Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

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

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи
перший (бакалаврський)
(рівень вищої освіти)

на тему: **«Розроблення мобільної 2D-гри "Words" засобами Marmalade SDK та C++»**

Виконав: студент **4** курсу, групи **КН - 41**
Спеціальності 122–"Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)
Кубіцький К. А.
(прізвище та ініціали)
Керівник: **Капран І. Д., Процик Ю. С.**
(прізвище та ініціали)
Рецензент: Покотилюк О.В.
(прізвище та ініціали)

Львів – 2025 р.

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут *комп'ютерних наук та інформаційних технологій*
Кафедра *комп'ютерних наук*
Рівень вищої освіти *перший (бакалаврський)*
Спеціальність *122 – "Комп'ютерні науки"*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
_____ Борецька І. Б.
"10" червня 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Кубіцькому Костянтину Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *«Розроблення мобільної 2D-гри "Words" засобами Marmalade SDK та C++»*.
керівник проекту (роботи) *Капран Ігор Дмитрович, старший викладач кафедри комп'ютерних наук, Процик Юрій Степанович, доцент кафедри комп'ютерних наук, кандидат фізико-математичних наук*.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджена наказом вищого навчального закладу від " *15* "*листопада* *2024* р. № *С-882*

2. Термін подання студентом проекту (роботи) *10 червня 2024 року*

3. Вихідні дані до проекту (роботи) *Аналіз шляхів вирішення поставлених задач, організаційна структура застосунку, огляд алгоритмів та програмних засобів для розроблення мобільного застосунку*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *Вступ., Розділ 1. Стан проблемної області., Розділ 2. Інформаційне та математичне забезпечення., Розділ 3. Програмне та технічне забезпечення., Висновки., Список використаних джерел., Додатки.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
слайди для доповіді (підготовка матеріалу загальним обсягом 10-15 слайдів)

6. Дата видачі завдання *18 листопада 2024 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1.	Системний аналіз стану проблемної області. Огляд літературних джерел згідно досліджуваної теми. Збір потрібних матеріалів. Формування функціональних вимог та постановка задачі проєкту.	*18. 11. 2025 р.* *28. 02. 2025 р.*	*Виконано*
2.	Огляд сучасного стану проблемної області. Оформлення першого розділу пояснювальної записки.	*01. 03. 2025 р.* *12. 03. 2025 р.*	*Виконано*
3.	Написання другого розділу. Аналіз математичного забезпечення.	*16. 03. 2025 р.* *29. 03. 2025 р.*	*Виконано*
4.	Оформлення третього розділу пояснювальної записки. Програмна реалізація	*02. 04. 2025 р.* *12. 04. 2025 р.*	*Виконано*
5.	Оформлення висновків пояснювальної записки. Формування апаратного забезпечення.	*13. 05. 2025 р.* *19. 05. 2025 р.*	*Виконано*
6.	Оформлення пояснювальної записки та здача на рецензування.	*23.05.2025 р.* *10.06.2025 р.*	*Виконано*

Студент _____ *Кубіцький К. А.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Капран І. Д.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Процик Ю. С.*
(підпис) (прізвище та ініціали)

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити та програмно реалізувати на основі комп'ютерних технологій мобільну 2D-гру «Words». У грі повинні дотримуватися принципи інформаційної закритості, модульності та ієрархічної організації. Дана мобільна гра повинна містити в собі реалізацію найбільш поширених алгоритмів вирішення завдання.

Розроблена гра повинна забезпечувати виконання наступних основних функцій:

- велика кількість рівнів;
- ергономічний дизайн;
- зручне управління процесом гри;
- підтримка книжкової та альбомної орієнтації екрану;
- можливість налаштування розміру шрифту відгаданих слів;
- швидкість обробки даних та рішення;
- збереження рішення гри в масиві даних;
- точність та якість результату;
- простий і зрозумілий для користувача інтерфейс та меню.

За результатами проведеної роботи сформувати пояснювальну записку, яка крім теоретичної частини має містити в собі детальний опис виконання кожного кроку завдання.

АНОТАЦІЯ

Бакалаврська дипломна робота (проект): пояснювальна записка: 55 стор., 21 рис., 2 додатки, 14 джерел.

В даній дипломній роботі розроблена мобільна 2D-гра «Words». Описані основні етапи розроблення застосунку під мобільні операційні системи. Розглянуті базові поняття програмування. Проаналізовані основні алгоритми вирішення поставлених завдань та їх комбінації.

Наведена програмна реалізація даної мобільної гри написана за допомогою Marmalade SDK на мові програмування C++.

Ключові слова: Marmalade SDK, C++, UML-діаграма, Oracle, online-словник.

ABSTRACT

Bachelor's thesis (project): explanatory note: 55 pages, 21 figures, 2 appendices, 14 sources.

This thesis develops a mobile 2D game "Words". The main stages of developing an application for mobile operating systems are described. The basic concepts of programming are considered. The main algorithms for solving the tasks and their combinations are analyzed.

The software implementation of this mobile game is given, written using the Marmalade SDK in the C++ programming language.

Keywords: Marmalade SDK, C++, UML diagram, Oracle, online dictionary.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

iOS - це власницька мобільна операційна система від Apple;

Marmalade SDK - кроссплатформне SDK від компанії Ideaworks3D Limited. Являє собою набір бібліотек, зразків, інструментів і документацій, необхідних для розробки, тестування і розгортання додатків для мобільних пристроїв;

Visual Studio - інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів;

БД (База даних) - сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами;

Oracle - система керування базою даних;

UML-діаграма - діаграма, що визначає зміну станів об'єкта у часі, одна з діаграм моделювання поведінки в UML;

Бекенд – це внутрішня, прихована від користувача начинка сайту або веб-програми. Іншими словами, це частина сервісу, яка працює на віддаленому сервері, а не у браузері чи персональному комп’ютері.

ВСТУП

У нашому світі, що стрімко розвивається, легко сконцентруватися виключно на технологічних досягненнях і забути про важливість людських зв'язків. Однак важливо пам'ятати, що незважаючи на всі досягнення у галузі комунікаційних технологій, ніщо не може замінити особистого контакту та розуміння, які виникають при спілкуванні з кимось його рідною мовою. Вивчення іноземної мови дозволяє знаходити спільну мову з людьми різних культур і будувати міцніші зв’язки та відносини.

Існує безліч ігор, де гравцеві потрібно шукати слова з певного набору букв. Мета гравця вгадати ці слова шляхом введення їх у спеціальне поле. Гравець, який першим набрав загадане слово (або його словоформу), отримує його вартість на свій гаманець або мобільник. Цікава гра, що тренує асоціативне мислення та вміння будувати зв'язки.

Об’єктом дослідження є логічна, мобільна 2D-гра «Words».

Предметом дослідження – інформаційні технології та алгоритми, які використовуються для розроблення логічних ігор під мобільні пристрої.

Метою роботи є засвоєння процесу розроблення програмних продуктів на прикладі реалізації логічної, мобільної 2-D гри «Words».

Актуальність роботи полягає у тому, що розв’язання даного виду головоломок з допомогою мобільного пристрою зменшує час необхідний для реалізації даного завдання, покращує точність та якість отриманих результатів.

Практична значимість роботи полягає в тому, що розроблена мобільна гра «Words» тренує асоціативне мислення та вміння будувати зв'язки. Гра розвиває інтелектуальні навички і збільшує словниковий запас користувача.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд словесних ігор

Зі словесних сучасних відеоігор як референси можна розглянути *Bookworm Adventures* і *Letter Quest: Grimm's Journey* (рис. 1.1, рис. 1.2). Але вони доступні лише однією мовою. І в них правилами не заборонено:

- слова у множині;
- слова, що повторюються, але які відрізняються закінченнями;
- прикметники;
- дієслова і т. д.

У моєму ж розумінні словесні ігри повинні бути більш схожі на настільні, в яких дозволяється використання тільки загальних іменників в однині і називному відмінку.

Рисунок 1.1 – Словесна відеогра Bookworm Adventures

Рисунок 1. 2 – Гра Letter Quest: Grimm's Journey

Балда - настільна гра, у якій на полі додаються літери так щоб утворювалися нові слова. Складаються вони у вигляді переходів від букви до букви по прямій чи під прямим кутом у напрямі, а бали нараховуються за довжину складеного слова. Начебто вже починає вимальовуватись механіка схожа на матч-три [1].

Скреббл (Ерудит) - чимось схожа на попередню, але за один хід можна викладати скільки завгодно букв, що є на руках, і враховуються все нові слова утворені в цьому ході. Слова читаються виключно зліва направо та зверху вниз, і чим більше слів та перетинів, тим більше очок за хід можна заробити (кожна буква, залежно від рідкості, коштує певну кількість очок). З таким типом побудови слів гра найбільше схожа на кросворд (по суті ще одна словесна гра, що заслуговує на увагу як референс, але, мабуть, не вимагає окремого представлення). І це явно підказує нам, що у грі потрібно дотримуватись правил читання слів зліва направо та зверху вниз (рис. 1. 3).

Рисунок 1. 3 – Настільна гра Скреббл (Ерудит)

Боггл - настільна гра, в якій потрібно з наявних літер на квадратному полі скласти якнайбільше слів за відведений час, використовуючи осередки з буквами, що стикаються по вертикалі, горизонталі і діагоналі. За структурою поля вона, звичайно ж, більше за всіх інших схожа на матч-три (рис. 1. 4).

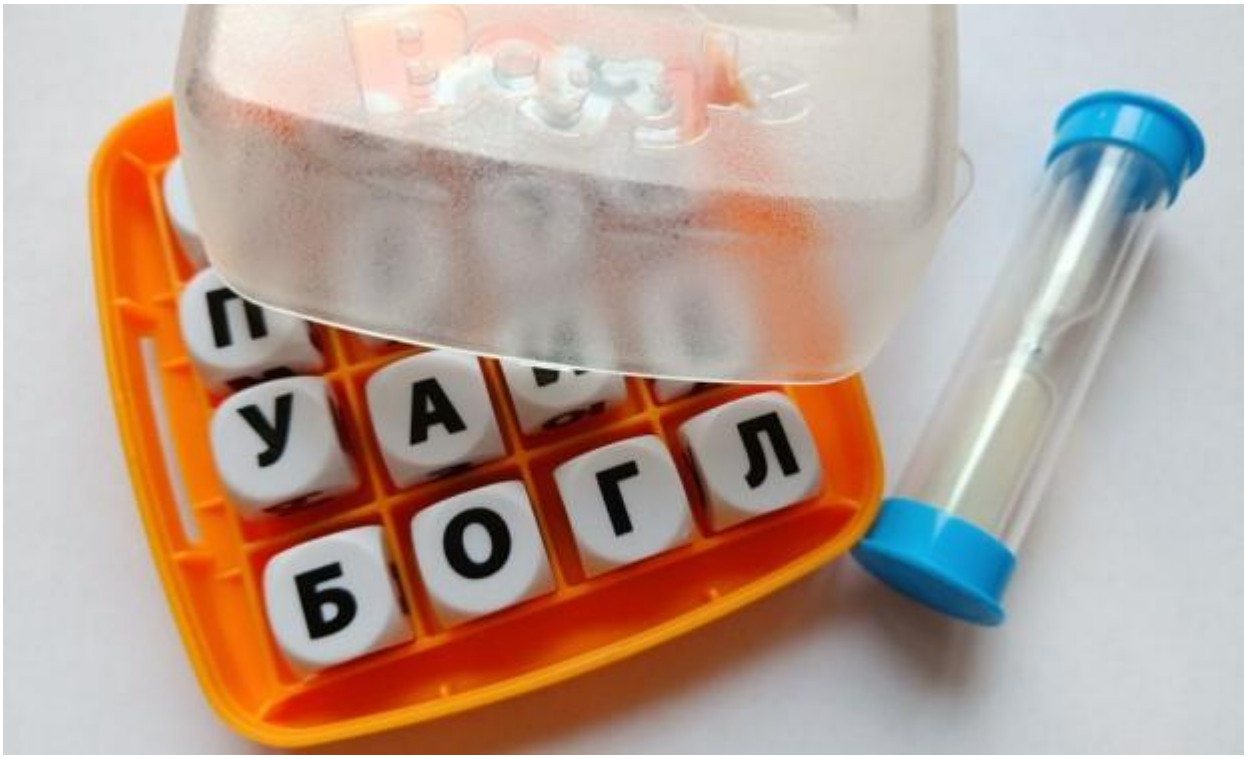

Рисунок 1. 4 – Настільна гра Боггл

Отже, ідея є, референси вивчені, механіки намічені, можна починати! Але творчий процес запущено, шестерні крутяться, і вже не зупинитися… Якоїсь миті з'явилася ідея відмовитися від ігрового поля та матч-три системи та зробити щось схоже на гру Zuma, але зі словами (рис 1. 5).

Правда тут з'являється купа питань про підбір букв - мало просто дати майданчик для гри, треба бути впевненим, що буде у що пограти, тобто, що слова можна буде скласти. У варіанті поля для матч-три ймовірність позитивного результату вища, тому що власне більше букв за рахунок розміру та великої кількості перестановок.

У результаті в моєму уявленні має вийти гра на буквеному полі з механіками переміщення букв як у матч-три та зі спрощеними правилами зчитування готових слів – лише по горизонталі зліва направо та по вертикалі зверху вниз. Це свого роду спроба відволікти гравців від матч-три складнішою механікою та можливістю поламати мізки.

Зрештою, це стало помилкою з погляду на завдання залучення аудиторії для ігор, так як вся краса словесних ігор в тому, щоб не напружуватися: відносно прості завдання і можливість перерватися в будь-який момент.

Рисунок 1. 5 – Гра Zuma

Існує безліч ігор, де гравцеві потрібно шукати слова з певного набору букв. Ось дві найпопулярніші з них:

1. 4 фото 1 слово (4 Pics 1 Word) в AppStore, Google Play; Ця гра має багато реалізацій, але ідея у всіх одна.
2. Словоманія (Wordsmania) AppStore, Google Play.

Суть першої гри: дано 4 картинки, довжина слова, що вгадується, і набір літер, вибирати літери можна в будь-якому порядку (рис. 1. 6).

Суть другої зводиться до того, що на полі 4х4, заповненому літерами, необхідно знайти якнайбільше слів, з кожної клітини можна пересуватися в наступну по вертикалі, горизонталі та діагоналям (рис. 1. 7).

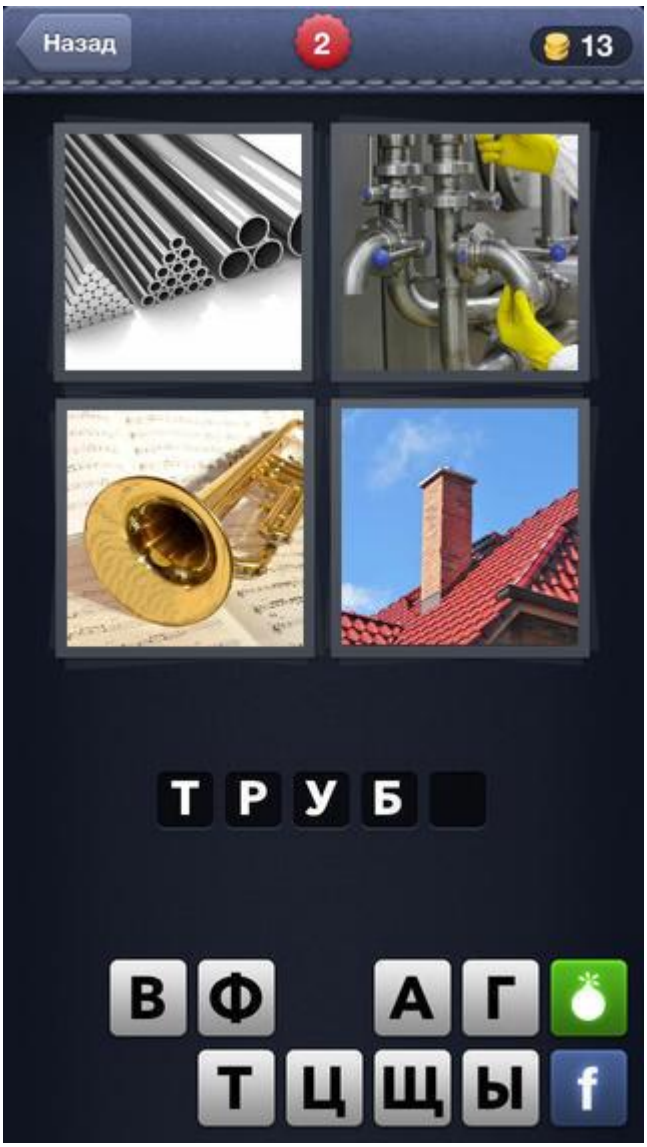

Рисунок 1. 6 – Гра 4 Pics 1 Word

Рисунок 1. 7 – Гра Словоманія (Wordsmania)

1.2. Гра Wordle

Wordle - браузерна гра, в якій загадується слово з 5 літер, яке гравець має відгадати за 6 спроб [2]. Після кожної спроби відгадати загадане слово літери слова-спроби позначаються одним із трьох кольорів:

- Зелений - правильна літера на правильному місці у слові;
- Жовтий - буква є у слові, але на іншій позиції;
- Сірий - буква в слові відсутня.

Грати можна 1 раз на день (альтернативний варіант без обмежень). Загадане слово за один день однакове для всіх гравців (рис. 1. 8).

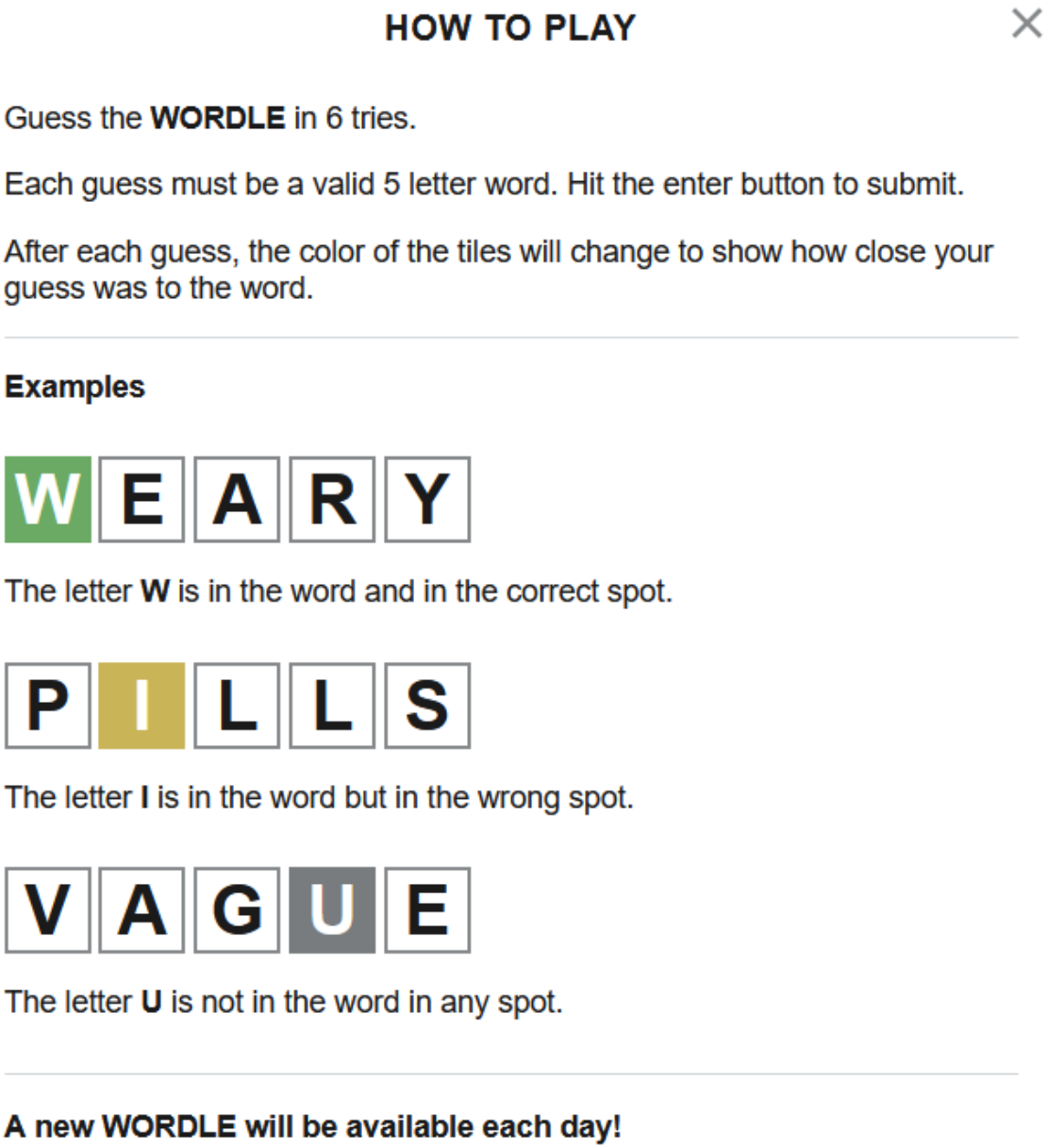

Рисунок 1. 8 – Правила гри Wordle

Крім основних правил, є неочевидні та приховані особливості:

- Усі слова, які можуть бути загадані, знаходяться у відкритому вигляді у коді гри. Список можливих слів загадок включає 2315 слів, спочатку підібраних людиною.
- За вихідним кодом гри легко визначити яке слово загадано сьогодні. Слова в вищезгаданому списку розташовані в порядку днів, коли вони будуть загадані. Знаючи слово, загадане вчора, можна знайти його у списку, наступне за ним слово буде сьогоднішньою загадкою.
- В якості слова-спроби можна використовувати лише слова із заздалегідь визначеного списку з 13 тисяч слів. Список у відкритому вигляді також знаходиться у вихідному коді гри. Якщо слово-спроба не входить до списку, то його гра не прийме.

Крім основного режиму гри, є ще складний режим, в якому слова, які вводяться для вгадування, обов'язково повинні використовувати раніше отримані зелені та жовті підказки. Аудиторія гри – кілька мільйонів гравців. 2022 року The New York Times купив гру за семизначну суму в доларах [3].

1 лютого 2022 року британський розробник Джош Вордл (Josh Wardle) розповів, що продав права на гру Wordle виданню New York Times. Браузерна версія головоломки залишиться безкоштовною для нових та існуючих гравців. Автор гри не розкрив суму угоди, але пояснив, що вона знаходиться у нижньому діапазоні семизначних сум. New York Times планує вбудувати Wordle у свою окрему підписку, де вже є різні кросворди та головоломки.

Wordle - це проста гра, в якій гравці мають шість спроб вгадати слово з п'яти літер. Кожне припущення перетворюється на підказку: літери слова підсвічуються різними кольорами. Якщо буква підсвічена сірим, її немає в загаданому слові. Жовтим підсвічені літери, які є, але стоять не на своєму місці. Позначені зеленим літери є у відповіді та стоять на правильних місцях. Основна фішка гри - своїми досягненнями гравці діляться між собою, чим підвищують її впізнаваність. І це відбувається без спойлерів слова дня.

Уордл придумав і запустив Wordle в жовтні минулого року без реклами і широкого розголосу 1 листопада у гри було всього 90 користувачів. На початку січня 2022 року у неї грали до 300 тис. користувачів на день. Наприкінці січня слова у браузері розгадували кілька мільйонів людей по всьому світу. Уордл із приводу продажу гри розповів, що він не очікував від неї такого успіху, а для розвитку цього проекту тепер грі потрібно виходити на новий рівень. Уордл пообіцяв, що історія ігор кожного користувача та всі їхні досягнення збережуться після переходу гри до New York Times.

У середині січня 2022 року Apple почистила App Store від клонів браузерної гри Wordle. Уордл не планував і не випустив мобільну версію гри, тому що хотів, щоб гра не відволікала і не вимагала великої уваги.

Наприкінці січня Twitter заблокувала робота Wordlinator за порушення правил платформи. Робот відповідав на пости користувачів грубостями і спойлерував задумане слово в Wordle [4].

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Середовище розробки Marmalade SDK

Компанія-партнер Tizen Association - Marmalade Technologies Ltd. днями представила повністю безкоштовну ліцензію для свого крос-платформного SDK, який на даний момент є світовим лідером і дає можливість розробникам додатків та ігор з усього світу здійснювати творчі задуми та комерційні ідеї, не обираючи між якістю та продуктивністю на всіх мобільних платформах (рис. 2. 1).

Рисунок 2. 1 - Marmalade Technologies Ltd.

Існуючі клієнти Marmalade також мають користь з комплексної програми нововведень: у кожній ліцензії тепер доступні платформи Tizen, BlackBerry, Windows Phone і Windows Store, крім того, Windows і Mac десктопи, Roku і деякі Smart TV платформи доступні для користувачів ліцензій Indie, Plus і Pro.

Безкоштовна ліцензія Marmalade дозволить розробникам створювати програми та ігри для Tizen, iOS, Android, Windows Phone, Windows Store та BlackBerry, використовуючи єдину кодову базу, та забезпечить рідну продуктивність для всіх пристроїв. Так само, як і з відомою Marmalade C++ SDK, всі розробники Marmalade

мають доступ до Marmalade Quick, Marmalade Web і Marmalade Juice - вони дають можливість перекомпілювати Objective C IOS проекти нативно під Android.

Крім цього, безкоштовна ліцензія Marmalade відрізняється широким спектром інтегрованих розширень, що включають білінг і IAP API для всіх основних аппсторів, а також сервіси, що лідирують - рекламні, соціальні, ігрові, сервіси графіки, фізики, аналітики та баз даних.

Продуктивність додатків є найважливішим питанням для розробників під час розгляду крос-платформної стратегії, як свідчить останнє дослідження, проведене фахівцями ринку додатків Research2Guidance. У їхньому останньому звіті «Cross Platform Tool Benchmarking 2023» Research2Guidance назвали Marmalade найкращим за продуктивністю серед усіх крос-платформних рішень, вказуючи, що 88% користувачів Marmalade оцінили продуктивність додатків, створених за допомогою Marmalade SDK, як рівну або найкращу порівняно з нативною розробкою (рис. 2. 2).

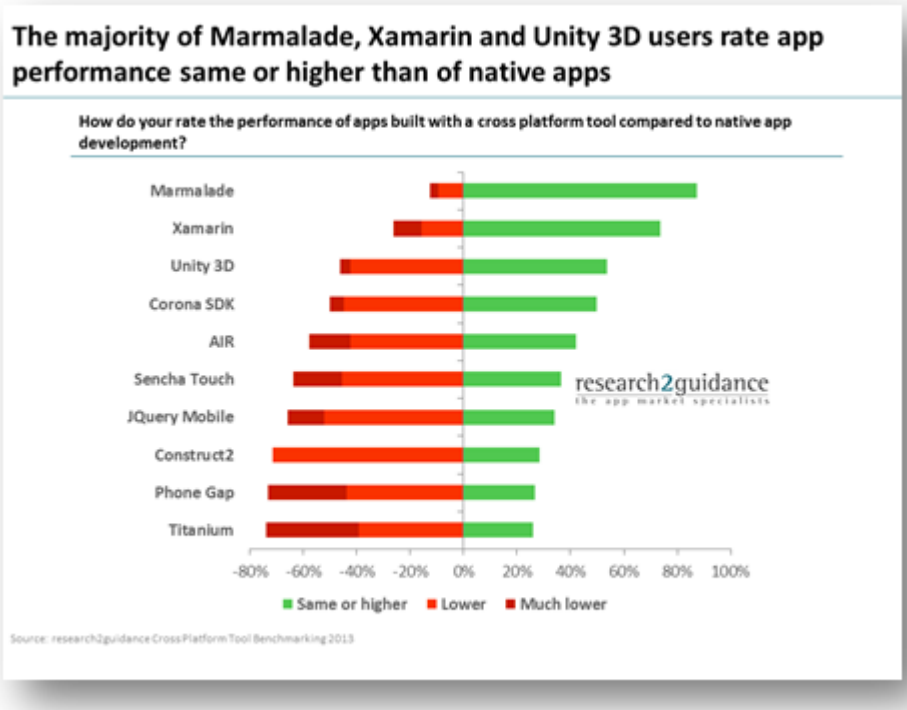

Рисунок 2. 2 – Оцінка продуктивності платформ розробниками

Marmalade SDK як середовище для розробки кросплатформових мобільних (і не тільки) додатків надає розробнику C++ API. По суті, це набір розширень (Extensions), кожне з яких містить у собі конкретну реалізацію функціоналу (робота

з графікою, файловою системою, мережею, UI, внутрішньоігрові покупки, робота з аудіо-відео тощо. д.) під кожну окрему платформу (Android, iOS, WinPhone та багато інших платформ).

Тому розробнику в процесі написання практично немає необхідності зав'язуватись на особливості тієї чи іншої платформи, за винятком деяких випадків (можливо список неповний):

- ✓ якийсь функціонал може не підтримуватися у певній ОС;
- ✓ розробник сам вирішив реалізувати логіку застосування по-різному в залежності від ОС.

Але ось у випадку, якщо розробнику буде потрібний функціонал, який відсутній у стандартному дистрибутиві Marmalade SDK, йому може знадобитися самому зібрати свій Extension зі своєю реалізацією під кожну платформу — а значить писати платформозалежний код.

Особисто мені для реалізації цієї програми вистачило стандартного набору розширень, тому платформозалежного коду у мене немає. Більшість тестування і налагодження (приблизно 90%) було зроблено на вінді на симуляторі (інші 10% — це налагодження масштабування на пристрої - тому що для цього потрібні zoom-жести двома пальцями). Білди під Android та iOS також збираються на вінді (для цього потрібно поставити відповідні інструменти, описані в доку мармеладу). Мак потрібен лише для заливки ipa-файлу в консоль iTunesConnect через еппловский ApplicationLoader.

Marmalade надає API для розробки крос-платформних програм, дозволяючи збирати їх, у тому числі, під Android та iOS. Робота в Marmalade досить комфортна, а його довідкова система супроводжується великою кількістю прикладів, але процес розробки носить досить низькорівневий характер. Використання готового Framework-а може сильно полегшити життя розробнику-початківцю.

Ні для кого не є секретом, що сьогодні мобільні ігри дуже популярні. Можливість написати одну з таких ігор є у кожного розробника, навіть початківця. Часто виникає питання щодо вибору платформи. Звичайно, хочеться, щоб гра була

одразу скрізь: на iOS та Android, на WP7 та MeeGo, на десктопі та у браузері. І щоб все це можна було легко реалізувати за допомогою безкоштовних інструментів.

2.2. Алгоритм для визначення рангів слів

Основний алгоритм, який буде використовуватися у грі — сортування слів за параметром близькості до загаданого слова. І тут важливі такі поняття, як ембеддінг слова і косинусна відстань.

Ембеддінги

Сучасні машини розуміють лише мову чисел, і нам складно сказати, що означає конкретне йому слово. Тут на допомогу приходять ембеддінги — вектори дійсних чисел, що визначають слово певному лінійному просторі. Уявити слово або текст у вигляді векторів можна кількома способами. Наприклад, першу версію гри було побудовано на векторах, які генеруються алгоритмом GloVe. Для другої версії ми використовували ембеддінги, отримані за допомогою асоціацій користувачів з сайту https://sociation.org. Відмінності цих методів розглянемо пізніше (рис. 2. 3).

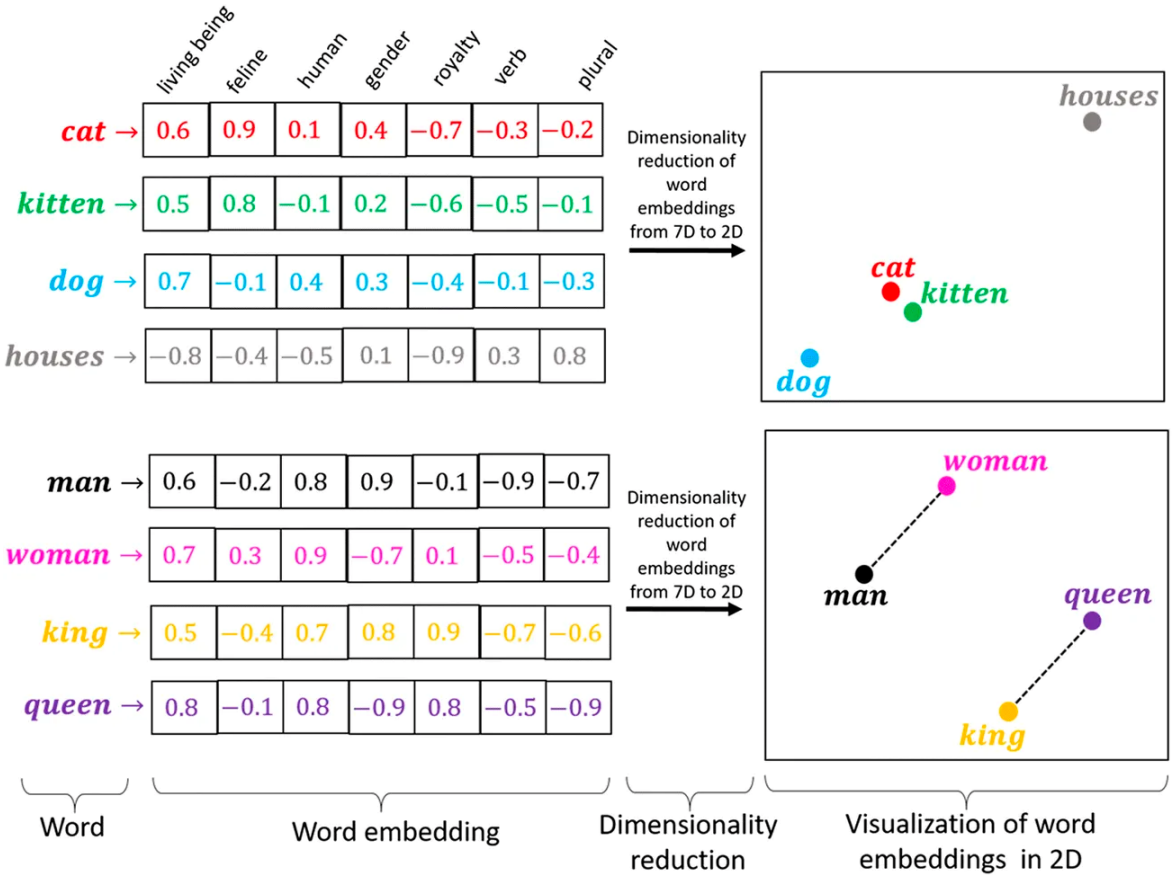

Рисунок 2. 3 - Ембедінг слова

Косинусна відстань

Косинусна відстань — це число, яке показує відстань між векторами, і між словами. Наприклад, вектор слова «вода» буде ближче до слова «озеро», ніж до слова «вогонь», оскільки косинусна відстань у першому випадку менша. Це та ж евклідова відстань, але на відміну від неї косинусна відстань показує, на який кут потрібно повернути перший вектор, щоб він став колінеарним другому (рис. 2. 4).

Рисунок 2. 4 - Косинусна відстань/схожість

Якщо зібрати ембеддінг слів і застосувати до них косинусну відстань, то вийде щось схоже на це:

```
from sklearn.metrics.pairwise import cosine_distances


class DictOrderer:
    def __init__(self, words: list[str], vectorizer) -> None:
        self.words = words
        self.vectorizer = vectorizer

        # передбачаємо вектор для всіх наших слів
        self._word2vector = {word: vectorizer[word] for word in self.words}

    def get_order(self, word: str) -> list[tuple[str, int]]:
        # отримуємо вектор для нашого слова
        word_vector = self._word2vector[word]

        # вектора всіх наших слів у словнику
```

```
        dict_vectors = list(self._word2vector.values())

        # виходить матриця 1xN, де N - це кількість слів у словнику.
        # Кожне значення показує відстань до нашого заданого слова

        distances = cosine_distances(
            [word_vector],
            dict_vectors,
        )[0]

        # отримуємо структуру виду (слово, відстань до заданого слова)


        word_distance = zip(list(self._word2vector.keys()), distances)

        # сортуємо по відстані до нашого слова
        word_distance = sorted(word_distance, key=lambda x: x[1])

        return word_distance
```

Так ми задаємо словник для сортування, а потім передаємо йому об'єкт vectorizer, який у свою чергу має реалізований метод __getitem__. Саме він повертає вектор для слова.

У першій версії гри ми використали алгоритм GloVe (global vectors for word representation) – глобальні вектори для представлення слів. Основна його ідея полягає в тому, щоб враховувати як локальні, так і глобальні відносини слів у різних корпусах текстів. На щастя, для української написали бібліотеку navec, яка навчалася за допомогою GloVe.

```
from navec import Navec


navec = Navec.load("./navec_hudlit_v1_12B_500K_300d_100q.tar")
```

Так виглядає ембедінг слова «поїзд»:

```
print(navec["поїзд"])

array([ 0.31074855, -0.4440416 ,  0.3544376 , …, -0.86068416,  0.12352141,  0.01766999])
```

Вектор до слова отримуємо через метод __getitem__, тому ми можемо передати об'єкт navec у наш DictOrderer:

```
orderer = DictOrderer(navec.vocab.words, navec)
```

Тим самим шляхом виходить готовий алгоритм сортування слів. Можемо випробувати:

```
print(orderer.get_order("поїзд")[:5])

[('поїзд', 5.9604645e-08),
 ('вагон', 0.24743819),
 ('автобус', 0.26409537),
 ('потяги', 0.31421912),
 ('вокзал', 0.34018022)]
```

Начебто все гаразд, але не поспішайте. Якщо ми спробуємо ввести слово «вода», то вийде зовсім не те, що потрібно:

```
print(orderer.get_order("вода")[:5])

[('вода', 0.0),
 ('води', 0.31439257),
 ('воду', 0.32432437),
 ('водою', 0.35056865),
 ('воді', 0.38383615)]
```

Тому потрібно залишити в словнику лише іменники у називному відмінку.

Для другої версії гри ми зв'язалися з розробником Sociation.org - гри в асоціації з колективним розумом. У ній гравцям пропонується вести асоціації до випадково заданих слів. За допомогою зібраних даних можна представляти слова як вектори, розклавши матрицю відношень.

Sociation.org - гра в асоціації з колективним розумом. Sociation.org – це експериментальний некомерційний проект, який має на меті зібрати найбільший словник асоціацій української мови. У нашій скарбничці зібрано вже 813 783 з 57 857 .

- **Гра** – тут ми граємо в асоціації з колективним розумом, тим самим роблячи його розумнішим.
- **Асоціації** – перегляд словника асоціацій у вигляді кольорової анімованої павутини.
- **Слова** – список усіх слів із нашого словника
- **Асоціатори** – список усіх, хто зробив свій внесок у колективний розум.
- **Лабораторія** – експериментальні інструменти для роботи з асоціаціями.

```
orderer = DictOrderer(navec.vocab.words, sociation)
print(orderer.get_order("поїзд")[:5])

[('поїзд', 6.661338147750939e-16),
 ('вагон', 0.09935828259768476),
 ('рельси', 0.13258020472841037),
 ('уз', 0.14348915441665822),
 ('електричка', 0.20177676803251465)]

print(orderer.get_order("вода")[:5])

[('вода', 0.0),
 ('водиця', 0.36609914765055607),
 ('рідина', 0.4206119110523321),
 ('джерело', 0.4404068720939991),
 ('водоймище ', 0.45224501168195497)]
```

Гравці почали відзначати покращення підбору асоціацій після заміни способу векторизації. За це ми дуже вдячні Денису, засновнику sociation.org. Завдяки тому, що ланцюжки асоціацій формувалися людьми, а не статичною машиною з корпусів текстів, виходить дуже цікаво та натуральніше [5].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Принцип гри

Все почалося з колеги, який закинув в локальний чат повідомлення, що він зіграв в гру #59 і вгадав слово з 33 спроб і однієї підказки. Гра виявилася простою і складною одночасно: сайт загадав слово і потрібно було його відгадати. У поле введення відправляєш слово, а штучний інтелект на сайті визначає, наскільки відправлене слово близьке за змістом до загаданого.

Цікава гра, що тренує асоціативне мислення та вміння будувати зв'язки. Нове слово з'являється щодня, що у певному сенсі виглядає обмежувачем. Також гра доступна лише португальською та англійською мовами. З одного боку, це додаткова практика, а з іншого — сумніви «Чи знаю я це слово?» погіршують враження від гри.

Так я задумався про розроблення логічної гри українською мовою. Свою гру «Words» будемо розміщувати на об'єктному сховищі, яке стійкіше прийме користувачів.

Оригінальна гра розташована за адресою context.me. Під час підготовки роботи ми дізналисяся про існування іншої версії guess-word.com. Але ця версія має більш обмежену функціональність. У сайту мінімалістичний інтерфейс зображений на рис. 3. 1. Відомості про гру: номер, кількість спроб та кількість підказок. Поле введення слова. Список відгаданих слів як смуги завантаження. Чим ближче, тим більше її заповнено. Номер праворуч означає відстань у словах, але його можна вимкнути. У меню, що випадає, є налаштування та додаткові ігрові опції:

- Вибрати гру.
- Взяти підказку.
- Здатись.

Якщо відгадати слово, то гра запропонує поділитися результатом і поглянути на найближчі 500 слів. Гра дуже швидко повертає відповідь і вміє визначати початкову форму слова. Іншими словами, cat і cats вважаються одним словом і

виводяться як cat. Усі введені слова трактуються як іменники, і у списку 500 найближчих слів дієслова не зустріти. Це наводить на думку, що список найближчих слів формується окремо, а гра просто звертається до списку. Залишається питання: як скласти список найближчих слів?

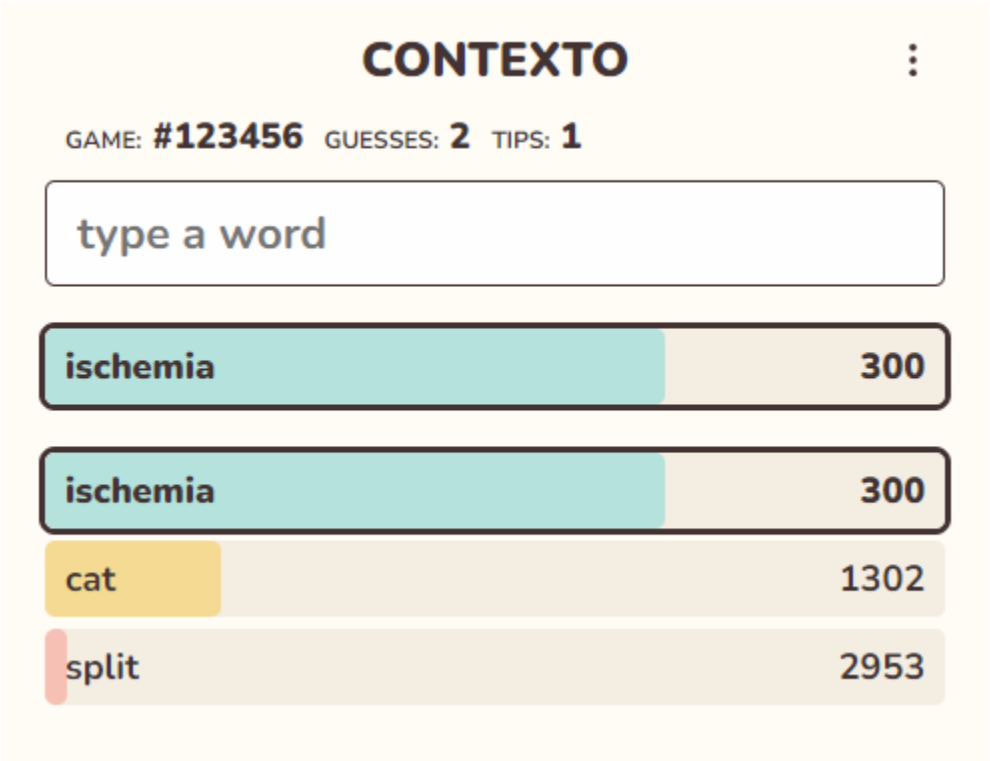

Рисунок 3. 1 – Гра contexto

Гра «Words» - це розумова гра, в якій потрібно вгадати слово, зрозумівши логіку штучного інтелекту. Два місяці тому разом із дипломним керівником ми натрапили на гру https://contexto.me та вирішили локалізувати її. Про те, як ми дійшли до ідеї створення проекту можна ознайомитися в даному розділі. Розглянемо технології та алгоритми, які використовувалися при розробці, і розповімо про допущені помилки на шляху.

Гра працює за принципом «гаряче-холодно»: чим сильніше асоціюється введений гравцем варіант із загаданим словом, тим вищий він у рейтингу. Перша версія була практично повною копією context.me українською мовою, відрізнялася лише скороченим функціоналом та назвою (рис. 3. 2).

Рисунок 3. 2 - Перша розроблена версія гри

Нам не хотілося випускати гру в такому вигляді, тому перед запуском на допомогу з дизайном прийшли колеги — буквально за годину вони створили симпатичний інтерфейс для старту (рис 3. 3).

Рисунок 3. 3 - Друга версія гри «Words»

Якщо коротко, то алгоритм у гри « Words » такий:

- гра задає вам секретне слово, ви повинні його відгадати (як Wordle),
- у користувача є спроби для введення майже будь-якого слова (як Wordle),
- кожна спроба має ранг — відстань до загаданого слова в асоціативному ланцюжку (на відміну від Wordle),
- гравцеві потрібно досягти рангу 1 - найтаємнішого слова.

За нашою оцінкою, реалізувати функціональну частину гри нескладно, якщо мати знання теорії NLP, вміти розробляти бекенд і розфарбовувати кнопки різними кольорами та розуміти фронтенд.

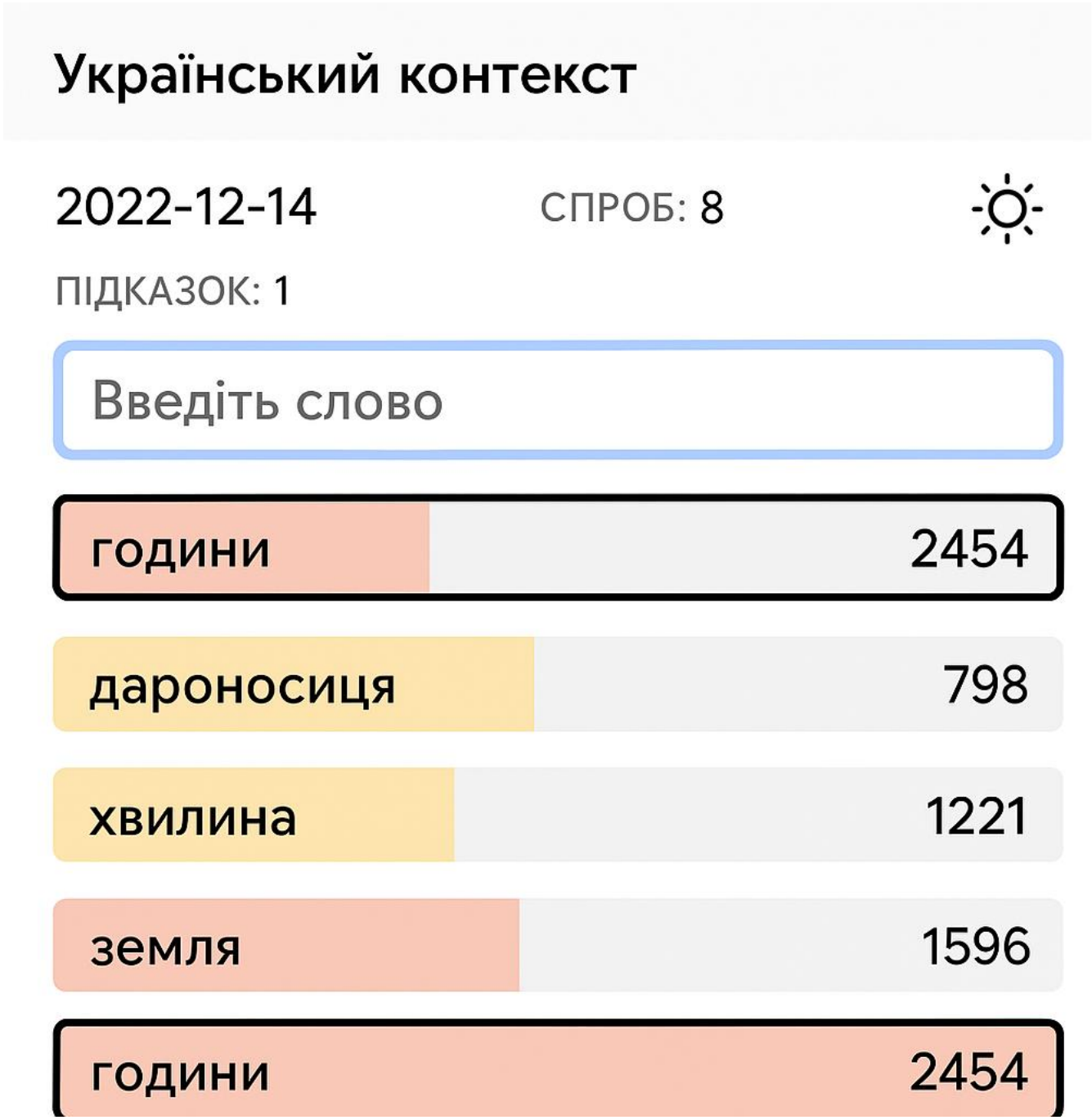

Рисунок 3. 4 – Попередня версія гри

3.2. Текстові ембеддінги

Спочатку комп'ютери не володіють жодною людською мовою. Але людина робить все можливе, щоб це виправити. Людина може сказати одну команду, використовуючи різні слова та у різному порядку. Машині потрібно вміти не просто розрізняти слова, а й розуміти сенс, який ховається за цими словами.

Тут на допомогу приходять текстові ембеддінги. Якщо коротко, то ембеддінг - це перетворення слова на набір чисел, який називають кортежем або вектором. Ці числа задають положення слова як точки у просторі, але не в тривимірному, а багатомірному. Чим ближче дві точки, то ближче слова за змістом, а комп'ютери вміють обчислювати.

Після операції зіставлення з'являється модель — файл, який визначає відповідність «слово — вектор» чи якось визначає правила зіставлення чи обчислення. Для роботи моделі потрібне програмне забезпечення, яке розуміє формат моделі.

Найпростіше і найшвидше спробувати ембеддінги мовою Python. Бібліотека *gensim* реалізує один із найпопулярніших підходів — word2vec. Для роботи необхідна модель, навчена на достатній кількості текстів. У документації gensim є посилання на англомовні моделі, але це нас не влаштовує.

На щастя, проект UAVectores надає моделі українською мовою. На сайті представлені контекстуалізовані та статичні моделі. Завантажуємо архів та розпаковуємо. Модель представлена у двох видах: бінарному (model.bin) та текстовому (model.txt). Спробуємо скористатися цією моделлю. Спершу завантажуємо.

```
from gensim.models import KeyedVectors
model = KeyedVectors.load_word2vec_format("model.txt", binary=False)
```

Тепер можемо знайти слова, найближчі до слова «провайдер»:

```
>>> model.most_similar(positive=["провайдер"])
…
KeyError: "Key 'провайдер' not present in vocabulary"
```

На жаль такого слова не знайшлося. Справа в тому, що дана модель приймає слова разом із міткою, яка визначає частину слова. Це зроблено на розпізнавання слів з однаковим написанням. Наприклад, «піч» можна уявити як «піч_NOUN» та «піч_VERB», тобто як іменник та дієслово відповідно.

```
>>> model.most_similar(positive=["провайдер_NOUN"])
[
  ('ip_PROPN', 0.677890419960022),
  ('internet_PROPN', 0.6627045273780823),
  ('інтернет_PROPN', 0.6595873832702637),
  ('інтернет_NOUN', 0.6567919850349426),
  ('веб_NOUN', 0.6510902047157288),
  ('сервер_NOUN', 0.6460723280906677),
  ('модем_NOUN', 0.6433334946632385),
  ('трафік_NOUN', 0.6332165002822876),
  ('безлімітний_ADJ', 0.6230701208114624),
  ('ритейлер_NOUN', 0.6218529939651489)
]
```

Також візьмемо простіший приклад із кількома словами. Задамо два слова: король та жінка. Людина здогадається, що жінка-король — це, швидше за все, королева.

```
>>> model.most_similar(positive=["король_NOUN", "жінка_NOUN"], topn=1)
[
  ('королева_NOUN', 0.6674807071685791),
  ('королева_ADV', 0.6368524432182312),
  ('принцеса_NOUN', 0.6262999176979065),
  ('герцог_NOUN', 0.613500714302063),
  ('герцогиня_NOUN', 0.5999450087547302)
]
```

Метод most_similar виводить список найбільш схожих слів та деяку метрику відстані до цього слова. Що ближче метрика до одиниці, то ближче слово. Список слів відсортовано за зменшенням цієї метрики. Так як сортування проводиться при виведенні, значення метрики далі ми використовувати не будемо.

Аргумент topn дозволяє встановити кількість слів, які ми хочемо отримати. Таким чином можна запросити якусь велику кількість слів та отримати список, необхідний для створення гри. Давайте поставимо сучасніше слово «кіберпростір» і подивимося на найближче слово і слово, наприклад, на десятитисячної позиції.

```
>>> result = model.most_similar(positive=["кіберпростір_NOUN"], topn=10000)
>>> result[0]
('віртуальний_ADJ', 0.39892229437828064)
>>> result[9998]
('євробут_VERB', 0.12139307707548141)
>>> result[9999]
('татуйований_NOUN', 0.12139236181974411)
```

Наявність специфічних слів, які можуть бути жартом, друкарською помилкою, помилкою в парсингу або локальним жаргонізмом, неприємно впливає на гру. Потрібно очистити словник від дивних слів і залишити тільки іменники.

3.3. Обробка словника

Один із способів зберігання моделі word2vec – текстовий. Формат простий: у першому рядку задаються два числа – кількість рядків у документі та кількість чисел у векторі. Далі на кожному рядку задається слово і далі числа, що позначають вектор.

Тут зручно скористатися особливістю цієї моделі, а саме тегами. Іменники мають тег _NOUN, що дозволяє прибрати з моделі непотрібні слова. Видалити не іменники легко, але як вчинити з друкарськими помилками і дивними словами? Тут на допомогу приходить інший ембеддінг, який навчався на літературі. Це ембеддінг Navec (навік) із проекту. Завантажуємо його та завантажуємо модель:

```
from navec import Navec
path = 'navec_hudlit_v1_12B_500K_300d_100q.tar'
navec = Navec.load(path)
```

Тепер можна перевіряти слова простим синтаксисом:

```
>>> "віртуальний" in navec
True
>>> "євробуд" in navec
False
>>> "татуйований" in navec
False
```

Таким чином, можна відсіяти чималу кількість слів, яким у грі не місце. Приклад видалених слів:

записка

зачаткувати

магазей

антитезти

завойовник

налицотець

прирубу

бислой

цвіт

громадин

міжрайонець

англіканство

скудетто

вибуття

діловик

щобль

Але водночас губляться і справжні слова:

агрокомплекс

кейтеринг

фемтосекунда

вуглепластик

електромашинобудування

мурмолка

реанімобіль

Алгоритм очищення моделі наступний:

1. Якщо слово тег не NOUN, то відкидаємо це слово.
2. Видаляємо зі слова послідовність _NOUN.
3. Перевіряємо чисте слово на наявність в ембеддінгу Navec. Якщо його там немає, слово відкидаємо.
4. Слово, яке пройшло всі перевірки, записуємо у файл.

Після обробки всіх слів в перший рядок нової моделі записуємо два числа: кількість рядків, що залишилися, і розмірність вектора. Розмірність вектора при цій обробці залишається незмінною. Якщо все зроблено правильно, то очищену модель вдасться завантажити:

```
model = KeyedVectors.load_word2vec_format("noun_model.txt", binary=False)
```

Подивимось чи стало після цього краще:

```
>>> result = model.most_similar(positive=["кіберпростір"], topn=10000)
>>> result[0]
('віртуальність', 0.4715898633003235)
>>> result[9998]
('компаунд', 0.15783849358558655)
>>> result[9999]
('хитрість', 0.15783214569091797)
```

Для статистики: вихідна модель містить 248978 токенів, з них 59104 токенів мають мітку іменника. І лише 36269 пройшли «сито» другого ембеддінгу.

3.4. Бекенд гри

Так як C++ є моєю робочою мовою програмування, бекенд я вирішив реалізувати на ній. Спершу розглянемо обробку вхідних даних. Видалити прогалини і перевести текст в нижній регістр — це, очевидно. Але як одержати початкову форму слова?

Тут можна скористатися інструментом MyStem. Для C++ є обгортка pymystem3.

```
import pymystem3
mystem = pymystem3.Mystem()
```

Метод lemmatize приймає на вхід рядок-пропозицію та повертає список слів у початковій формі.

```
>>> mystem.lemmatize("кіт коти котів котах кота")
['кіт', ' ', 'кот', ' ', 'кот', ' ', 'кот', ' ', 'кот', '\n']
```

На перший погляд, навіть продуктивність на гідному рівні: на моїй віртуальній машині лематизація одного слова займає до 10 мс. За мірками сучасного Інтернету це досить швидко.

Поки я працював над бекендом, довелося познайомитися з об'єктним сховищем, серед функцій якого є можливість розміщення статичних сайтів. І тут мені в голову прийшла цікава думка.

3.5. Гра на об'єктному сховищі

Під час розроблення бекенда ми продумували способи захиститися від нечесної гри:

1. Здатись не можна.
2. Список топ-500 найближчих слів отримати можна лише надавши загадане слово.
3. Підказку можна отримати за словом та позицією.

Але невдовзі мені здалося це надто суворими правилами.

На даний момент єдине призначення бекенда - приведення слів до початкової форми. Щоправда, як показало тестування на колегах, і це не обов'язково: усі й так намагалися писати початкові форми слів. Та й модель ембеддінгів не лематизована, тобто гра розуміє слова у початковій формі.

Відмова від бекенда на користь фронтенду - це помилка. Однак від бекенда повністю відмовитися не вийде: генератор близьких слів десь треба запускати. Генератор приймає на вхід задумане слово та формує текстовий файл, де на кожному рядку по одному слову в порядку смислового зменшення від загаданого. Вміст цього файлу також дублюється в JSON-словник, де кожному слову відповідає його відстань від загаданого слова.

JSON файл на кожну гру займає до 2 МБ. При відкритті гри файл скачується до браузера і JavaScript реалізує логіку гри. Цей спосіб не найпродуктивніший, але після завантаження файлу дозволяє грати без підключення до інтернету.

3.6. Серверна частина

Найчастіша операція у грі – отримання рейтингу слова. Але якщо ми щоразу формуватимемо рейтинг, то це займе у користувача 3–5 секунд. Тому найкраще закешувати у пам'яті рейтинги слів, тож ми отримаємо швидкий доступ до визначення рангу слова.

Серверна частина для гри "Words" була написана на Django. У першій версії гри ми використовували стек Django + SQLite + Gunicorn. Так, за первинної ініціалізації проекту визначається сьогоднішнє слово і для нього формується рейтинг. Далі цей рейтинг кешується у пам'яті за допомогою патерну Signleton. Якщо навісити на DictOrder синглтон, то вийде перша версія нашого бекенду.

```
apps.ua
class WordsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'words'

    def ready(self):
        from words.models import DayKeyword, Word, get_today_keyword
        from words.guess import WordGuesser

        day_keyword = get_today_keyword()

        print("initializing app…")
        # ініціалізація займає кілька секунд
WordGuesser(day_keyword.word, [word.word for word in Word.objects.all()])
        print("app is initialized!")

views.py
def make_guess(request):
    # об'єкт ініціалізується миттєво
guesser = WordGuesser()
    guessed_word = json.loads(request.body.decode("utf-8"))["word"]
    guessed_word = guessed_word.strip().lower().replace("ё", "е")

    if not guesser.has_word(guessed_word):
        return JsonResponse({"error_text": f"Я не знаю слово {guessed_word}"}, status=400)

    # отримуємо рейтинг слова
    order = guesser.guess(guessed_word)

    return JsonResponse({
        "order": order
    })
```

Відразу можна побачити кілька мінусів запропонованого підходу:

- одночасно можна зберігати рейтинг лише для одного слова,
- щоразу під час заміни слова потрібно перезавантажувати весь проект.

Але насправді цей підхід дає змогу отримати практично миттєву відповідь на спробу гравців.

У другій версії ми хотіли додати функцію створення власних кімнат з випадковим словом, а за преміум-аккаунт можливість задати своє слово. Для того, щоб підтримувати кеш для кількох слів, ми додали до стеку PostgreSQL та Memcache.

Окремо для кожної кімнати протягом 24 годин ми зберігаємо кеш «{room_id}{word} → order».

На відміну від NLP та Backend-розробки, я не займався клієнтською розробкою професійно. Тому ця частина досі залишається для мене найскладнішою. Для стартового запуску гри була написана простенька html сторінка з одним запитом у бекенд. Для другої версії потрібно було врахувати багато інших факторів та побудувати проект по оптимальній архітектурі.

У планах на майбутнє провести CustDev серед наших гравців, особливо платних. Також шукаємо потенційних партнерів (наприклад, промокоди за вгадування слів, взаємний PR, активну співпрацю) для того, щоб опрацювати майбутній функціонал для реалізації.

З іншого боку, ми накопичили дані про спроби гравців. Це можливість автоматично підбирати асоціативні ланцюжки слів, як у sociation, тим самим будувати нові методи на формування ембеддінга слів. Щоб розвинути цю ідею, ми плануємо провести змагання серед програмістів. Завдання - написати алгоритм, який за оптимальну кількість спроб зможе вгадати задумане слово. Для змагання поділимося всіма зібраними даними.

Під час розроблення гри «Words» ми зрозуміли наші помилки та зробили висновки. Це допоможе нам в інших проектах.

3.7. Тестування гри «Words»

Натхненний грою context.me вирішив розробити власну логічну гру наукраїнській мові (UA). Гра має назву «Words». Правила гри дуже прості, потрібно вгадати загадане слово. Є необмежена кількість спроб. Слова відсортовані штучним інтелектом за змістом у порядку спадання. Після надсилання слова з'явиться його позиція у списку. Загадане слово має 0.

Перед тим, як потрапити у гру, штучний інтелект працював над мільйонами текстів. Він використовує отриманий досвід та розуміння контексту для визначення смислової близькості слів.

Для створення частини користувача необхідний NPM.

```
cd frontend
npm run build
```

У каталозі dist з'явиться prod-складання програми. Підкладаємо її на веб-сервер. У тому ж каталозі створюємо підкаталог game. У цьому підкаталозі повинні бути файли гри. Гра готова до роботи у контейнері об'єктного сховища.

Для того, щоб загадати слово, потрібно запустити модуль processor. Для цього потрібен інтерпретатор Python версії 3.9 або вище.

```
# Встановлюємо залежності
python3 -m pip install -r requirements.txt

# Запускаємо
python3 -m processor -m model -o out/ -n "2025-05-12" "сніг"
```

При першому запуску буде завантажено дві моделі текстових ембеддінгів і на їх основі буде створено іменник. Усі проміжні файли, необхідні для ембеддінгів, будуть збережені в каталог, вказаний через ключ -m. При повторних запусках завантаження не проводиться.

Завантажена модель обробить слово *сніг* і складе два словники гри:

- у форматі *txt*, де кожен рядок містить лише одне слово в порядку "загадане слово на першому рядку, а далі за спаданням";
- у форматі *json*, де один великий асоціативний масив виду слово-позиция.

Ім'я файлу передається у форматі YYYY-MM-DD. Модуль processor не здійснює перевірку імені файлу, однак, фронтенд шукає саме в такому форматі.

Якщо день або місяць складаються з однієї цифри, необхідно доповнити нулями. Наприклад:

- 2024-12-12
- 2025-01-01

Орієнтовна структура файлів на веб-сервері:

```
- /
  - index.html
  - assets/
    - index.XXXXXX.css
    - index.XXXXXX.js
  - game/
    - 2024-12-09.json
    - 2024-12-10.json
    - 2024-12-11.json
    - 2024-12-12.json
    - 2024-12-13.json
```

Відомі обмеження

1. Гра не ставить завдання максимально сховати загадане слово. Чесність гравця завжди справа самого гравця.
2. Гра не перевіряє доступність гри. При запуску шукається гра з ім'ям, що дорівнює поточній даті. Якщо файлу немає, поле введення слова буде закрито.
3. Список ігор починається з 2024-11-13. Це значення жорстко задано у коді.
4. Словник не ідеальний і іноді зустрічаються дивні слова.
5. Лематизація слів не проводиться.

На рисунку 3. 5 зображена власноруч створена іконка логічної мобільної 2-D гри «Words».

Рисунок 3. 5 – Значок мобільної 2-D гри «Words»

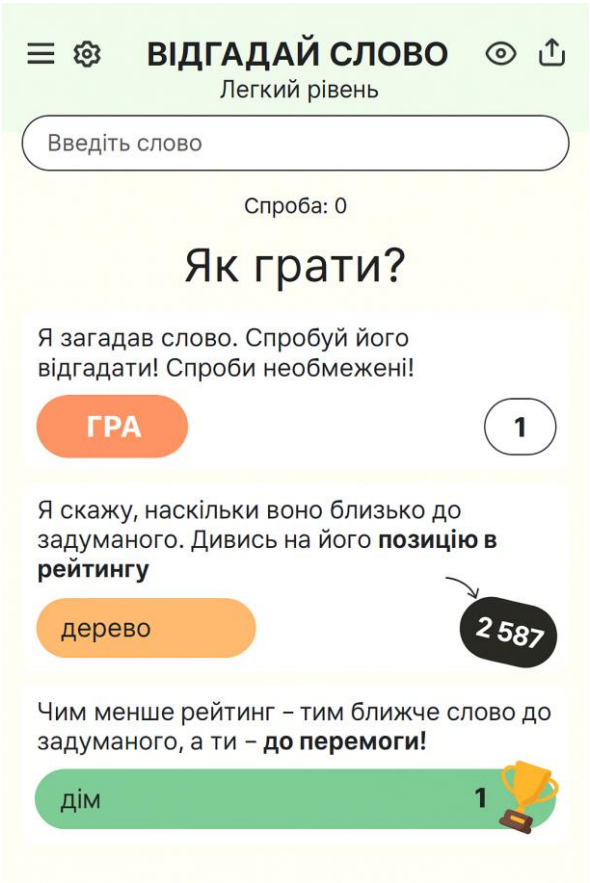

Рисунок 3. 6 – Початкова сторінка гри

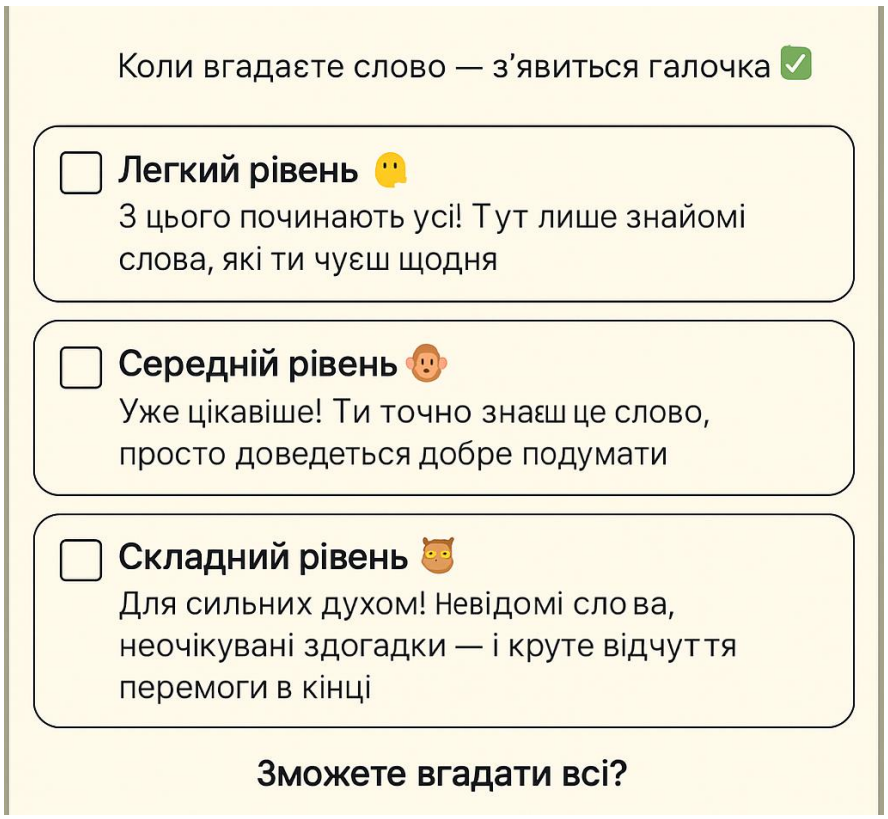

Рисунок 3. 7 – Вибір рівня складності гри

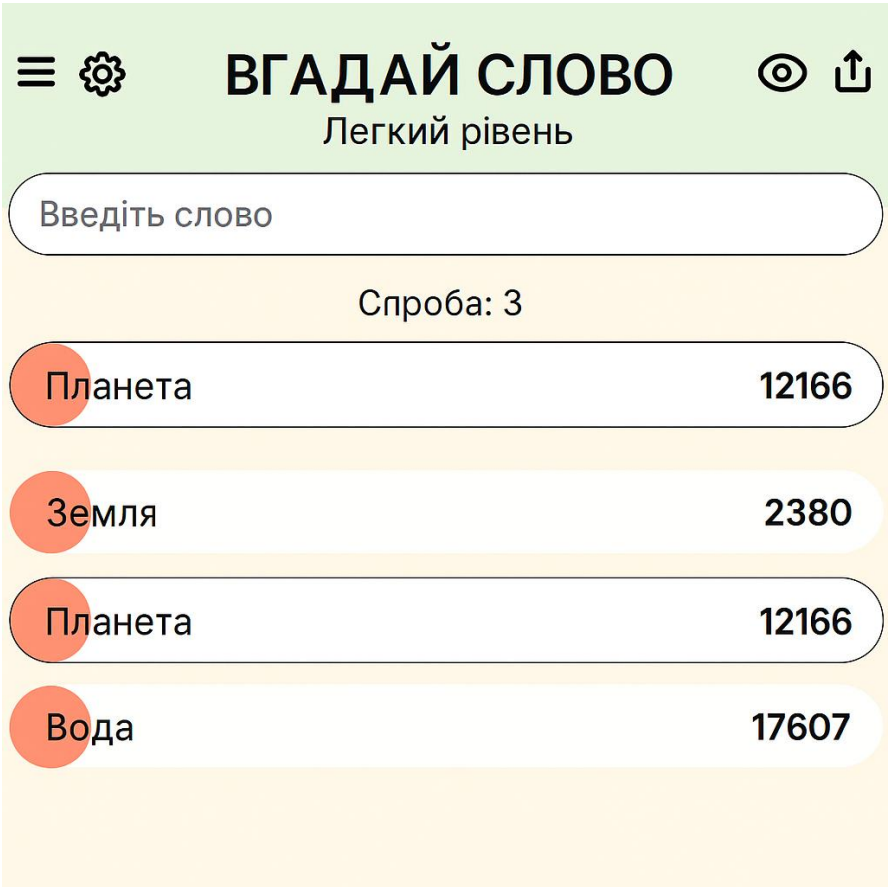

Рисунок 3. 8 – Процес гри «Words»

Рисунок 3. 9 – Наближені слова за рейтингом та кольором до загаданого слова

3.8. Вимоги до програмного та апаратного забезпечення

Нижче подано приклад технічних вимог до програмного та апаратного забезпечення для мобільної гри **"Words"** для платформи Android. Варто зазначити, що ці вимоги є орієнтовними та можуть бути адаптовані залежно від особливостей реалізації гри, її графічного наповнення та інтерактивних функцій.

Вимоги до програмного забезпечення

- **Операційна система:**
  - Android 5.0 (Lollipop) або новіша версія.
- **Середовище розробки та технології:**
  - Використання Android SDK (наприклад, Android Studio).
  - Мови програмування: Java або Kotlin.
  - За потребою – використання ігрового рушія (наприклад, Unity чи Godot) для спрощеної роботи з графікою та анімаціями.
- **Бібліотеки та інтеграції:**
  - Підключення Google Play Services для таблиць лідерів, досягнень та інших соціальних функцій.
  - Використання графічних бібліотек: підтримка OpenGL ES 2.0 (мінімум) або OpenGL ES 3.0 (для покращених візуальних ефектів).
  - Засоби роботи з мережею: бібліотеки для HTTP/REST запитів, WebSocket або інших API-комунікацій, якщо гра має онлайн-компоненти.
- **Інтерфейс користувача:**
  - Розробка адаптивного та responsive UI для коректного відображення на пристроях з різними розмірами екранів.
  - Підтримка локалізації (зокрема українська, англійська та інші мови) для забезпечення комфортної роботи користувачів.
- **Безпека та конфіденційність:**
- Використання стандартних механізмів шифрування для зберігання та передачі даних користувача.

- Реалізація заходів для запобігання несанкціонованому доступу та змінам даних гри.

Вимоги до апаратного забезпечення

- **Процесор:**
  - Мінімум: 1,2–1,3 ГГц однопроцесор або багатоядерний мобільний процесор, що забезпечить стабільну роботу додатка.
- **Оперативна пам’ять (RAM):**
  - Мінімум: 1 ГБ (для базової версії гри).
  - Рекомендовано: 2 ГБ або більше для плавного відтворення анімацій та інтерактивних елементів.
- **Графічний процесор (GPU):**
  - Інтегрований GPU, який підтримує OpenGL ES 2.0 як мінімум.
  - Для більш насичених графічних ефектів – бажана підтримка OpenGL ES 3.0.
- **Дисплей:**
  - Сенсорний екран з мінімальною роздільною здатністю 720p (1280×720) або вище, що дозволяє коректно відображати інтерфейс та графічні елементи гри.
- **Внутрішня пам’ять:**
  - Мінімум: 100 МБ вільного простору для встановлення гри та збереження даних.
  - Рекомендовано перевіряти наявність додаткового простору для кешу та майбутніх оновлень.
- **Додаткові апаратні можливості:**
- Сенсори: акселерометр та гіроскоп (за потребою, якщо у грі передбачено інтерактивні елементи, що реагують на положення пристрою).
- Аудіо: підтримка стандартних аудіовихідних засобів (динаміки та/або аудіовихід для навушників) для відтворення музики та звукових ефектів.

Ці вимоги допоможуть забезпечити стабільну роботу гри на широкому спектрі пристроїв та створити комфортний користувацький досвід. Якщо гра передбачає

розширені мережеві можливості (наприклад, онлайн-таблиці лідерів, мультиплеєр) або інтеграцію зі сторонніми сервісами, варто додатково описати вимоги до роботи з мережею та безпеки зв’язку.

Вимоги до програмного та апаратного забезпечення для мобільної гри "Words":

1. Апаратні вимоги:

Мінімальні:

- Операційна система: Android 8.0 (Oreo) / iOS 12.0
- Процесор: 4-ядерний, 1.4 ГГц
- Оперативна пам'ять: 2 ГБ
- Вільне місце на пристрої: 200 МБ
- Дисплей: 720x1280 px

Рекомендовані:

- Операційна система: Android 11 / iOS 15
- Процесор: 8-ядерний, 2.0 ГГц і вище
- Оперативна пам'ять: 4 ГБ і більше
- Вільне місце на пристрої: 500 МБ
- Дисплей: 1080x2400 px або вище

2. Програмні вимоги:

- Мобільна платформа: Android (підтримка Google Play Services) / iOS (підтримка App Store)
- Мова програмування: Kotlin / Java для Android, Swift / Objective-C для iOS
- Ігровий рушій: Unity або Android SDK + iOS SDK (у разі нативної розробки)
- Підтримка інтернет-з'єднання (для оновлень, рейтингових таблиць, реклами тощо)
- Доступ до бібліотек машинного навчання (за потреби – наприклад, для розпізнавання зображень або голосу)

3. Додаткові вимоги:

- Адаптація під різні розміри екранів і співвідношення сторін
- Підтримка багатомовності (мінімум: українська, англійська, російська)

- Інтеграція з Google Play / Game Center для досягнень та збереження прогресу
- Підтримка push-повідомлень
- Вбудована аналітика (наприклад, Firebase Analytics або аналогічне рішення)
- Оптимізація продуктивності для енергоефективності та зменшення навантаження на батарею

ВИСНОВКИ

У результаті виконання дипломної роботи було розроблено мобільну 2D-гру "Words" з використанням середовища Marmalade SDK та мови програмування C++. У процесі реалізації проєкту були проаналізовані технічні вимоги до мобільних ігор, вивчені особливості побудови 2D-графіки, системи обробки подій та оптимізації продуктивності на мобільних пристроях.

Здійснено повний цикл розробки програмного забезпечення: від проєктування ігрової логіки до реалізації графічного інтерфейсу та тестування готового додатка. У розробці враховано вимоги до кросплатформеності, зручності користування та ефективності використання ресурсів пристрою.

Розроблена гра демонструє стабільну роботу на різних мобільних платформах, має інтуїтивно зрозумілий інтерфейс і привабливий візуальний стиль. Реалізована логіка гри сприяє розвитку словникового запасу та логічного мислення у користувачів. Таким чином, поставлені в роботі цілі були досягнуті, а задачі успішно виконані.

Результати цієї дипломної роботи можуть бути використані як основа для подальшого розвитку проєкту, зокрема, для інтеграції онлайн-функціональності, додавання нових мовних пакетів, підключення реклами або внутрішньоігрових покупок.

Отже, виконання даної дипломної роботи дозволило набути практичних навичок розробки мобільних ігор, застосування сучасних інструментів програмування та створення якісного програмного продукту з урахуванням потреб цільової аудиторії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Шляховий В. І., Мельник П. І. *Програмування мовою C++*. — Львів: Видавництво ЛНУ, 2020. — 352 с.
2. Глинський Я. М., Рудий Ю. М. *Основи об'єктно-орієнтованого програмування*. — Київ: Каравела, 2019. — 288 с.
3. Соловйов В. В. *Проектування комп'ютерних ігор*. — Харків: Фоліо, 2021. — 274 с.
4. Страуструп Б. *The C++ Programming Language*. — 4th Edition. — Addison-Wesley, 2013. — 1376 p.
5. Marmalade SDK Documentation [Електронний ресурс]. — Режим доступу: https://docs.madewithmarmalade.com/
6. Marmalade Quick Start Guide [Електронний ресурс]. — Режим доступу: https://github.com/marmalade/sdk
7. GameDev.net – Articles on 2D Game Development [Електронний ресурс]. — Режим доступу: https://www.gamedev.net/
8. Wikipedia contributors. *Game loop* — Wikipedia, The Free Encyclopedia. [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Game_loop
9. Бублик В. П. *Мобільні додатки: розробка та тестування*. — Київ: КНУ, 2020. — 200 с.
10. Майерс Дж. *Ігровий дизайн. Практичний підхід*. — Київ: Наш Формат, 2022. — 310 с.
11. SDL – Simple DirectMedia Layer [Електронний ресурс]. — Режим доступу: https://www.libsdl.org/
12. ISO/IEC 14882:2020 – *Programming Languages — C++* [Електронний ресурс]. — Режим доступу: https://isocpp.org/
13. Unity vs Marmalade SDK: A Comparative Study [Електронний ресурс]. — Режим доступу: https://www.gamasutra.com/
14. Блінов Д. С. *Розробка мобільних ігор: принципи, архітектура, реалізація*. — Одеса: Технополіс, 2021. — 245 с.

ДОДАТКИ

ДОДАТОК А

```
<template>
  <div>
    <div class="row stats-panel">
      <div class="d-flex align-bottom col-6">
<!--        <span class="label">Гра:</span>-->
        <span>{{this.game_id}}</span>  
      </div>
      <div class="d-flex col-5">
        <span class="label">Спроб:</span>
        <span>{{this.attempt}}</span>  
      </div>
      <div class="d-flex col-1 justify-content-end">
        <img class="img-hint" src="/tip.png" :disabled="this.loading" v-
on:click="tip">
      </div>
    </div>

    <div class="row stats-panel">
      <div class="d-flex col-3" v-if="hint">
        <span class="label">Підказок:</span>
        <span>{{this.hint}}</span>
      </div>
    </div>

    <input class="form-control form-control-lg" type="text" placeholder="Введіть
слово" aria-label=".form-control-lg example" v-on:keyup.enter="guess" v-model="text"
:disabled="this.loading">

    <div class="message-panel">
      <div class="alert alert-primary" role="alert" v-if="solved">
        <h1>Вітаємо!</h1>
        Ви вгадали слово за {{this.attempt}} спроб і {{this.hint}} підказок.

        <router-link :to="closeWordsLink"><button type="button" class="btn mt-3 btn-
light w-100">Найближчі слова</button></router-link>
      </div>
      <div class="alert alert-primary" role="alert" v-if="lastWordNotFound">
        Здається, слова {{lastPayload.word}} немає в словнику або воно занадто
далеко!
      </div>
      <div class="alert alert-danger" v-if="dup">
        Слово {{this.lastPayload.lemma}} вже відгадано!
      </div>
<!--      <div-->
<!--          class="progress progress-bar progress-bar-striped progress-bar-animated
loading"-->
<!--          role="progressbar"-->
<!--          v-if="this.loading"-->
<!--      ></div>-->
      <ProgressBar :payload="this.lastPayload" v-if="lastWordCorrect" :strip="true"/>

      <div class="alert alert-dark" role="alert" v-if="this.words.length === 0 &&
!this.id">
        <span class="title">Как играть?</span><br><br>
        <p> Вгадайте загадане слово. У вас безліч спроб.</p>
        <p> Слова відсортовані штучним інтелектом за змістом у порядку спадання.</p>
```

```
        <p> Після надсилання слова з'явиться його позиція у списку. Загадане слово
має номер 0.</p>
        <p> Перед тим, як потрапити сюди, штучний інтелект працював над мільйонами
текстів. Він використовує отриманий досвід та розуміння контексту для визначення
смислової близькості слів.</p>
        <p> Зроблено під враженням від <a href="https://contexto.me/">contexto.me</a>
</p>
        <p> Додавання нових ігор поки що призупинено.</p>
      </div>
    </div>

    <div class="col mt-4">
      <ProgressBar v-for="payload in this.sortedWords" :payload="payload" :dup="dup"
:last="lastPayload"/>
    </div>
  </div>
</template>

<script>
import axios from 'axios';
import ProgressBar from "../components/ProgressBar.vue";
export default {
  name: "PlaygroundView",
  components: {ProgressBar},
  props: ["id"],
  data() {
    return {
      game_id: "0",
      text: "",
      attempt: 0,
      words: [],
      lastPayload: {},
      dup: false,
      solved: false,
      hint: 0,
      loading: false,
      dict: {}
    }
  },
  computed: {
    sortedWords: function () {
      return this.words.sort(function (a, b) {
          if(a.distance < b.distance) return -1;
          if(a.distance > b.distance) return 1;
          return 0
      })
    },
    closeWordsLink: function () {
      return '/' + this.game_id + '/words'
    },
    lastWordCorrect: function () {
      if ("error" in this.lastPayload) return false;
      if (!("distance" in this.lastPayload)) return false;
      return this.lastPayload.distance !== -1 && !this.dup;
    },
    lastWordNotFound: function () {
      if("error" in this.lastPayload) return false;
      if(!("distance" in this.lastPayload)) return false;
      return this.lastPayload.distance === -1;
    },
    lastWordGuessed: function () {
      if("error" in this.lastPayload) return false;
      if(!("distance" in this.lastPayload)) return false;
      return this.lastPayload.distance === 0;
```

```
    }
  },
  methods: {
    leading_zero: function (data) {
      if(String(data).length === 1) return "0" + data
      return data
    },
    guess: function () {
      let clean_word = this.text.trim().toLowerCase().replace("ë", "e")

      let result = this.dict[clean_word]
      if(result === undefined) result = -1

      this.lastPayload = {
        "word": this.text,
        "lemma": clean_word,
        "distance": result
      }

      this.dup = false;
      for(let i in this.words) {
        let data = this.words[i]
        if(data.lemma === clean_word) {
          this.dup = true;
        }
      }

      if(!this.dup) {
        if(this.lastPayload.distance !== -1) {
          this.words.push(this.lastPayload)
          if(!this.solved) this.attempt++;
        }

      }

      if (this.lastPayload.distance === 0) {
        this.solved = true;
      }
      this.saveState()
      this.text = ""
    },
    tip: function () {
      this.dup = false;
      let self = this;

      let min_distance = 50_000;
      for(let i in this.words) {
        if(this.words[i].distance < min_distance) min_distance =
this.words[i].distance;
      }
      min_distance = ~~(min_distance / 2);
      if(min_distance < 1) min_distance = 1;

      let found = true;
      while(found) {
        found = false;
        for(let i in this.words) {
          if(this.words[i].distance === min_distance) {
            found = true;
            min_distance += 1;
            break;
          }
        }
      }
```

```
      if (!self.solved) this.hint++;

      for(let i in self.dict) {
        if(self.dict[i] === min_distance) {
          this.lastPayload = {
            "word": i,
            "lemma": i,
            "distance": min_distance
          }
          break
        }
      }

      this.words.push(this.lastPayload)
      self.saveState()
    },
    loadState: function() {
      console.log("loading " + this.game_id)

      let self = this;
      axios.get("/game/" + this.game_id + ".json").then(function (response) {
        self.dict = response.data
        self.loading = false
      })
      if(!localStorage.games) return;

      let games_data = JSON.parse(localStorage.games)
      if(!(self.game_id in games_data)) return;

      self.attempt = games_data[self.game_id].attempt
      self.words = games_data[self.game_id].words
      self.solved = games_data[self.game_id].solved
      self.hint = games_data[self.game_id].hint
    },
    saveState: function () {
      if(!localStorage.games) {
        localStorage.games = "{}"
      }

      let games_data = JSON.parse(localStorage.games)
      games_data[this.game_id] = {
        'attempt': this.attempt,
        'words': this.words,
        'solved': this.solved,
        'hint': this.hint
      }
      localStorage.setItem("games", JSON.stringify(games_data))
    }
  },
  mounted() {
    this.loading = true;
    if (!this.id) {
      let now = new Date(2023, 1, 16);
      let year = now.getFullYear()
      let month = this.leading_zero(now.getMonth() + 1)
      let day = this.leading_zero(now.getDate())
      this.game_id = "" + year + "-" + month + "-" + day;
    } else {
      this.game_id = this.id
    }
    this.loadState()
  }
}
```

```
</script>

<style scoped>
.label {
  font-size: 12px;
  text-transform: uppercase;
  margin-right: 6px;
  font-weight: 700;
  margin-top: 6px;
}

span {
  font-size: 18px;
  font-weight: 900;
}

.stats-panel {
  padding: 10px 10px 0;
}

input {
  margin-top: 10px;
  margin-bottom: 10px;
}

.loading {
  width: 100%;
  background-color: #cccccc;
  border: 5px solid #000000;
  height: 47px;
}

p {
  font-weight: 550;
}

.help-button {
  width: 36px;
  height: 36px;
  border-radius: 19px;
}

.img-hint {
  width: 27px;
  height: 27px;
}
</style>
```

ДОДАТОК Б

```
import json
import logging
import os

from navec import Navec

logging.basicConfig(level=logging.INFO)

import argparse
import requests
import navec

from zipfile import ZipFile
from tqdm import tqdm
from pathlib import Path
from gensim.models import KeyedVectors


parser = argparse.ArgumentParser(
    prog="Processor",
    description="Create list of similar words"
)

parser.add_argument("-o", "--out", help="Path for output dir", default=".")
parser.add_argument("-n", "--name", help="Name for output file", default="word")
parser.add_argument("-m", "--model", help="Path for directory with model",
default=".")
parser.add_argument("word", help="")

args = parser.parse_args()

model_dir = Path(args.model)
out_dir = Path(args.out)
word = str(args.word)
out_name = str(args.name)

# Create directories if necessary
model_dir.mkdir(parents=True, exist_ok=True)
out_dir.mkdir(parents=True, exist_ok=True)
assert out_dir.is_dir() is True

noun_only_model_txt = model_dir / "noun_model.txt"

if noun_only_model_txt.exists() is False:
    logging.info("Downloading model...")
    # Download model
    # http://vectors.nlpl.eu/repository/20/180.zip - 116
    # http://vectors.nlpl.eu/repository/20/182.zip -
    zip_path = model_dir / "182.zip"
    response = requests.get("http://vectors.nlpl.eu/repository/20/182.zip",
stream=True)
    with zip_path.open("wb") as f:
        for data in tqdm(response.iter_content(chunk_size=4*1024*1024), total=153):
            f.write(data)

    logging.info("Unzipping...")
    with ZipFile(zip_path) as archive:
        archive.extractall(model_dir)

    navec_model_tar = model_dir / "navec_hudlit_v1_12B_500K_300d_100q.tar"
```

```
    response = requests.get("https://storage.yandexcloud.net/natasha-
navec/packs/navec_hudlit_v1_12B_500K_300d_100q.tar", stream=True)
    with navec_model_tar.open("wb") as f:
        for data in tqdm(response.iter_content(chunk_size=4*1024*1024), total=13):
            f.write(data)

    navec = Navec.load(navec_model_tar)

    original_model_txt = model_dir / "model.txt"

    logging.info("Extracting nouns...")
    nouns = []
    # mystem = pymystem3.Mystem()
    with original_model_txt.open("r", encoding="utf-8") as f:
        for line in tqdm(f, total=185925):
            if "_NOUN" not in line or "::" in line:
                continue

            clear_line = line.replace("_NOUN", "")
            clear_word = line.split('_')[0].strip()

            if "-" in clear_word:
                continue

            if clear_word not in navec:
                print(f"Strange word: {clear_word}")
                continue

            nouns.append(clear_line)

    with noun_only_model_txt.open("w", encoding="utf-8") as f:
        f.write(f"{len(nouns)} 300\n")
        for line in nouns:
            f.write(line)
else:
    logging.info("Model found!")

logging.info("Loading model...")
model = KeyedVectors.load_word2vec_format(noun_only_model_txt, binary=False)


def create_dictionary(word: str, out_dir: Path, filename: str):
    out_txt_file: Path = out_dir / f"{filename}.txt"
    out_json_file: Path = out_dir / f"{filename}.json"
    words = {
        word: 0
    }

    with out_txt_file.open("w", encoding="utf-8") as f:
        f.write(f"{word}\n")

        for i, (similar_word, _) in enumerate(model.most_similar(positive=[word],
topn=50000), 1):
            f.write(f"{similar_word}\n")
            words[similar_word] = i

    with out_json_file.open("w", encoding="utf-8") as f:
        json.dump(words, f, ensure_ascii=False)


create_dictionary(word, out_dir, out_name)
```