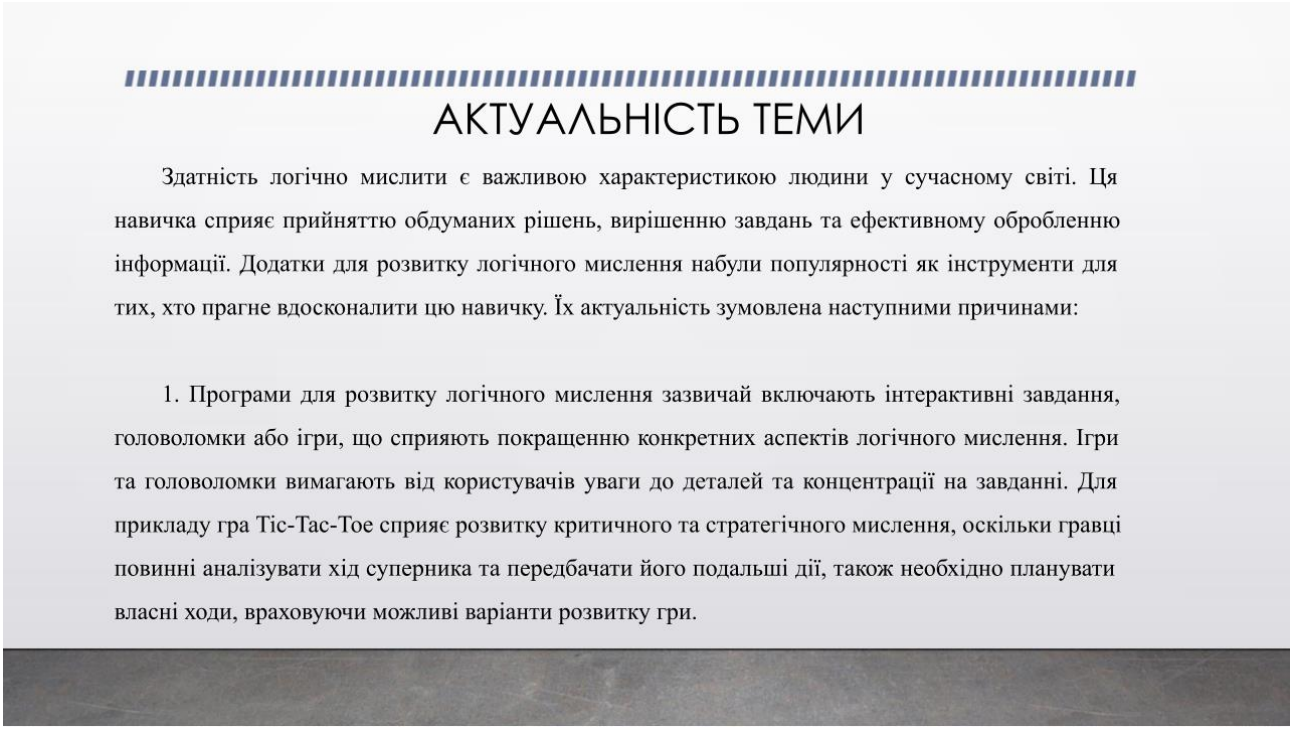


Рисунок Г.2 – Актуальність теми (1)

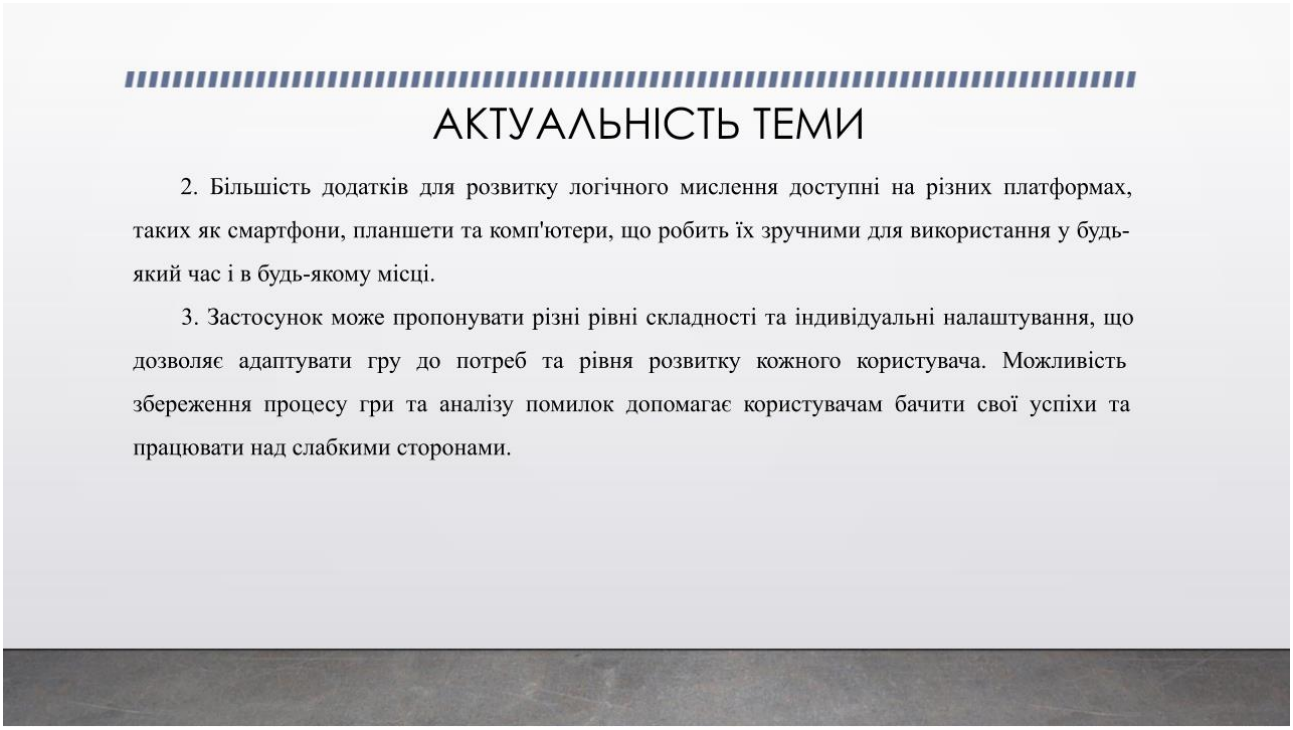


Рисунок Г.3 – Актуальність теми (2)

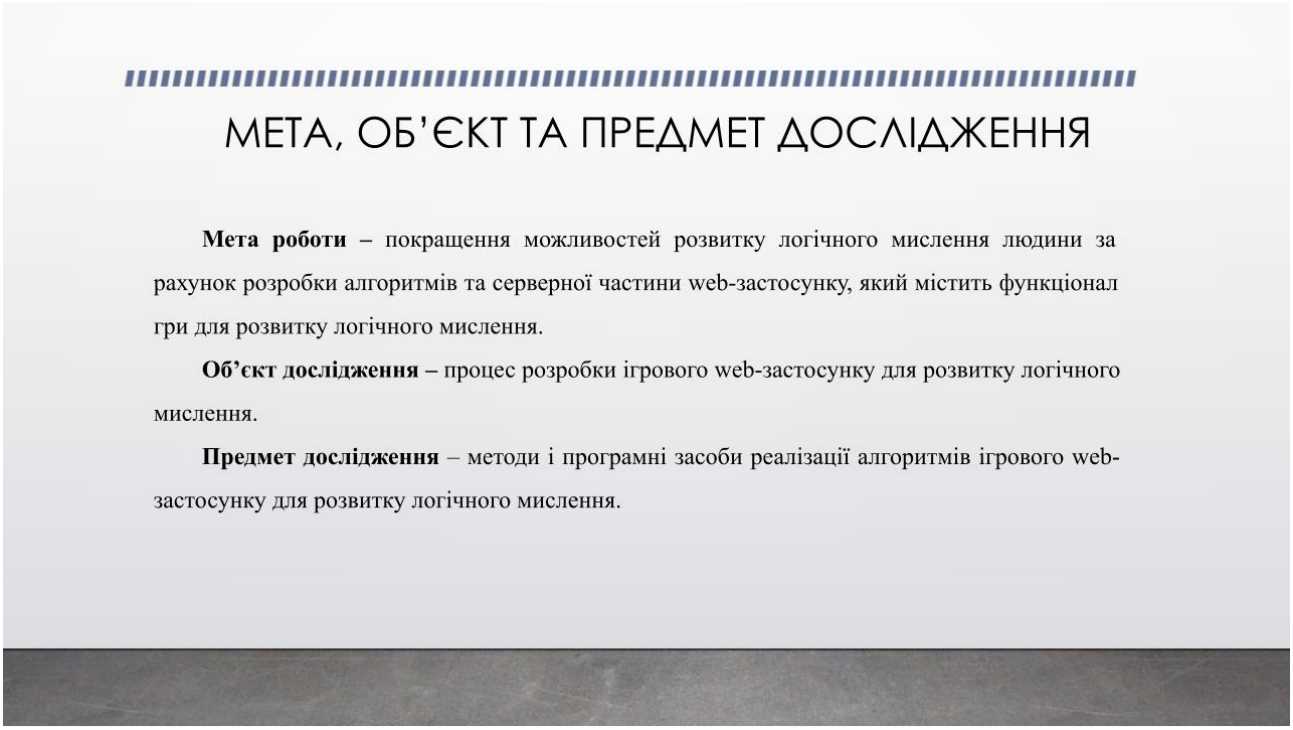


Рисунок Г.4 – Мета, об'єкт та предмет дослідження

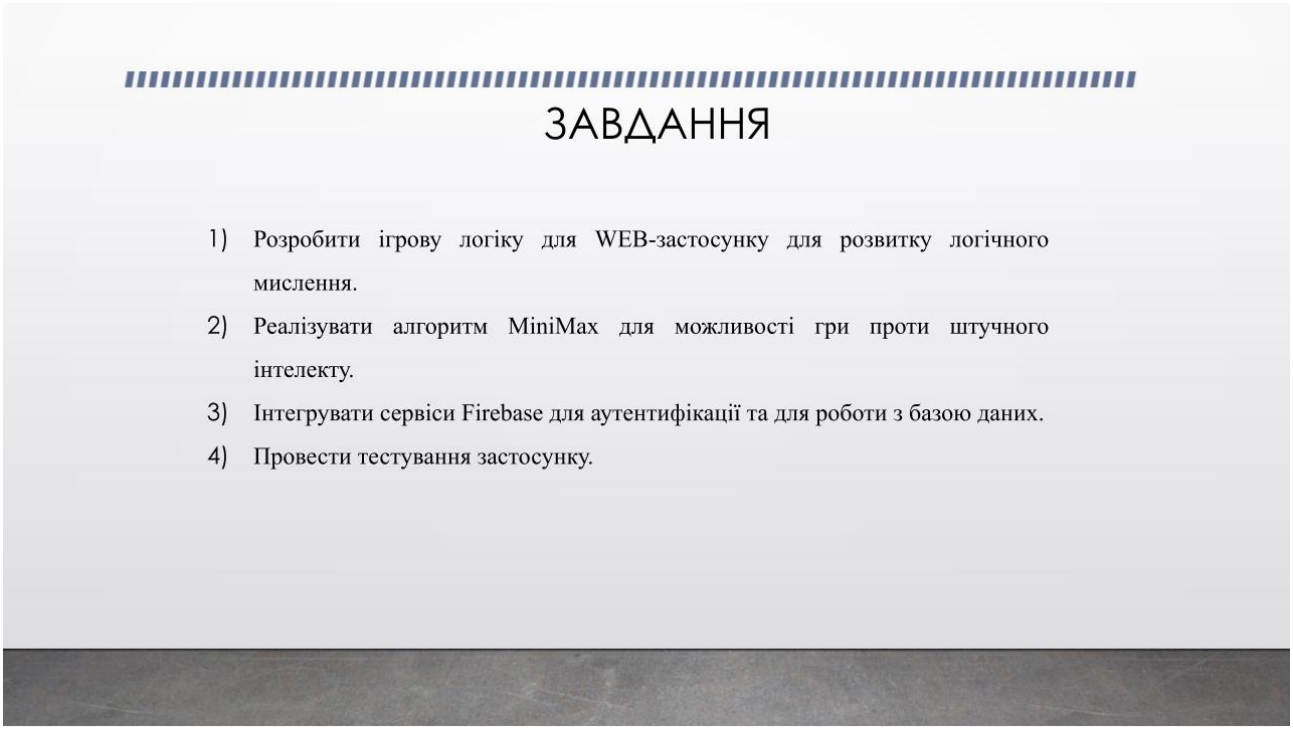


Рисунок Г.5 – Завдання

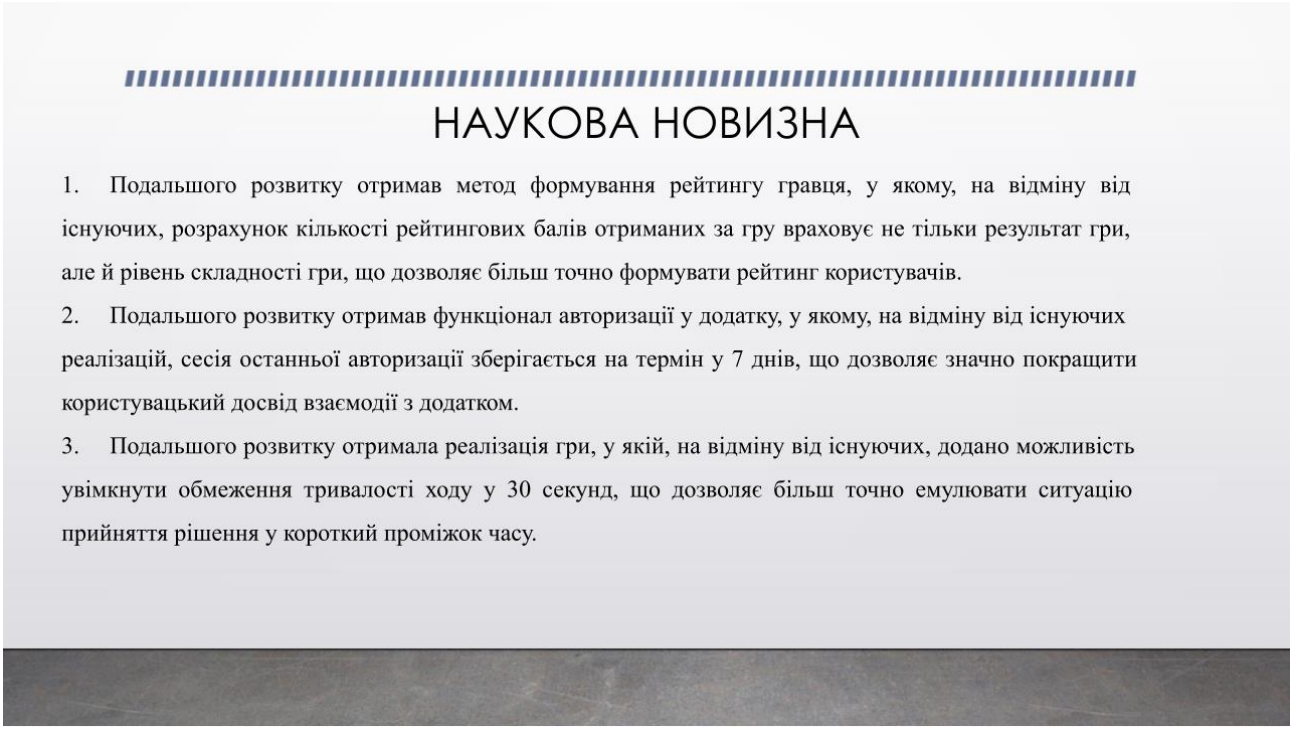


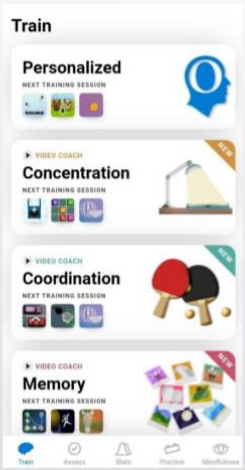
Рисунок Г.6 – Наукова новизна

ПРАКТИЧНА ЦІННІСТЬ

Практична цінність отриманих результатів Практична цінність одержаних в бакалаврській дипломній роботі результатів полягає у можливості використання розроблених на мові програмування Type-Script алгоритмів та коду серверної частини, що реалізовує інтеграцію з сервісом Firebase, у web-застосунку для розвитку логічного мислення.

Рисунок Г.7 – Практична цінність

АНАЛОГИ



Cognifit

Lumosity


Peak

Рисунок Г.8 – Аналоги

	CogniFit	Lumosity	Peak	Власна розробка
Адаптивний рівень складності	+	+	+	+
Режим змагань	+	-	-	+
Кросплатформеність	+	+	-	+
Відстеження прогресу	+	+	+	+
Впровадження штучного інтелекту	-	-	-	+
Сумарний коефіцієнт	4	3	2	5

Рисунок Г.9 – Порівняльний аналіз аналогів

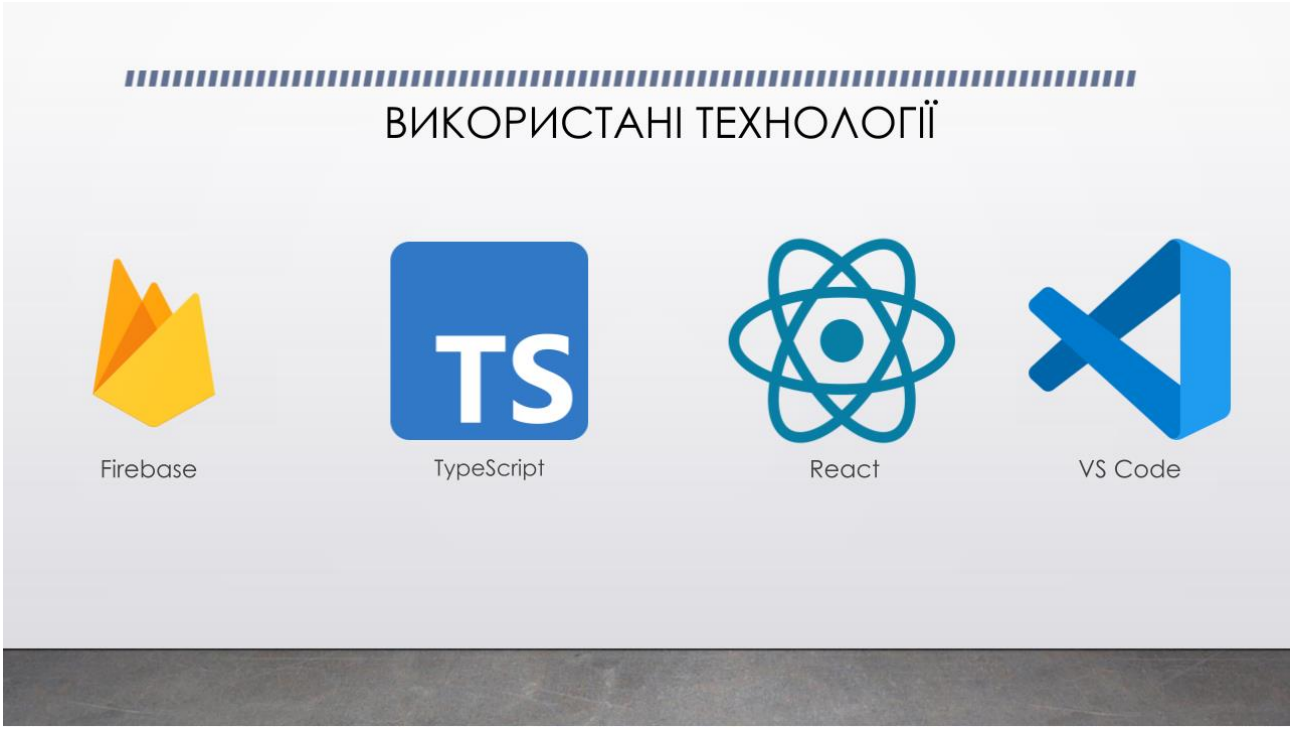


Рисунок Г.10 – Використані технології

Рисунок Г.11 – Алгоритм роботи ШІ (Minimax)

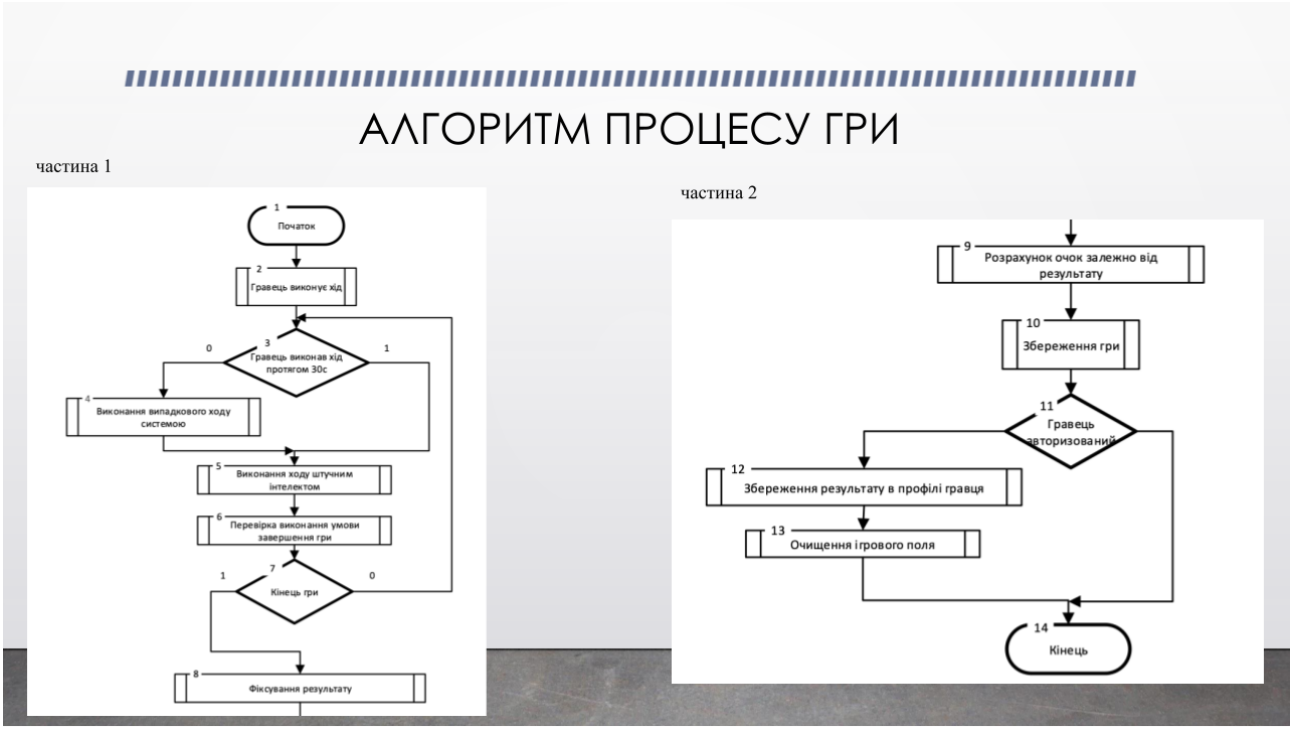


Рисунок Г.12 – Алгоритм процесу гри

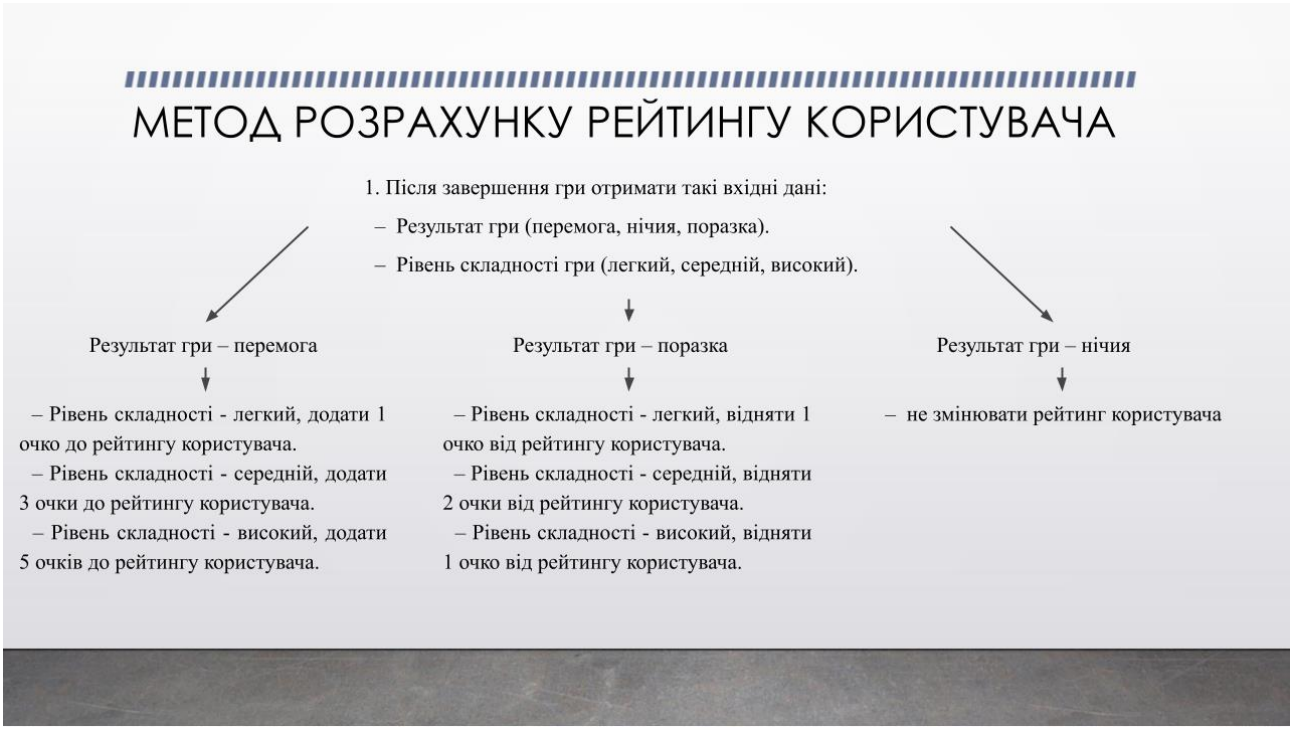


Рисунок Г.13 – Метод розрахунку рейтингу користувача

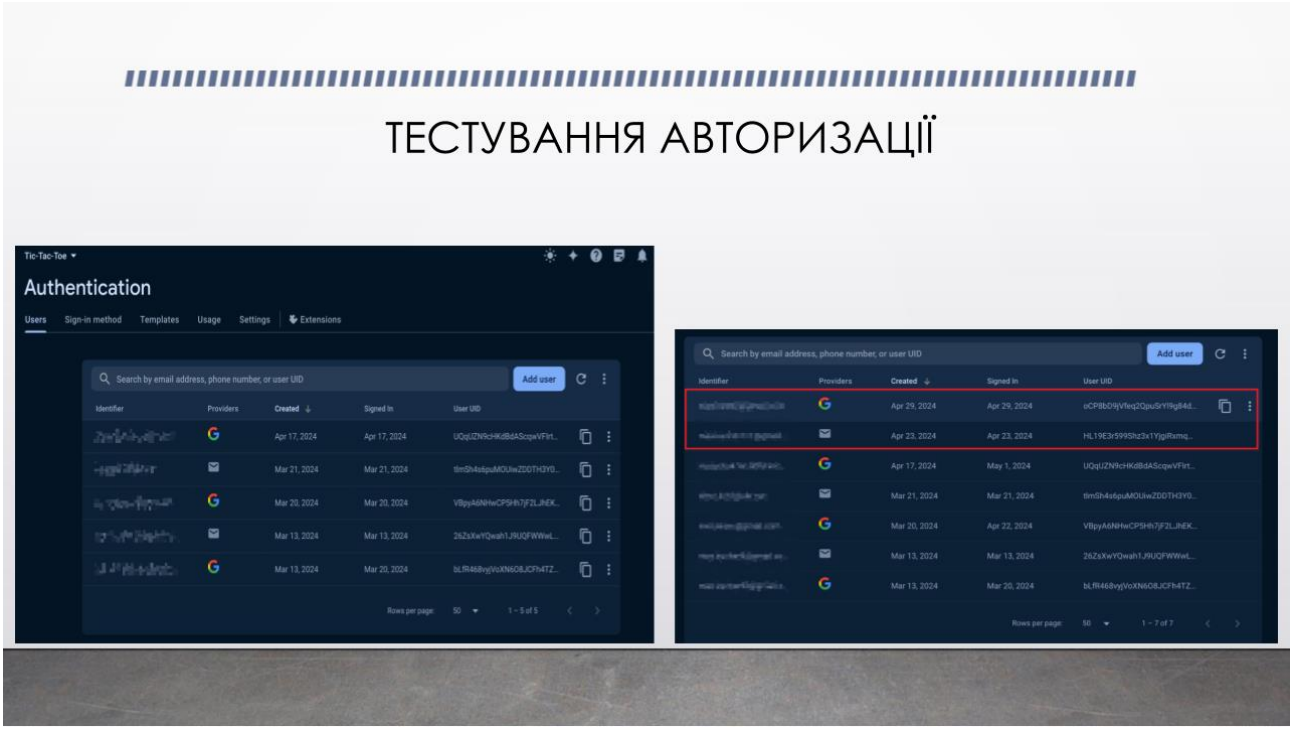


Рисунок Г.14 – Тестування авторизації

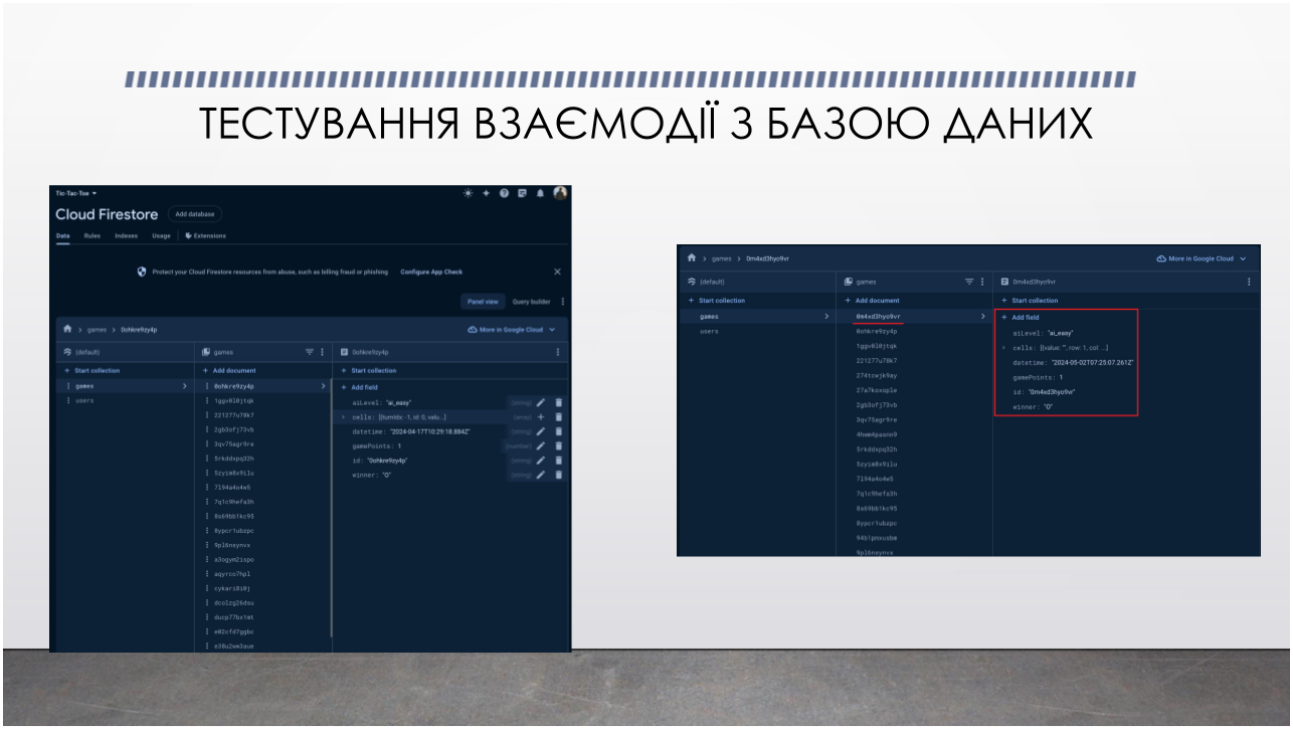


Рисунок Г.15 – Тестування взаємодії з базою даних

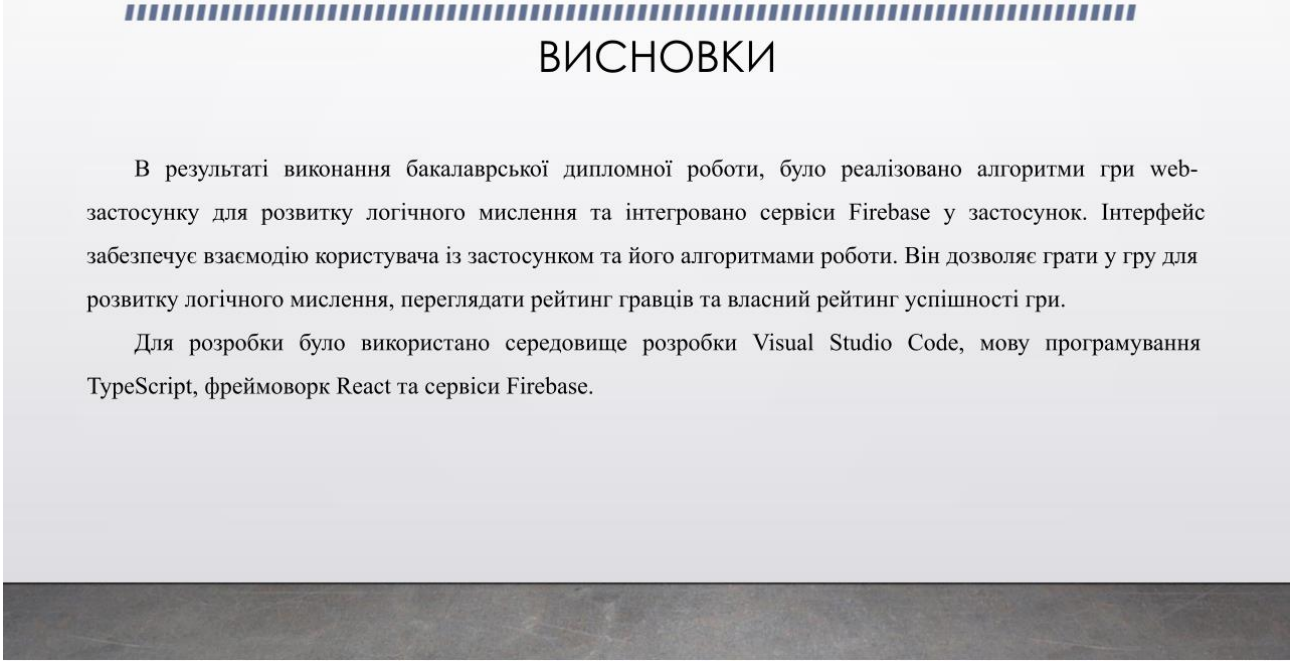


Рисунок Г.16 – Висновки (1)

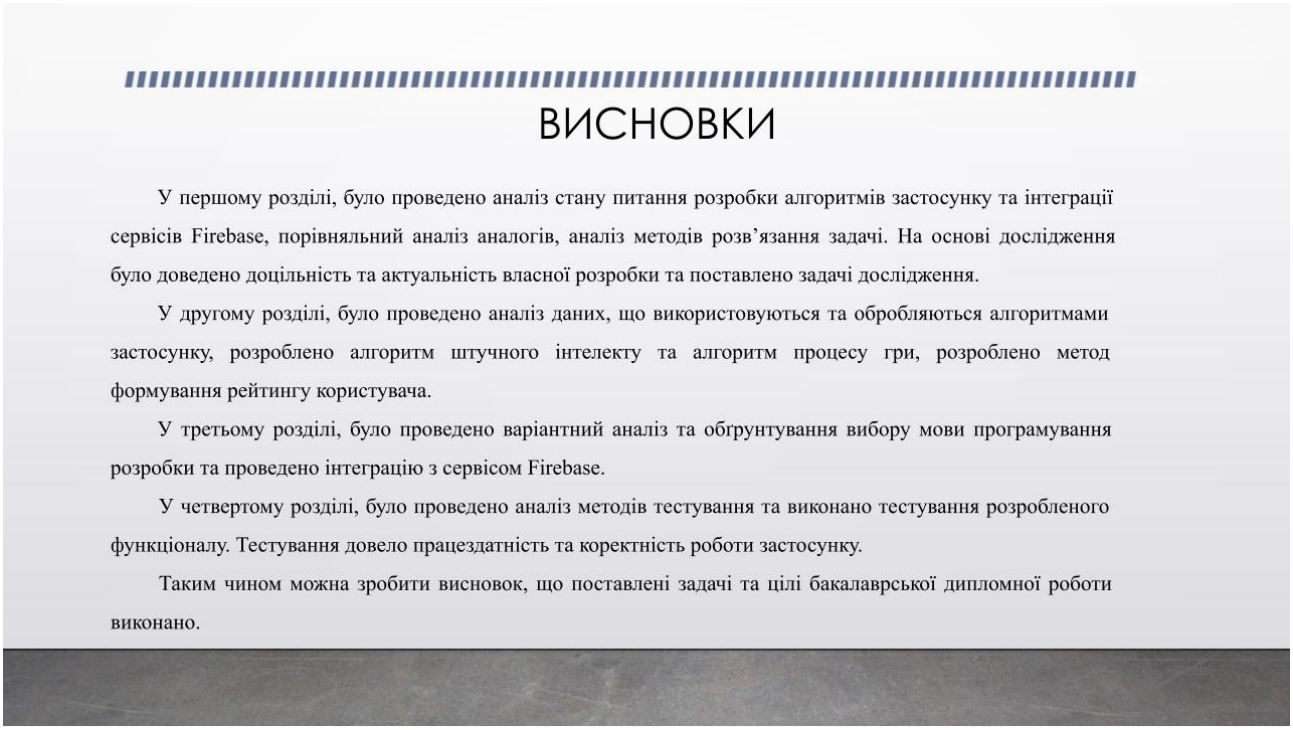


Рисунок Г.17 – Висновки (2)

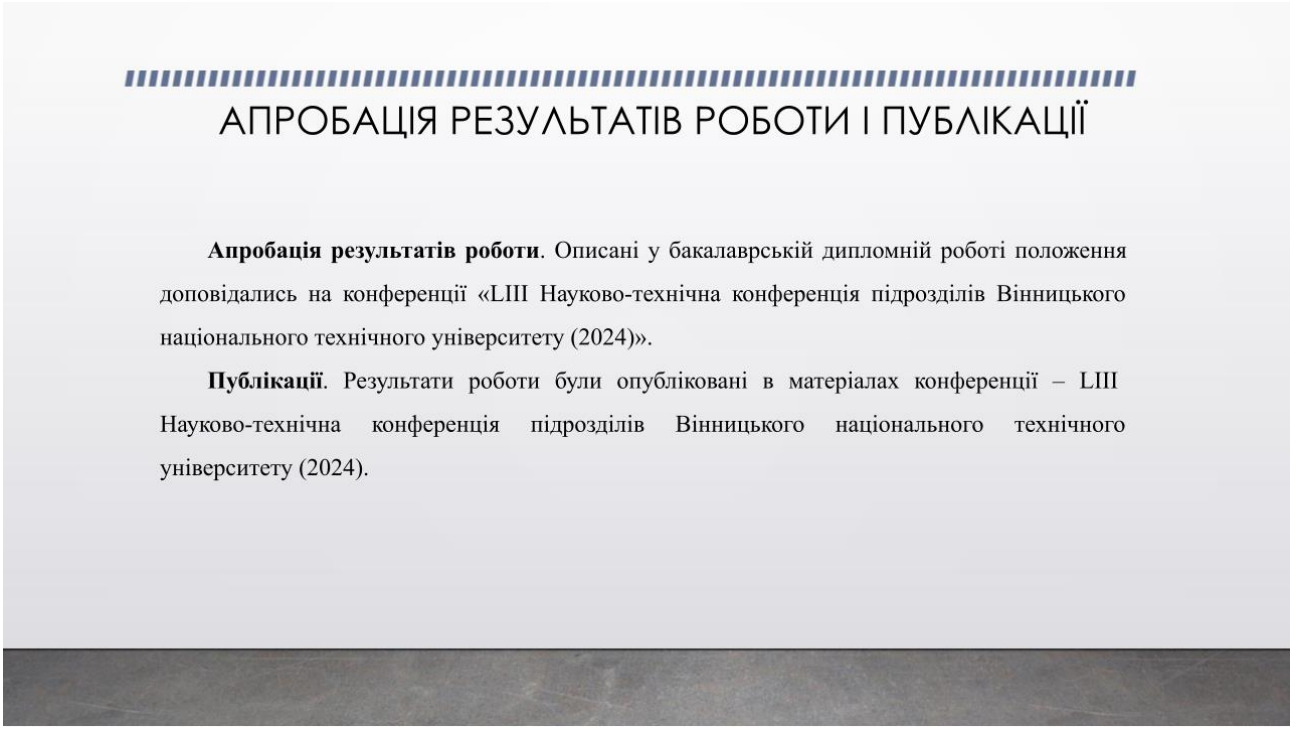


Рисунок Г.18 – Апробація результатів роботи і публікації

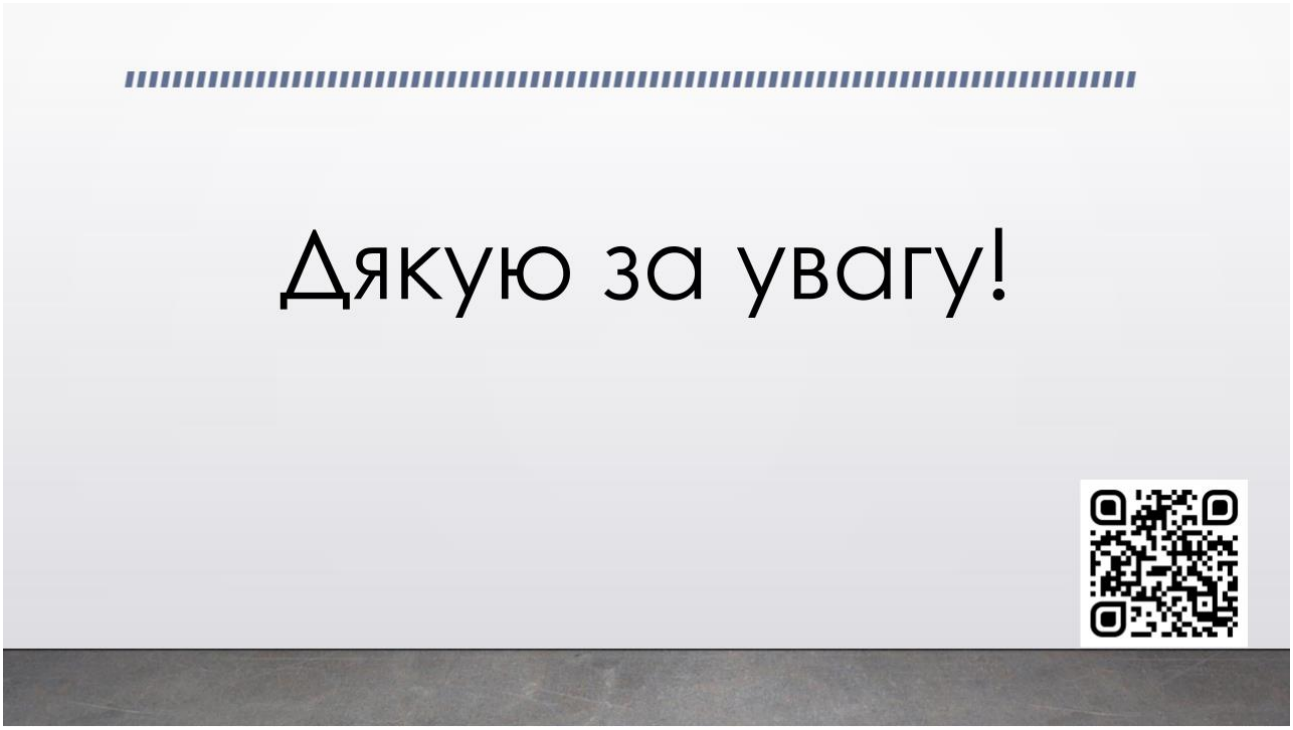


Рисунок Г.19 – Завершальний слайд