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))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: "Використання засобів штучного інтелекту
для розроблення гри Crosswordium"

Виконав: студент 6 курсу групи КН-62м
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Янчишин Роман Ростиславович
(прізвище та ініціали)

Керівник Шиманський В. М.
(прізвище та ініціали)

Рецензент Гентош Л. І.
(прізвище та ініціали)

м. Львів – 2025 рік

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра комп'ютерних наук
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

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

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛПФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Янчишину Роману Ростиславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Використання засобів штучного інтелекту для розроблення гри Crosswordium

керівник роботи Шиманський Володимир Михайлович, кандидат технічних наук, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "29" *квітня* *2025 року* № С-288

2. Термін подання студентом роботи 10.12.2025 р.

3. Вихідні дані до роботи вимоги до функціонування ігрових систем типу кросвордів; словникові ресурси для формування лексичного наповнення; документація та специфікації API сервісів штучного інтелекту для генерації текстового контенту; середовище розробки Unity та мова програмування C#.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області й існуючих методів генерації кросвордів; опис архітектури гри Crosswordium та застосованих технологій; розроблення та інтеграція модуля ШІ для підбору слів і створення підказок, експериментальна оцінка результатів та висновки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) структурні та модульні схеми системи *Crosswordium*, UML-діаграми роботи редактора та модуля ШІ, блок-схема алгоритму створення кросворду, схема взаємодії з OpenAI API.

6. Дата видачі завдання: 01.05.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми, формулювання мети та завдань дослідження.	01.05.2025	Виконано
2	Аналіз предметної області та огляд існуючих рішень для створення кросвордів.	05.09.2025	Виконано
3	Розроблення концепції системи та алгоритмів генерації кросворду з використанням ШІ.	26.09.2025	Виконано
4	Реалізація програмної частини у середовищі Unity з інтеграцією OpenAI API.	17.10.2025	Виконано
5	Тестування роботи модулів і перевірка якості генерації контенту.	31.10.2025	Виконано
6	Аналіз результатів і розроблення ідей стартап-проєкту Crosswordium.	14.11.2025	Виконано
7	Оформлення дипломної роботи та підготовка до захисту.	10.12.2025	Виконано

Студент _____ Янчишин Р. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шиманський В. М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У роботі представлено систему автоматизованого створення кросвордів із використанням технологій штучного інтелекту на прикладі гри Crosswordium. Розроблений у середовищі Unity редактор рівнів поєднує класичні алгоритми побудови сітки з можливостями мовних моделей GPT для підбору слів, генерації підказок і оцінювання якості. Інтеграція з OpenAI API забезпечує гнучке оновлення контенту, а локальні словники — лінгвістичну точність. Описано архітектуру, результати тестування та перспективи розвитку стартап-проєкту Crosswordium із підтримкою користувацьких рівнів і AI-генерації контенту.

Ключові слова: штучний інтелект, кросворд, Unity, GPT, OpenAI API, Crosswordium..

ABSTRACT

The thesis presents an automated crossword creation system using Artificial Intelligence based on the Crosswordium game. Developed in Unity, the AI-powered level editor combines classical grid algorithms with GPT language models for word selection, clue generation, and quality assessment. Integration with the OpenAI API enables flexible content updates, while local dictionaries ensure linguistic accuracy. The work outlines the system architecture, testing results, and development prospects of the Crosswordium startup with user-created levels and AI-based content generation.

Keywords: artificial intelligence, crossword, Unity, GPT, OpenAI API, Crosswordium.

ТЕХНІЧНЕ ЗАВДАННЯ

Метою роботи є розроблення програмного та алгоритмічного забезпечення системи створення кросвордів із використанням технологій штучного інтелекту на прикладі гри Crosswordium.

Необхідно:

1. Проаналізувати існуючі методи створення кросвордів та доцільність застосування ШІ.
2. Реалізувати методи пошуку слів за шаблоном із локального словника.
3. Розробити модуль інтеграції OpenAI API для підбору слів і генерації підказок за заданим шаблоном.
4. Розробити систему оцінювання адекватності й актуальності підібраних слів за допомогою ШІ.
5. Протестувати й оцінити ефективність розробленого модуля та сформулювати рекомендації щодо його використання.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

AI (ШІ) — Artificial Intelligence — штучний інтелект, галузь комп'ютерних наук, що досліджує методи створення систем, здатних до інтелектуальної поведінки.

API — Application Programming Interface — інтерфейс програмування застосунків, що дозволяє взаємодію між програмними компонентами.

GPT — Generative Pre-trained Transformer — генеративний попередньо навчений трансформер, архітектура сучасних мовних моделей.

UI — User Interface — користувацький інтерфейс, система елементів взаємодії користувача із застосунком.

UX — User Experience — досвід користувача, сукупність вражень від взаємодії з продуктом.

IDE — Integrated Development Environment — інтегроване середовище розробки програмного забезпечення.

JSON — JavaScript Object Notation — формат обміну структурованими даними.

Unity — мультиплатформенний рушій для розробки ігор та інтерактивних застосунків.

Firebase — хмарна платформа Google для зберігання даних, обробки запитів і керування оновленнями у реальному часі.

API Key — унікальний ідентифікатор доступу до зовнішнього API-сервісу (зокрема OpenAI).

Crosswordium — назва мобільної гри, розробленої у межах дипломного проєкту.

AI Editor — внутрішній редактор рівнів Crosswordium із підтримкою генерації контенту за допомогою ШІ.

Кросворд — головоломка, у якій слова вписуються в сітку відповідно до заданих підказок.

ВСТУП

Сучасний розвиток інформаційних технологій та стрімке поширення штучного інтелекту (ШІ) істотно впливають на процеси створення інтерактивного контенту, зокрема — освітніх та розважальних застосунків. Ігрова індустрія дедалі частіше використовує інтелектуальні системи не лише для побудови віртуальних світів, а й для автоматизації створення ігрового наповнення. Одним із перспективних напрямів є використання мовних моделей для генерації текстового контенту — завдань, підказок, описів чи сюжетних елементів.

Актуальність теми зумовлена потребою у скороченні часу та ресурсів, необхідних для створення якісного ігрового контенту, а також підвищенням рівня персоналізації і динамічності у навчальних іграх. На сьогодні більшість класичних кросвордних ігор створюються вручну, що обмежує масштабування проєктів та знижує варіативність ігрового процесу. Інтеграція засобів штучного інтелекту, зокрема моделей природної мови OpenAI, які активно застосовуються для генерації текстового контенту у прикладних системах [1], відкриває можливість часткової або повної автоматизації процесу формування слів, підказок та оцінювання складності рівнів.

Об’єктом дослідження є процес розроблення ігрових систем із використанням засобів штучного інтелекту.

Предметом дослідження є методи інтеграції мовних моделей у процес створення контенту гри типу «кросворд», а також алгоритми оцінки якості та складності згенерованих даних.

Метою роботи є розроблення інтелектуальної системи підтримки створення кросвордів — гри Crosswordium, що поєднує класичні алгоритми побудови кросвордів із сучасними ШІ-технологіями генерації та аналізу контенту.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Проаналізувати існуючі підходи до створення кросвордів та методи застосування штучного інтелекту у генерації текстових даних.
2. Розробити архітектуру ігрової системи Crosswordium та редактора рівнів.

3. Інтегрувати OpenAI API для автоматизованого формування слів і підказок за заданими шаблонами.
4. Провести експериментальне дослідження ефективності використання ШІ при генерації контенту, порівнявши з традиційними методами.
5. Реалізувати механізм оцінювання складності та якості згенерованих рівнів, який враховує аналітику користувачів і забезпечує перевірку семантичної, мовної та етичної відповідності контенту.

Наукова новизна.

Робота зосереджена на дослідженні можливостей використання сучасних мовних моделей штучного інтелекту для автоматизованого створення контенту кросвордних ігор. Новизна полягає у практичному поєднанні класичних алгоритмів побудови кросвордів із генеративними моделями, що дозволяють не лише підбирати слова та підказки, а й оцінювати їх складність, змістову коректність та відповідність етичним нормам.

Реалізовано інтеграційний модуль для Unity, який забезпечує зручну взаємодію з OpenAI API, спрощує процес запитів і обробки відповідей, та може бути використаний для швидкого розширення функціоналу гри без змін основної архітектури проєкту.

Практичне значення.

Результати роботи демонструють, у яких випадках застосування штучного інтелекту реально підвищує ефективність розробки контенту — зменшує витрати часу, полегшує творчу частину процесу та покращує якість отриманих результатів.

Отримані напрацювання підтверджують, що сучасні ШІ-сервіси можуть стати практичним інструментом для розробників ігор, які прагнуть швидко створювати, оновлювати чи масштабувати текстовий контент без суттєвих фінансових витрат. Це узгоджується з сучасними дослідженнями у сфері застосування генеративного штучного інтелекту в ігрових та освітніх системах [3, 7].

Запропоноване рішення може бути використане в комерційних і навчальних проєктах як гнучкий інструмент інтеграції штучного інтелекту у процеси контентної генерації.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Вступ до проблемної області

Словесні ігри залишаються одним із найпоширеніших видів інтелектуальних розваг, що поєднують логіку, мовознавство й елементи навчання. Кросворди, анаграми та подібні головоломки мають не лише розважальне, а й освітнє значення — розвивають пам’ять, увагу, мовну креативність і логічне мислення. Завдяки цим перевагам вони успішно адаптувалися до цифрових платформ, зокрема мобільних, де утримують стабільну популярність серед широкої аудиторії.

Сучасний користувач очікує від таких ігор постійного оновлення контенту, різноманітності тем і стилів. Проте ручне створення кросвордів вимагає значних часових і трудових ресурсів: добір слів, написання підказок і перевірка коректності виконуються переважно вручну. Це обмежує швидкість розробки та ускладнює масштабування проєктів, особливо комерційних чи навчальних.

Тому все більшої актуальності набувають інтелектуальні методи автоматизації створення контенту, що дозволяють швидко формувати нові рівні та адаптувати завдання під різні категорії користувачів. Одним із перспективних напрямів є застосування мовних моделей штучного інтелекту, здатних підбирати слова за шаблонами, створювати осмислені підказки, оцінювати складність і навіть контролювати якість тексту.

Попри активне поширення генеративних сервісів (OpenAI, Gemini, Claude, Mistral тощо), їх практичне використання у сфері ігрового контенту, зокрема україномовного, поки досліджене недостатньо. Це формує потребу у вивченні ефективності інтеграції таких інструментів у реальні проєкти, де важливо не лише отримати згенеровані дані, а й оцінити їх якість, смислову коректність і відповідність ігровим правилам.

Отже, проблемна область роботи полягає у пошуку способів поєднання класичних алгоритмів побудови кросвордів із сучасними засобами штучного інтелекту, що здатні підвищити ефективність і якість створення контенту в іграх типу Crosswordium.

1.2 Класичні методи підбору слів і генерації підказок

До традиційних способів створення кросвордів та словесних головоломок належить ручний підбір слів і написання підказок, що вимагає значного залучення людини-редактора чи укладача. На етапі підготовки завдань укладач використовує власні знання, тематичні словники, енциклопедії або електронні бази даних. Кожне слово та відповідна підказка підбираються індивідуально, що забезпечує якість, унікальність та точність змісту, але водночас істотно обмежує швидкість створення нового контенту.

З розвитком комп'ютерних технологій поширилися автоматизовані засоби пошуку слів за заданими параметрами — наприклад, за кількістю букв, фіксованими літерами на певних позиціях тощо. Для цього часто використовуються локальні текстові словники або бази даних слів. Найпростішим способом є фільтрація словника за допомогою регулярних виразів чи шаблонів, що є типовим підходом у класичних системах генерації кросвордів і словесних ігор [2, 8]. Це дозволяє швидко знаходити усі варіанти, які задовольняють умову.

Для генерації підказок класичними методами застосовується пошук у спеціалізованих лексичних ресурсах, наприклад, енциклопедіях, довідниках або відкритих базах підказок, часто — з подальшою ручною редакцією результатів для підвищення якості.

Основні переваги класичних підходів полягають у їх простоті, зрозумілості та передбачуваності результатів, що дозволяє уникнути двозначностей і помилок. Разом з тим, такі методи мають суттєві обмеження:

- потребують багато часу на підготовку;
- мало пристосовані до оперативного створення великої кількості нових завдань;
- часто не забезпечують достатньої креативності та різноманіття контенту.

Це стимулює пошук нових, більш гнучких рішень, зокрема із залученням сучасних технологій штучного інтелекту.

1.3 Використання ШІ у лінгвістичних та ігрових системах

Штучний інтелект (ШІ) поступово перетворюється на один із ключових інструментів у сфері обробки природної мови. Завдяки розвитку генеративних мовних моделей нового покоління, що базуються на трансформерних архітектурах та широко застосовуються для розв’язання задач обробки природної мови [3, 9–10], з’явилася можливість автоматично створювати, аналізувати й адаптувати текстовий контент для найрізноманітніших прикладних завдань. До таких моделей належать GPT-4, GPT-5, Gemini, Claude, Mistral, LLaMA.

Особливістю сучасних моделей є мультимовність, що дозволяє ефективно працювати з україномовними даними й створювати локалізований контент без значної втрати якості.

У лінгвістичних системах штучний інтелект застосовується для виконання таких завдань:

- автоматичний переклад текстів між десятками мов із урахуванням контексту;
- генерація визначень, пояснень і коротких описів понять;
- добір синонімів, антонімів і тематично споріднених слів;
- побудова словникових або термінологічних баз;
- пошук контекстного значення слова та аналіз семантичних зв’язків.

Такі можливості вже використовуються у навчальних застосунках і системах дистанційної освіти для створення адаптивних навчальних матеріалів, тестів і тренажерів, що автоматично підлаштовуються під рівень користувача.

У ігровій сфері ШІ активно використовується для:

- генерації діалогів, сюжетних гілок і описів предметів;
- створення текстових підказок, назв, квестів і завдань;
- адаптації контенту до стилю або мови гравця;
- автоматичного формування завдань у словесних іграх та головоломках.

Завдяки інтеграції з рушіями Unity, Unreal Engine або іншими платформами розробники можуть поєднувати алгоритмічні механізми з мовними моделями для динамічного створення текстового наповнення.

У кросвордах і подібних іграх це відкриває можливість генерації слів за шаблонами, створення оригінальних підказок різних стилів (навчальних, жартівливих, асоціативних) та оцінювання змістовної якості контенту.

Сьогодні ця галузь активно розвивається: великі компанії пропонують відкриті API-сервіси, а рушії підтримують прямі інтеграції з мовними моделями. Такі рішення стають основою для побудови інтелектуальних інструментів контент-генерації, де система може створювати матеріали, адаптовані до тематики гри, рівня гравця та мовних особливостей.

Таким чином, використання штучного інтелекту у лінгвістичних та ігрових системах відкриває новий етап розвитку інтерактивних застосунків. Це не лише скорочує час створення контенту, а й підвищує його різноманітність і креативність, що робить ШІ ефективним засобом удосконалення процесу розробки ігор типу Crosswordium.

1.4 Основні проблеми та виклики

У процесі створення сучасних словесних ігор, зокрема кросвордів, розробники стикаються з рядом проблем, що ускладнюють автоматизацію підбору слів та генерації підказок. Ці виклики стосуються як класичних методів, так і використання засобів штучного інтелекту.

Однією з ключових проблем є обмеженість та статичність локальних словників. Навіть великі бази слів з часом втрачають актуальність, а їх поповнення вимагає ручної праці та контролю. У результаті контент може ставати одноманітним, а завдання — передбачуваними для гравців.

Використання ШІ, хоча й відкриває нові можливості для генерації слів і підказок, супроводжується низкою додаткових викликів:

- Недостатня точність підбору слів за шаблоном. Мовні моделі не завжди коректно дотримуються заданих умов, можуть пропонувати слова, що не відповідають критеріям (наприклад, неправильна довжина або невідповідність позицій літер).

- Якість і релевантність підказок. Автоматично згенеровані підказки іноді є нечіткими, занадто загальними або не пов’язаними зі словом.
- Мовні обмеження. Сучасні моделі ШІ працюють краще для популярних мов (наприклад, англійської) і можуть допускати більше помилок у менш поширених мовах, зокрема українській.
- Потреба у додатковій перевірці та модерації. Через ймовірність генерації некоректних чи небажаних результатів виникає необхідність ручної перевірки відповідей.
- Залежність від зовнішніх сервісів. Для роботи з ШІ, зокрема через OpenAI API, потрібен стабільний доступ до інтернету, а також дотримання лімітів і політик сервісу.

Окрім цього, залишається актуальною проблема пошуку оптимального балансу між автоматизацією й контролем з боку людини. Повна автоматизація може зменшити якість контенту, тоді як надмірна участь редактора — знижує швидкість і масштабованість процесу.

Таким чином, подальший розвиток редакторів кросвордів і подібних систем потребує впровадження комбінованих підходів, які поєднують переваги класичних методів і сучасних технологій штучного інтелекту, з одночасним урахуванням вказаних викликів.

Висновки до розділу

У розділі розглянуто сучасний стан розроблення словесних ігор та способи автоматизації підбору слів і створення підказок. Класичні методи забезпечують точність і логічність, але потребують багато часу й не підходять для швидкого оновлення великої кількості завдань.

Завдяки розвитку технологій штучного інтелекту з’явилися нові можливості для автоматичного створення і перевірки контенту. Сучасні мовні моделі можуть не

лише генерувати слова й підказки, а й оцінювати їх складність і якість, що значно полегшує роботу розробників.

Сьогодні спостерігається перехід від повністю ручного процесу до комбінованих рішень, де класичні алгоритми поєднуються з гнучкістю ШІ. Такий підхід робить створення кросвордів швидшим, різноманітнішим і доступнішим для постійного оновлення.

Отримані результати стали основою для подальшої частини роботи, де розглядається практична реалізація цих принципів у грі Crosswordium.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Концепція та архітектура проєкту Crosswordium

Проєкт Crosswordium — це комплексна система, що об’єднує декілька компонентів, спрямованих на створення, збереження та використання словесних головоломок. Архітектура проєкту побудована за принципом модульності та розподілу відповідальностей, що дозволяє незалежно розвивати гру, редактор рівнів, систему аналітики та модуль інтеграції зі штучним інтелектом.

Основна ідея полягає у створенні єдиної екосистеми для автоматизованого формування та розповсюдження кросвордів між різними платформами — мобільними пристроями, сервером і редактором розробника.

Архітектура Crosswordium має клієнт-серверну структуру (рис. 2.1), що є типовою для сучасних ігрових і контентних систем з динамічним оновленням даних [4]. У межах такої структури всі компоненти взаємодіють через централізовані сервіси даних і API-запити.

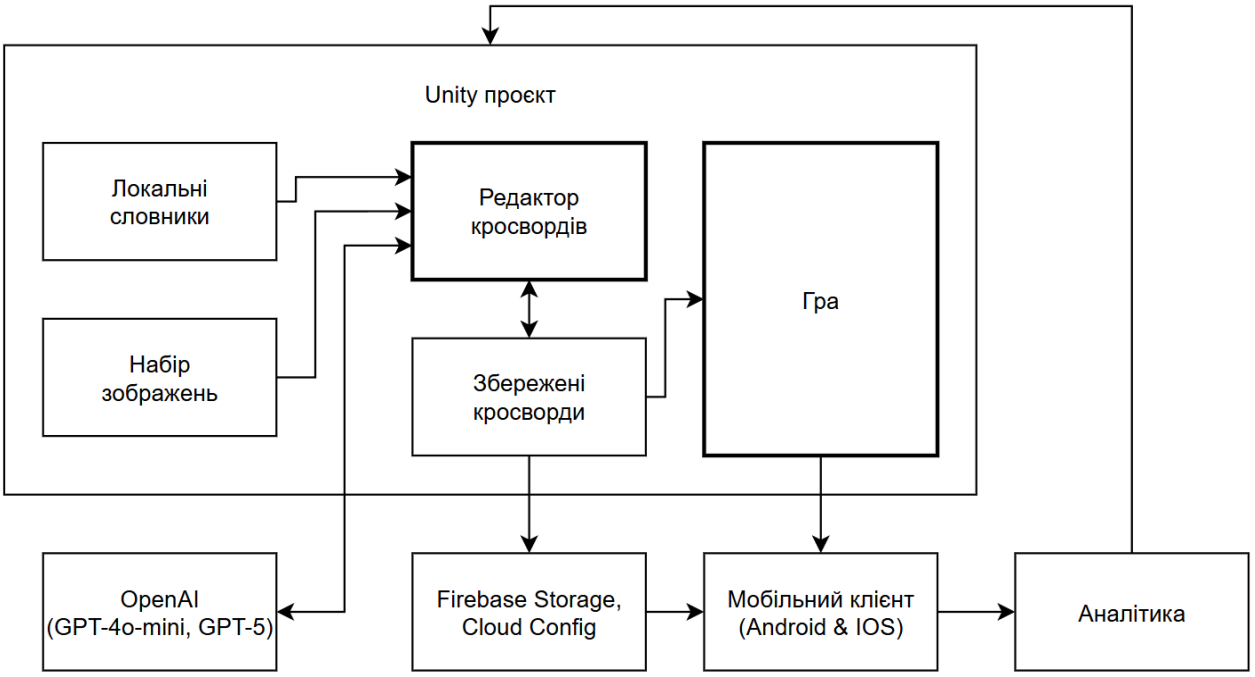

Рисунок 2.1 – *Загальна архітектура системи Crosswordium*

Система містить такі основні компоненти:

1. Unity-проєкт, який реалізує два режими роботи:
    a. гра для користувачів — мобільний застосунок під Android та iOS, що дає змогу проходити рівні, заробляти внутрішню валюту та взаємодіяти з ігровим контентом;
    b. редактор рівнів — внутрішній інструмент розробника для створення кросвордів, тестування генерації підказок і публікації готових рівнів на сервері.
2. Мобільний клієнт (Android / iOS build) — безпосередньо ігрова частина, з якою взаємодіє користувач. Завантажує рівні з сервера, кешує їх локально на пристрої. Включає систему монетизації (In-App Purchases, IAP) та базову офлайн-підтримку.
3. Серверна частина (хмарне сховище і API) — відповідає за збереження, оновлення та розповсюдження рівнів між користувачами. Забезпечує актуальність рівнів для гравців.
4. Модуль інтеграції зі штучним інтелектом (OpenAI API) — використовується в редакторі для генерації підказок і підбору слів за визначеними шаблонами.
5. Сервіс аналітики — збирає дані про поведінку користувачів під час гри: кількість відкритих і пройдених рівнів, середній час розв’язання, частоту використання підказок. Ці дані застосовуються для оцінки якості контенту й оптимізації складності рівнів.

Завдяки такій архітектурі Crosswordium поєднує етапи розробки, генерації контенту, публікації та збору статистики в єдиний інформаційний цикл. Усі компоненти системи взаємодіють через уніфіковані формати даних, що спрощує оновлення контенту, підтримку нових платформ і розширення функціональності гри.

Таким чином, архітектура проєкту забезпечує масштабованість, стабільну роботу на різних пристроях і можливість гнучкої інтеграції з інструментами штучного інтелекту. У наступному підрозділі буде детальніше розглянуто структуру даних, які використовуються для зберігання рівнів, слів і підказок у системі.

2.2 Структура і формати даних

У процесі роботи редактора рівнів Crosswordium використовується низка типів даних, які забезпечують збереження, редагування та обмін інформацією між його компонентами.

У більшості інтерфейсних елементів застосовуються стандартні типи — рядкові (string), числові (int, float), логічні (bool), перелічувальні (enum), а також базові об’єкти, що серіалізуються у формат JSON для внутрішніх налаштувань і параметрів.

Окрім цих базових елементів, у редакторі реалізовано й кастомні структури, спеціально розроблені для зберігання складніших об’єктів — таких як шаблони запитів до ШІ (ScriptableObjects), структура кросворду з його складовими класами, а також текстові словники для пошуку й добору слів.

Таке поєднання типових і спеціалізованих форматів забезпечує простоту роботи з даними та стабільність усієї системи під час створення і редагування рівнів.

Далі розглянемо основні власні формати та принципи їх використання.

2.2.1. ScriptableObjects

ScriptableObject — це спеціальний тип класу в середовищі Unity, призначений для зберігання даних незалежно від конкретних об’єктів сцени, що відповідає рекомендаціям офіційної документації Unity щодо організації конфігураційних та редакторських даних [5].

На відміну від звичайних компонентів (MonoBehaviour), ScriptableObject не потребує присутності в сцені та може існувати як окремий ресурс у проєкті. Це дозволяє централізовано зберігати параметри, конфігурації або шаблони даних, які використовуються різними об’єктами гри.

У проєкті Crosswordium ScriptableObjects застосовуються для зберігання шаблонів запитів (промптів), що використовуються при взаємодії з модулями штучного інтелекту. Такий підхід спрощує редагування текстів запитів і забезпечує можливість локалізації або зміни стилю без необхідності змінювати код.

Приклад реалізації класу ScriptableObject для шаблонів промптів:

```
namespace Generator
{
    [CreateAssetMenu(menuName = "AI/Prompt Template")]
    public class GenAIPromptInfo : ScriptableObject
    {
        [Multiline(8)]
        public string getWord;
        [Multiline(8)]
        public string selectWord;
        [Multiline(8)]
        public string getHint;
    }
}
```

Лістинг 2.1 – *Приклад ScriptableObject класу для зберігання шаблонів промптів*

Цей клас зберігається як окремий .asset-файл у проєкті та може бути багаторазово використаний у різних сценаріях генерації.

Кожне з трьох полів (getWord, selectWord, getHint) містить текстові шаблони, які під час виконання формуються у фінальні запити до ШІ-моделі, наприклад:

```
Згенеруй список слів для українського кросворду.
Кількість слів — {0}.
Кожне слово має містити рівно {1} літер.
Кожне слово має відповідати шаблону: {2}.
```

Лістинг 2.2 – *Приклад збереженого промпта у полі getWord*

У такий спосіб поля заповнюються з урахуванням майбутнього редагування або локалізації — можна створювати різні варіанти промптів для різних мов або для різних типів завдань.

Використання ScriptableObject у цьому контексті має кілька переваг:

- дозволяє зберігати шаблони запитів окремо від коду;
- спрощує тестування й оновлення текстів без перекомпіляції;
- дає змогу створювати набори промптів для різних мов, стилів або рівнів складності;

2.2.2. Структура та формат кросвордів

Для зберігання кросвордів у проєкті використовується власна структура даних, представлена у вигляді серіалізованого класу Crossword. Вона описує повний набір інформації, необхідної для відтворення рівня — назву, розмір сітки, слова, запитання та розташування літер.

Основна структура

Клас Crossword містить такі основні поля:

- name — назва кросворду;
- size — розмір сітки у клітинках;
- words — масив об’єктів типу Word, які описують окремі слова з позиціями, літерами та запитаннями;
- difficulty — визначається автоматично на основі назви або розміру сітки.

Допоміжні типи:

- Word — зберігає запитання (Question) та масив літер (Letter[]);
- Question — містить позицію на полі, тип (текст або зображення) та вміст;
- Letter — координату літери на сітці та сам символ.

```
[Serializable]
public class Crossword
{
    public string name;
    public Vector2Int size;
    public Word[] words;

    public Crossword(string name, Vector2Int size, Word[] words)
    {
        this.name = name;
        this.size = size;
        this.words = words;
    }
}
```

Лістинг 2.3 – *Фрагмент класу Crossword, який визначає основні поля структури рівня (скорочено).*

Ідентифікація та категоризація

Для впорядкування й швидкої адресації рівнів використовується структура CrossId, що поєднує:

- мову (ELanguage),
- рівень складності (EStage),
- числовий ідентифікатор рівня (Id).

На основі цих параметрів формується унікальний хеш та ім’я файлу рівня, наприклад: UA_Normal_5.bytes.

Серіалізація та формат зберігання

Після створення у редакторі кросворд серіалізується у формат JSON, який містить усі текстові та позиційні дані.

Додатково формується клас-обгортка, що включає:

- JSON-представлення кросворду;
- байтовий масив (byte[]) з допоміжними даними, наприклад із зображеннями чи іншими ресурсами.

Обидві частини об’єднуються у єдиний двійковий масив, який зберігається як файл із розширенням .bytes. Такий підхід забезпечує швидке завантаження й компактне зберігання даних без необхідності створювати окремі каталоги для медіа.

У процесі редагування використовується спрощена структура — список слів із додатковими параметрами для зручності розміщення, але у фінальному форматі зберігається лише узагальнена серіалізована версія класу Crossword.

Переваги такого підходу:

- забезпечується універсальність — кросворди можна зберігати, передавати або оновлювати без додаткових конвертацій;
- підтримується змішане зберігання текстових і графічних даних;
- структура легко серіалізується стандартними засобами Unity;
- можлива подальша компресія або шифрування .bytes файлів.

2.2.3. Текстові словники

Для роботи з лексичними даними у проєкті використовується система текстових словників, що зберігаються у вигляді простих файлів формату .txt. Такий підхід забезпечує гнучкість, простоту редагування та незалежність від складних баз даних чи зовнішніх сервісів.

Кожен словник містить список слів у текстовому вигляді, де кожен рядок відповідає одному слову. Символи зводяться до базового алфавіту обраної мови, а форматування відсутнє, що дозволяє швидко обробляти такі файли програмно.

Організація словників

Словники поділені на мовні та тематичні набори, що спрощує їх використання в різних режимах генерації:

- words_EN_base — усі англійські слова;
- words_UA_noun — лише іменники українською;
- words_UA_used — список слів, які вже були використані у створених рівнях.

Така структура дозволяє гнучко підключати потрібний набір даних залежно від мови або типу генерації.

Принцип роботи

Під час запуску редактора всі словники автоматично завантажуються та кешуються в оперативній пам’яті. Для підвищення швидкодії слова додатково групуються за кількістю літер, що суттєво пришвидшує пошук і перевірку відповідності шаблонам під час генерації кросвордів.

Наприклад, усі чотирилітерні слова можуть бути збережені в окремій групі, що дозволяє одразу вибирати відповідні елементи без потреби повного перебору.

Переваги підходу:

- простота оновлення — словник можна редагувати звичайним текстовим редактором;

- легка масштабованість — нові мови чи набори додаються створенням окремих файлів;
- незалежність від формату гри — той самий словник може використовуватись для різних проєктів або інструментів;
- ефективність — завдяки кешуванню пошук і фільтрація відбуваються миттєво навіть для великих списків.

2.3 Інформаційні потоки та взаємодія модулів Crosswordium

Система Crosswordium побудована таким чином, щоб усі основні етапи роботи з контентом — створення, генерація, збереження та використання кросвордів — відбувалися послідовно й узгоджено між різними модулями.

Інформаційні потоки поєднують редактор рівнів, модуль штучного інтелекту, хмарне сховище з рівнями та ігровий клієнт, утворюючи замкнений цикл від створення контенту до його використання гравцем (рис. 2.1).

Редактор рівнів

Редактор є вихідною точкою всієї системи. У ньому розробник створює нові рівні, добирає слова та генерує підказки.

Під час роботи редактор звертається до локальних словників — текстових списків слів, які завантажуються при запуску та кешуються в пам’яті для швидкого пошуку. Якщо потрібного слова або підказки немає, користувач може виконати запит до модуля штучного інтелекту (OpenAI API).

Такі запити є текстовими, тобто передають лише сформульовані інструкції (промпти) у вигляді звичайного рядка. Відповідь, яку повертає модель, також надходить у текстовому вигляді з мінімальним форматуванням. Редактор обробляє цей результат, виділяючи потрібні слова чи фрази, після чого може використовувати їх у процесі формування рівня.

Після завершення генерації дані кросворду зберігаються локально у внутрішній структурі редактора — у вигляді серіалізованого об’єкта, який може бути легко експортований для гри або резервного копіювання.

Хмарне сховище рівнів

Після створення кросворду розробник може розмістити його у хмарному сховищі, яке використовується як централізоване джерело ігрового контенту.

Сховище забезпечує зберігання опублікованих рівнів і доступ до них для клієнтських застосунків.

Ігрові клієнти можуть завантажувати необхідні файли безпосередньо із цього середовища, отримуючи актуальні версії контенту.

Такий підхід спрощує оновлення даних і забезпечує зручний механізм розповсюдження нових кросвордів між усіма користувачами.

Ігровий клієнт

Мобільна версія гри Crosswordium під час запуску отримує список доступних рівнів і за потреби завантажує їх із хмарного сховища.

Після завантаження файли зберігаються локально, що дозволяє користувачу грати без постійного з’єднання з мережею.

Ігровий клієнт використовує готові дані — слова, координати, підказки — без виконання будь-яких додаткових обчислень або генерації. Це забезпечує стабільну швидкодію та однакову якість контенту на всіх пристроях.

Зворотний потік даних

Після того, як гравці проходять рівні, система може збирати аналітичну інформацію — наприклад, які кросворди найчастіше відкриваються, скільки часу потрібно для розв’язання, як часто використовуються підказки.

Ці дані надсилаються до окремого аналітичного сервісу або зберігаються у хмарному сховищі, що дозволяє розробнику надалі вдосконалювати контент та баланс складності.

Таким чином, система Crosswordium поєднує всі етапи роботи з контентом в єдиний цикл — від генерації рівня у редакторі до його використання гравцем і подальшого аналізу результатів. Завдяки простій, але гнучкій структурі обміну даними забезпечується узгоджена робота всіх компонентів без складної серверної інфраструктури, що є важливою перевагою для незалежного розвитку проєкту.

2.4 Взаємодія з модулем штучного інтелекту

Застосування інструментів штучного інтелекту у проєкті Crosswordium виступає не основним, а допоміжним елементом, який розширює можливості редактора рівнів і підвищує ефективність створення контенту.

Інтеграція з мовною моделлю OpenAI дозволяє автоматизувати окремі етапи роботи — генерацію нових слів, добір варіантів зі словників, створення текстових підказок, а також попередню перевірку правильності вже сформованого кросворду.

Такий підхід значно спрощує процес розробки, але водночас потребує подальшого вдосконалення механізмів контролю якості результатів.

Формування запиту

Взаємодія з моделлю здійснюється через текстові запити (prompts), які формуються безпосередньо в редакторі рівнів.

Для зручності та уніфікації всі шаблони запитів зберігаються у спеціальних об’єктах типу ScriptableObject, що дає змогу швидко змінювати або локалізувати їх без редагування коду. Приклад шаблону промпту наведено у лістингу 2.2.

Редактор підставляє у шаблон конкретні параметри — наприклад, кількість слів, довжину чи заданий шаблон символів — і передає готовий текст до API для обробки.

Передача та отримання відповіді

Обмін із системою OpenAI здійснюється через стандартизований HTTP-інтерфейс відповідно до офіційної документації OpenAI [1], який містить сформований промпт і базові параметри генерації: температуру випадковості (temperature), кількість токенів (max_tokens) та модель, яку потрібно використати. Відповідь надходить у вигляді звичайного тексту, який містить список слів або підказок, сформованих згідно з умовами запиту.

Клас, який відповідає за роботу з API, описано у додатку А (GenAI.cs). Саме цей компонент реалізує створення запиту, обробку відповіді, очищення результату та передачу отриманих даних до відповідного модуля редактора.

Обробка результатів

Отриманий результат попередньо аналізується редактором:

- перевіряється відповідність мові, довжині або заданому шаблону;
- відсіюються повтори та некоректні варіанти;
- за потреби користувач може виконати ручне редагування або доопрацювання підказок.

Після валідації дані можуть бути збережені у словнику або використані безпосередньо під час формування нового рівня.

Застосування штучного інтелекту

Модуль ШІ може використовуватись у кількох напрямках роботи редактора:

- генерація слів за заданими параметрами (мова, довжина, шаблон);
- підбір слів зі словника з урахуванням контексту або складності;
- створення підказок до вже відомих слів різними стилями;
- перевірка готового кросворду, наприклад, оцінка коректності або унікальності завдань.

Таке поєднання автоматичних і ручних етапів дозволяє отримати збалансований результат — редактор залишається керованим користувачем, але водночас підтримує інтелектуальну допомогу при створенні контенту.

На рисунку 2.2 подано схему взаємодії редактора з модулем штучного інтелекту на прикладі методу *GenerateHints*. Редактор формує текстовий запит, надсилає його до OpenAI API, отримує відповідь, після чого виконує аналіз і збереження даних у внутрішніх структурах.

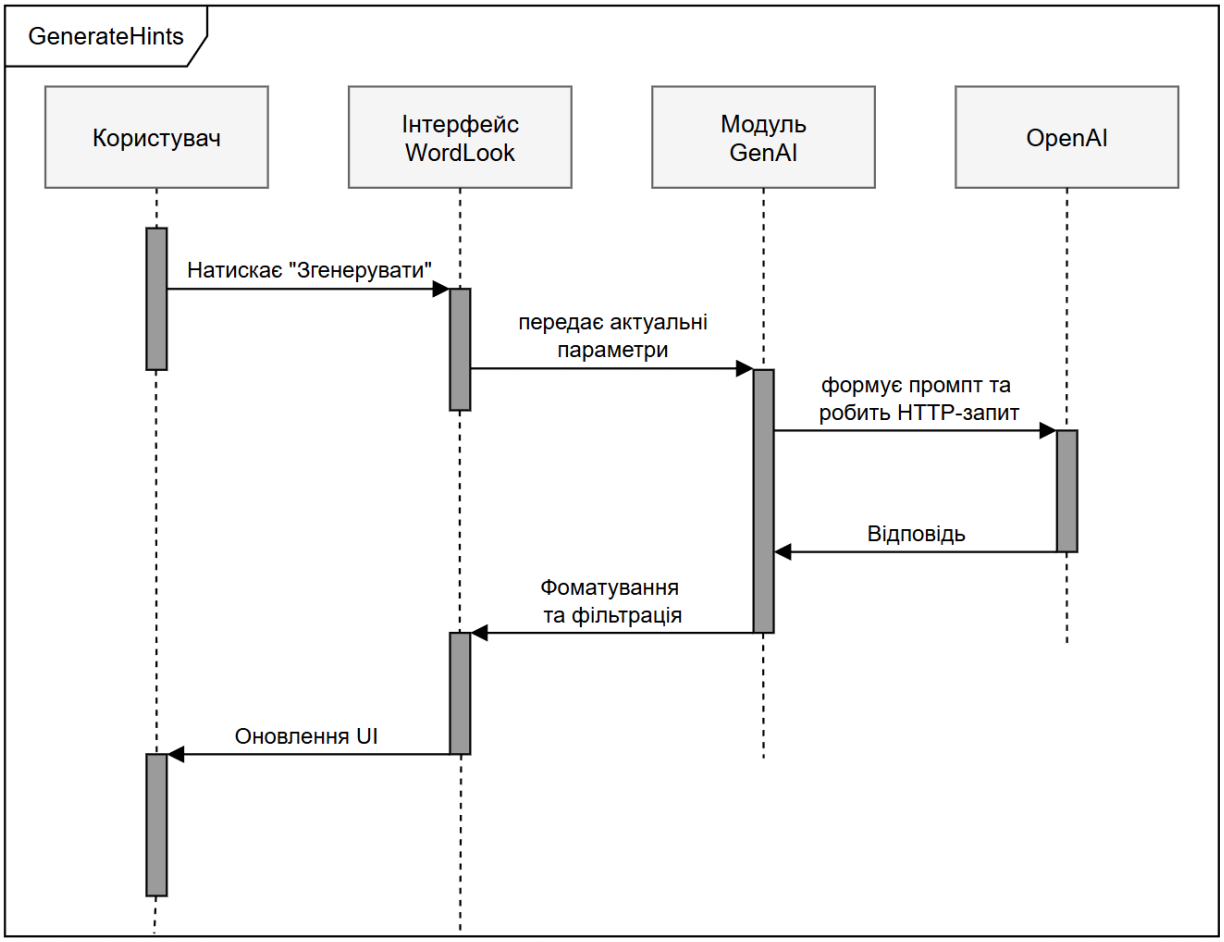

Рисунок 2.2 – *UML-діаграма послідовності взаємодії редактора Crosswordium із модулем штучного інтелекту (метод GenerateHints)*

Таким чином, використання штучного інтелекту в Crosswordium не замінює традиційні методи генерації, а доповнює їх, створюючи основу для подальших досліджень і вдосконалення процесів автоматизації у створенні словесних ігор.

Висновки до розділу

У цьому розділі було розглянуто інформаційне забезпечення системи Crosswordium, яке поєднує архітектуру проєкту, структури даних і механізми взаємодії між його компонентами. Описано основні типи даних, що використовуються в редакторі рівнів, зокрема ScriptableObjects, структури кросвордів і текстові словники, а також наведено принципи їх організації та обміну. Проаналізовано інформаційні потоки системи та схему взаємодії між редактором, ігровим клієнтом, хмарним сховищем і модулем штучного інтелекту.

Окрему увагу приділено інтеграції з мовною моделлю OpenAI, яка розширює можливості системи в частині генерації слів і підказок, підвищує ефективність створення контенту та відкриває перспективи для подальшого вдосконалення процесів автоматизації у грі Crosswordium.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Алгоритм створення кросворду

Алгоритм створення кросворду в системі Crosswordium визначає повну логіку формування нового рівня — від ініціалізації параметрів до публікації готового файлу у хмарному сховищі.

У загальному випадку побудова кросворду передбачає послідовне розміщення слів у сітці з дотриманням правил перетину, цілісності та лінгвістичної узгодженості. Проте для Crosswordium цей процес має низку специфічних особливостей, зумовлених вимогами до повного заповнення поля, візуальної цілісності та можливості автоматизованого редагування.

Система побудована за модульним принципом, де кожен компонент відповідає за окремий етап роботи: добір слів, створення запитань, перевірку, збереження та публікацію. Такий підхід забезпечує масштабованість і розширюваність редактора, а також дозволяє поєднувати ручне редагування з автоматизованими процедурами генерації.

На відміну від класичних кросвордних редакторів, у Crosswordium застосовується підхід суцільного заповнення сітки, коли всі клітинки поля мають бути частиною певного слова, а не лише вибраних ділянок. Це підвищує щільність інформаційного наповнення та створює більш динамічну структуру рівнів.

Додатково система враховує положення майбутніх полів під питання та можливість їхнього інтегрованого відображення в інтерфейсі гри.

Таким чином, алгоритм створення кросворду в Crosswordium можна розглядати як уніфікований процес, що поєднує класичні принципи побудови лінгвістичних головоломок із сучасними підходами до автоматизації та використання штучного інтелекту.

Послідовність етапів створення рівня

1. Ініціалізація параметрів.

   Користувач (редактор) задає основні налаштування нового рівня:

   - розмір поля кросворду (наприклад, 10×10 або 12×12);
   - мову створення (українська, англійська);
   - належність рівню складності.

   Після цього редактор переходить до створення або завантаження сітки.

2. Створення або завантаження сітки.

   На цьому етапі редактор може:

   - створити нову сітку з нуля, задавши параметри структури;
   - або завантажити існуючу з локального списку збережених шаблонів.

У будь-якому випадку формується прямокутна структура з клітинок, у які згодом розміщуються слова.

Алгоритм забезпечує, щоб:

- кожна послідовність двох і більше клітинок утворювала слово;
- уся сітка була заповнена або перетинами слів, без порожніх зон;
- слова не розташовувалися занадто близько паралельно одне одному;
- резервувалися позиції для майбутніх полів під питання.

3. Заповнення сітки словами (модуль WordLook).

У процесі редагування кросворду редактор працює з візуальним інтерфейсом, у якому одночасно відображається поточна сітка та список доступних слів. Редактор виділяє слово, яке бажає заповнити або змінити, після чого воно передається до модуля WordLook. Код реалізації модуля WordLook наведено в додатку Б.

Залежно від того, чи містить вибране слово вже встановлені літери або перетинається з іншими словами, модуль автоматично формує шаблон (наприклад, [Т][?][?][?][О]). Отриманий шаблон можна використати для фільтрації або пошуку відповідних варіантів у словнику, або скинути його для ручного введення нового слова.

Після вибору потрібного варіанта редактор застосовує слово до сітки, і вона автоматично оновлюється з урахуванням усіх перетинів і нових символів. Таким

чином забезпечується динамічне заповнення поля, у якому кожна зміна одразу впливає на подальший підбір слів та узгодженість кросворду.

4. Формування полів під питання.

Після заповнення сітки редактор додає місця для підказок.

Кожне поле має форму прямокутника (від 1×1 до кількох клітинок) і розміщується поруч із відповідним словом, яке воно описує.

Система автоматично контролює, щоб ці поля не перекривали слова й не порушували структуру сітки.

5. Підбір питань до слів (модуль QuestionLook).

Для кожного слова створюється текстова або графічна підказка.

Процес виконується через модуль QuestionLook, який має власний інтерфейс. Код модуля QuestionLook наведено в додатку В.

Модуль дозволяє:

- вводити питання вручну;
- автоматично згенерувати варіанти запитань через модуль штучного інтелекту GenAI;
- при використанні візуальної підказки — вказати назву зображення, що завантажується з локальних асетів.

6. Перевірка якості та складності.

Після формування сітки й підказок редактор виконує кілька загальних перевірок:

- структурну: на коректність перетинів, відсутність пропусків і логічну зв’язність усіх слів;
- лінгвістичну: наявність слів у словнику, відсутність русизмів чи неетичних лексем;
- аналітичну: перевірку складності рівня на основі середньої довжини слів, частоти використання та кількості перетинів.

Для спрощення перевірки формується запит до ШІ, який оцінює рівень складності й дає рекомендації щодо можливих виправлень.

7. Редагування та уточнення.

У разі виявлення неточностей або потреби в доопрацюванні редактор здійснює правки. Може бути змінено слово, підказку або місце її розташування.

8. Надання назви та збереження рівня.

Після фінальної перевірки кросворду присвоюється назва відповідно до правил, поданих у підрозділі 2.2.2, після чого рівень зберігається локально через модуль CrossLook.

Цей модуль відповідає за серіалізацію об’єкта кросворду у файл і формування метаданих (ідентифікатор, дата створення, мова, складність).

9. Публікація та синхронізація.

Після перевірки та збереження рівень може бути опублікований у хмарному сховищі Firebase. Після цього оновлюється загальний список доступних кросвордів на сервері, і користувачі додатка можуть завантажити новий рівень для проходження.

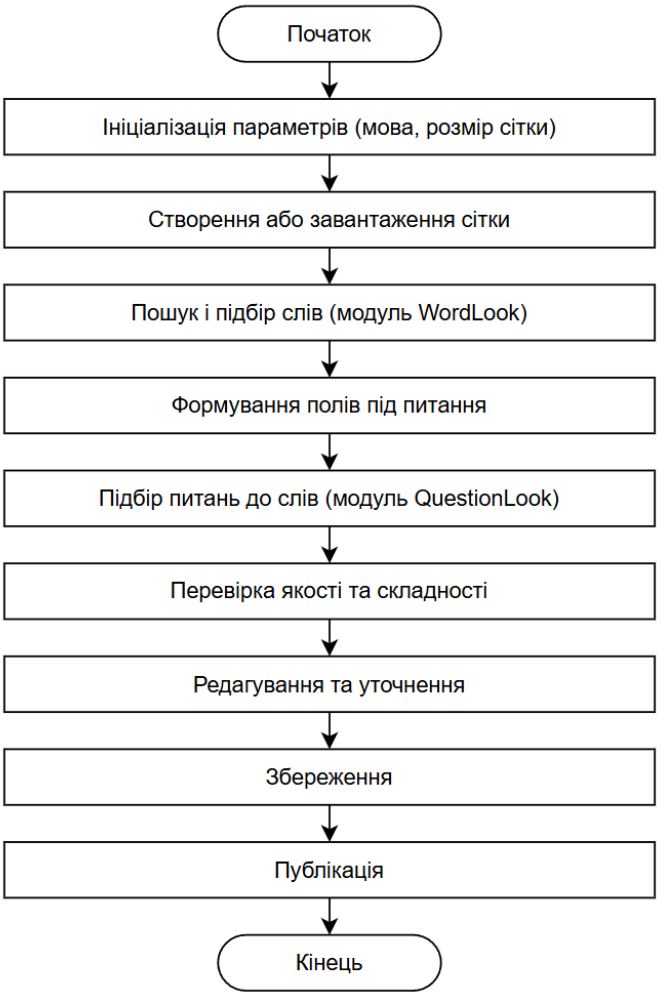

Рисунок 3.1 – *Узагальнена схема алгоритму створення кросворду*

Алгоритм створення кросворду реалізує цілісний цикл генерації рівня — від формування сітки до публікації у сховищі.

Завдяки модульній структурі (WordLook, QuestionLook, CrossLook) система забезпечує розділення функцій, що спрощує розроблення, підтримку та подальше розширення редактора.

Поєднання ручних дій редактора та автоматизованих процедур підвищує якість контенту, зберігаючи творчий контроль і можливість адаптації під різні потреби.

3.2 Алгоритми пошуку в локальних словниках

Мета підрозділу — описати принципи та алгоритми добору слів для заповнення кросворду відповідно до шаблону, заданої мови та рівня складності.

У системі Crosswordium реалізовано три взаємодоповнювальні підходи:

- пошук слова у локальному словнику;
- генерацію слова через модель штучного інтелекту;
- комбінований метод, який поєднує обидва підходи.

3.2.1. Пошук локальним методом

Пошук слів у локальному словнику є базовим і найшвидшим способом заповнення кросворду в системі Crosswordium. Він реалізований як незалежний алгоритм у складі модуля WordLook і використовується у випадках, коли потрібно знайти слово, що точно відповідає певному шаблону або набору обмежень. Основна мета цього методу — забезпечити стабільність та швидкість підбору без залучення зовнішніх ресурсів. Подібні підходи до пошуку за шаблоном використовуються у класичних задачах комбінаторної оптимізації та словесних головоломок [6].

Загальна логіка роботи

Модуль WordLook отримує від редактора початкові вхідні дані:

- шаблон слова (наприклад, [T][?][?][?][O]);
- мову (українська, англійська тощо);
- тип словника — повний словник або лише іменники (для тематичних кросвордів);
- параметр унікальності — чи дозволено повторне використання слова.

Після цього відбувається послідовність кроків:

1. Вибір кешованої версії словника.

Залежно від мови та типу словника, модуль підключає відповідну локальну базу (наприклад, words_EN_base або words_UA_noun). Використання кешованих словників зменшує час доступу та дозволяє швидко проводити пошук навіть при великій кількості слів.

2. Попереднє фільтрування за довжиною.

Серед усіх слів обираються лише ті, що мають довжину, ідентичну довжині шаблону. Це значно зменшує обсяг подальших перевірок.

3. Перевірка відповідності шаблону.

Для кожного слова у відфільтрованому списку здійснюється посимвольне порівняння з шаблоном. Якщо на певній позиції в шаблоні задана конкретна літера, слово повинно мати ту саму; позиції зі знаком «?» допускають будь-який символ. Таким чином, формується набір слів, які точно відповідають критеріям.

4. Фільтрація за унікальністю.

Якщо задано параметр «уникати повторів», із результатів вилучаються слова, що вже використовувалися раніше у цьому або попередніх рівнях.

5. Формування списку результатів.

Отриманий перелік передається редактору для відображення у візуальному інтерфейсі. Редактор може вибрати конкретне слово зі списку або скористатися функцією «Вибрати випадкове», після чого слово вставляється у відповідну позицію

сітки. У разі потреби користувач може повторно оновити список або змінити критерії пошуку.

Такий метод дозволяє швидко отримати релевантний результат без підключення до зовнішніх API, що є критично важливим для роботи в офлайн-режимі або при великій кількості запитів. Завдяки фільтрації за шаблоном, довжиною та унікальністю він забезпечує високу точність, хоча не може виходити за межі попередньо визначеного словника.

Переваги локального методу:

- висока швидкість виконання (результат формується за частки секунди);
- повна автономність — не потребує інтернет-з’єднання;
- контроль якості словникової бази (усі слова попередньо перевірені);
- можливість обмеження типу слів (наприклад, лише іменники).

Недоліки:

- обмежений діапазон нових слів — система не здатна генерувати неологізми, сучасні терміни чи власні назви, що не входять до словника;
- у випадку малого словника пошук може не дати жодного результату за складними шаблонами;
- у випадку великого цілісного словника виникає протилежна проблема — надмірна кількість маловживаних або малозмістовних слів, які формально відповідають шаблону, але є непридатними для ігрового контенту (наприклад, діалектизми, застарілі або технічні терміни);
- не враховується тематика рівня — добір здійснюється виключно за формальними ознаками;
- без підключення зовнішніх джерел складно забезпечити лінгвістичне різноманіття й креативність завдань.

Таким чином, локальний метод добору слів виступає базовим механізмом, що гарантує стабільність і передбачуваність результатів, а в подальших підрозділах

(3.2.2–3.2.3) розширюється можливістю використання моделей штучного інтелекту для підвищення креативності та тематичної гнучкості системи.

3.2.2. Генерація слів через ШІ

Генерація слів за допомогою моделей штучного інтелекту використовується у системі Crosswordium як альтернативний метод, коли локальний словник не дає задовільних результатів або потрібно отримати нові, тематично релевантні варіанти.

Загальна логіка роботи

1. Формування запиту.

Редактор або система передає до модуля GenAI шаблон слова (наприклад, [T][?][?][?][O]), зазначає мову генерації та додаткові параметри, зокрема кількість слів, яку необхідно отримати в результаті.

2. Надсилання промпту.

На основі отриманих даних система формує текстовий запит до мовної моделі.

Приклад шаблону промпту наведено у листингу 2.2. Запит надсилається через API до обраної моделі (GPT-4o-mini, GPT-5 тощо).

3. Отримання і обробка відповіді.

Після обробки запиту модель повертає набір слів. Система виконує додаткову очистку тексту — видаляє зайві символи, розділові знаки та повтори, після чого проводить перевірку кожного слова за шаблоном та повертає фінальний список слів.

4. Відображення у редакторі.

Редактор отримує оновлений перелік кандидатів — у якому можуть бути як локальні, так і нові слова, згенеровані штучним інтелектом. Користувач може обрати потрібне слово, або скористатися функцією «Вибрати випадкове», після чого слово автоматично застосовується до сітки.

Практичне порівняння генерації різними моделями

Для оцінки ефективності генерації слів було проведено експериментальне тестування з використанням кількох моделей штучного інтелекту. Методика оцінювання базується на загальних рекомендаціях щодо тестування мовних моделей у прикладних системах [7].

Було застосовано три шаблони: [T][?][?][?][O], [?][A][?][A][?][A] та [?][?][P][?][?][?][H][?]. У кожному запиті моделі пропонувалося згенерувати 10 слів, які відповідають заданому шаблону. В усіх випадках використовувалася однакова інструкція — текстовий запит, сформований за шаблоном, наведеним у листингу 2.2.

Запити надсилалися до різних моделей ШІ, частина яких тестувалася з варіаціями параметра temperature для оцінки впливу стохастичності на результати генерації. В окремих експериментах використовувалися інтегровані чат-сервіси, що забезпечило доступ до додаткових моделей без прямої інтеграції через API, оскільки реалізація повноцінного API-доступу виходила за межі завдання дослідження.

Для кожного запиту оцінювалися такі показники:

- час обробки (у секундах);
- кількість слів, що відповідають шаблону;
- кількість змістовних слів, тобто таких, що не є вигаданими або граматично некоректними.

У таблиці 3.1 наведено результати порівняння для кожної моделі ШІ та кожного шаблону окремо, а також подано усереднені показники — за моделлю та за шаблоном. Отримані дані дають змогу узагальнено оцінити стабільність та відтворюваність результатів генерації, а також сформувати основу для подальшого аналізу ефективності застосованих підходів.

Таблиця 3.1 – *Порівняння генерації слів різними моделями ШІ*

№	Модель	temperature	Час, с	[T][?][?][?][O]		[?][A][?][A][?][A]		[?][?][P][?][?][?][H][?]		Усереднено по моделі	
				Відповідних шаблону, %	Кількість змістовних, %	Відповідних шаблону, %	Кількість змістовних, %	Відповідних шаблону, %	Кількість змістовних, %	Відповідних шаблону, %	Кількість змістовних, %
1	gpt-4o-mini	0,7	4	44	18	65	18	10	4	40	13
2	gpt-4o-mini	1	4	60	14	16	4	2	0	26	6
3	gpt-4o-mini	0,2	4	58	20	24	8	6	0	29	9
4	gpt-5-instant	auto	5	90	50	100	40	10	0	67	30
5	gpt-5-thinking	auto	116	100	80	100	80	100	40	100	67
6	copilot (gpt-4)	auto	6	80	50	90	40	20	10	63	33
7	gemini 2.5 flash	auto	5	100	60	90	30	20	10	70	33
8	Sonet 4.5	auto	8	90	50	90	60	10	2	63	37
Усереднено по шаблону				**78**	**43**	**72**	**35**	**22**	**8**	**57**	**29**

Аналіз результатів генерації

На основі результатів, поданих у таблиці 3.1, можна зробити кілька узагальнених висновків щодо ефективності генерації слів різними мовними моделями.

У середньому, мовні моделі демонструють обмежену здатність точно дотримуватись шаблонів, особливо при роботі зі складними структурами. Для коротких шаблонів, таких як [T][?][?][?][O], відсоток відповідних слів становить близько 78 %, тоді як для довших (наприклад, [?][?][P][?][?][?][H][?]) точність різко падає до 22 %. Це свідчить про те, що зі зростанням довжини слова моделі втрачають узгодженість із шаблоном, особливо коли потрібно утримувати певні позиційні літери.

Водночас кількість змістовних слів, тобто таких, що реально існують у мові та мають осмислене значення, становить лише половину або менше від тих, що формально задовольнили шаблон (усереднено — близько 29 %). Це вказує, що навіть коли модель правильно відтворює структуру слова, значна частина результатів не є

корисною з точки зору лексичної наповненості — часто це випадкові послідовності літер або морфологічно некоректні утворення.

Параметр temperature продемонстрував помірний вплив на якість генерації. Як видно з результатів моделі GPT-4o-mini, найкращі показники досягаються при значенні temperature = 0,7, що дає баланс між різноманітністю і структурною точністю. При зниженні до 0,2 модель видає надто однотипні результати й “застрягає” на повторенні одних і тих самих слів, а при підвищенні до 1,0 — починає занадто відходити від шаблону, утворюючи помилки або зайві символи. Таким чином, рекомендованим значенням для генерації слів у Crosswordium є temperature ≈ 0,7.

Серед усіх протестованих моделей найкращий результат показала GPT-5-thinking — вона досягла 100 % відповідності шаблонам і середньо 67 % змістовних слів, проте час обробки запиту становив понад 100 секунд, що у 15–20 разів більше, ніж у швидких моделей. Це вказує на те, що глибший контекстуальний аналіз покращує якість результатів, але вимагає значно більших обчислювальних ресурсів.

Більшість сучасних мовних моделей виконують генерацію слів за шаблоном у середньому за кілька секунд (переважно в діапазоні 4–6 с), що робить їх придатними для інтерактивного використання у редакторі Crosswordium. При цьому спостерігаються певні відмінності в якості результатів: моделі, орієнтовані на швидку обробку без глибокого аналізу, забезпечують прийнятну точність, але часто повертають обмежену кількість змістовних слів. Натомість новіші покоління моделей демонструють значно кращу узгодженість із шаблонами та більш осмислені результати

Тому для оперативних запитів доцільно використовувати легкі моделі з помірною глибиною аналізу, а для перевірки складних шаблонів — просунуті версії GPT-5-рівня, які здатні забезпечити вищу якість генерації при довших обчисленнях.

Переваги:

- можливість створення нових і тематично релевантних слів, недоступних у локальному словнику;
- вища точність і змістовність у новіших моделях (GPT-5-серія);
- здатність одночасно генерувати слова та короткі підказки, що прискорює створення контенту.

Недоліки:

- низька точність у швидких моделей, які жертвують якістю заради часу обробки;
- значна затримка у складніших моделей через глибший аналіз;
- потреба в перевірці результатів, оскільки лише частина слів, що відповідають шаблону, є справді змістовними;
- залежність від онлайн-доступу для виконання запитів.

3.2.3. Комбінований метод

Комбінований метод підбору слів поєднує переваги локального пошуку та генерації через штучний інтелект. Він використовується у системі Crosswordium як основний режим роботи, оскільки забезпечує баланс між швидкістю отримання результатів, змістовністю та мовною різноманітністю. На відміну від окремих методів, комбінований підхід дозволяє одночасно використовувати наявні словникові бази й отримувати від ШІ додаткові слова або уточнення до вже знайдених.

Загальна логіка роботи

5. Формування початкового списку слів.

Модуль WordLook здійснює первинний пошук у локальному словнику за заданим шаблоном і параметрами (мова, частина мови, унікальність). Результатом є список базових варіантів, що відповідають формальним вимогам.

6. Передача списку до модуля GenAI.

Сформований список передається у запит до модуля GenAI, який звертається до мовної моделі (наприклад, GPT-4o-mini або GPT-5). У промпті зазначаються: шаблон, мова, контекст (тема рівня) і список початкових варіантів. Модель має завдання розширити або оптимізувати цей список — додати нові слова, усунути невалідні, впорядкувати за поширеністю чи змістовністю.

7. Отримання й аналіз відповіді.

Після надсилання запиту система отримує від ШІ-модуля згенеровану відповідь, яка містить опрацьований перелік слів. Отримані дані проходять автоматичну фільтрацію: усуваються дублікати, некоректні або нерелевантні елементи, а також проводиться впорядкування за рівнем відповідності шаблону й змістовності. У результаті формується релевантно відсортований список слів, готовий до відображення в інтерфейсі редактора. Залежно від типу запиту, у відповіді можуть також міститися додаткові варіанти, тематичні уточнення або короткі підказки до кожного слова, що підвищує зручність подальшої роботи редактора.

8. Відображення у редакторі.

Редактор отримує оновлений перелік кандидатів — у якому можуть бути як локальні, так і нові слова, згенеровані штучним інтелектом. Користувач може обрати потрібне слово, або скористатися функцією «Вибрати випадкове», після чого слово автоматично застосовується до сітки.

Переваги комбінованого методу:

- забезпечує найкращий баланс між точністю локального пошуку й креативністю ШІ;
- дозволяє підвищити тематичну релевантність завдяки аналізу контексту;
- дає більше різноманіття слів, не втрачаючи контроль якості;
- скорочує час підготовки контенту, оскільки більшість рутинних операцій виконується автоматично;

- зменшує кількість малозмістовних або непридатних слів за рахунок післяобробки результатів ШІ.

Недоліки:

- потребує доступу до інтернету та активного API-підключення;
- можливе варіювання результатів при різних генераціях через стохастичну природу моделей;
- вимагає додаткової перевірки з боку редактора, щоб уникнути помилкових або контекстно некоректних варіантів;
- збільшує загальний час відповіді у порівнянні з чисто локальним пошуком.

Комбінований підхід став основним методом у Crosswordium, оскільки поєднує надійність локальної бази з динамікою й творчістю штучного інтелекту. Це дозволяє системі створювати не лише технічно правильні, а й лінгвістично цікаві кросворди, адаптовані до тематики рівня й цільової аудиторії.

3.3 Аналіз і перевірка якості слів за допомогою штучного інтелекту

У цьому підрозділі розглядаються можливості застосування мовних моделей штучного інтелекту для оцінки правильності, етичності та складності слів, згенерованих під час формування кросвордів у системі Crosswordium. Аналіз проводився на прикладі україномовної версії кросвордів, що є основною для даного проєкту. Тому особливу увагу приділено таким аспектам, як україномовність слів, наявність русизмів, діалектизмів або культурно чутливих термінів, а також загальна мовна та етична відповідність контенту.

Метою аналізу є забезпечення мовної коректності, культурної нейтральності та збалансованої складності завдань для різних рівнів гравців. Використання ШІ для перевірки таких параметрів дозволяє значно скоротити час ручного редагування та підвищити якість кінцевого контенту в Crosswordium.

3.3.1. Перевірка етичності та мовної коректності слів за допомогою ШІ

Одним із важливих етапів валідації контенту у системі Crosswordium є перевірка згенерованих або підібраних слів на етичність, мовну правильність та відповідність українській культурній нормі. Для цього використовується модуль на основі мовної моделі GenAI, який здатен аналізувати слово або список слів за кількома критеріями. Такий підхід дозволяє автоматично фільтрувати небажані або сумнівні варіанти ще до їх публікації в редакторі рівнів.

Загальний принцип перевірки

Редактор або система формує запит, що містить список слів для аналізу, і передає його до ШІ-модуля з короткою інструкцією: наприклад, “Перевір кожне слово, чи є воно етичним, українським і придатним для навчального або розважального контенту.” Модель оцінює кожен елемент і повертає висновок у вигляді короткого коментаря або маркера (“етичне”, “неетичне”, “русизм”, “сумнівне”). Результати аналізу зберігаються в таблиці перевірки, на основі якої система або сам редактор вирішує — залишити, замінити чи вилучити слово. Такий підхід особливо зручний при автоматичному поповненні словників новими словами з відкритих джерел або під час генерації великих наборів кросвордів.

Для перевірки ефективності аналізу було сформовано набір із восьми українських слів, різних за походженням, стилістичним забарвленням і сферою вживання. Оцінку виконано за двома критеріями — етичність (відсутність образливих чи чутливих конотацій) та мовна коректність (відповідність нормам сучасної української мови). Результати перевірки подано в таблиці 3.2.

Таблиця 3.2 – *Приклад оцінки етичності та мовної коректності слів за допомогою ШІ*

Слово	Оцінка етичності	Оцінка мовної коректності	Коментар (пояснення ШІ)	Висновок щодо використання
Райдуга	нейтральне	нормативне українське слово	Повністю українське, хоча іноді помилково сприймається як русизм. Є усталене літературне слово.	можна використовувати.
Одіяло	нейтральне	русизм (правильно: ковдра)	Поширене у побуті, але не належить до нормативної української лексики.	небажано
Ґазда	нейтральне	діалектизм	Поширене в Галичині слово “хазяїн”, може вживатися у регіональних контекстах.	можна використовувати.
Фронт	культурно чутливе	нормативне	Має військово-політичну конотацію, нейтральне у географічному значенні.	можна використовувати з контекстом.
Снігурочка	нейтральне	русизм (культурне запозичення)	Російський персонаж; недоречно в україномовній грі.	небажано
Жидівка	образливе	нормативне (застаріле)	Етнонім із негативним забарвленням, замінюється на “єврейка”.	заборонити
Татарин	культурно чутливе	нормативне	Етнонім; потребує обережного використання у нейтральних контекстах.	з обережністю.
Галичина	нейтральне	нормативне	Географічна назва, нейтральна.	можна

Отримані результати свідчать, що моделі штучного інтелекту можуть достатньо точно визначати мовний статус і культурну прийнятність більшості слів. Система безпомилково ідентифікує очевидні русизми (одіяло, снігурочка) та образливі етноніми (жидівка), пропонуючи коректні заміни. Водночас ШІ демонструє розуміння контекстуальної чутливості — наприклад, для слів фронт або татарин дається рекомендація “використовувати з обережністю”, а не повна заборона.

Такі перевірки дають змогу автоматично відсівати лінгвістично або етично непридатні слова, зменшуючи потребу у ручній модерації. Разом із тим, для спірних або контекстно залежних випадків (діалектизми, власні назви) доцільно зберігати редакторське підтвердження перед публікацією кросворду.

3.3.2. Оцінювання складності слів за допомогою ШІ

Окрім лінгвістичної та етичної перевірки, у системі Crosswordium штучний інтелект застосовується також для оцінки складності слів і кросвордів загалом.

Така оцінка допомагає визначати рівень завдання (від Easy до Hard), адаптувати контент до різних груп гравців і підтримувати баланс між навчальною та розважальною складовими гри. ШІ аналізує як окремі слова, так і їх підказки, що дозволяє оцінити не лише лексичну, а й когнітивну складність сприйняття.

Загальний принцип оцінки

Редактор формує запит до модуля GenAI, у якому вказує основні дані кросворда — список слів і відповідні підказки. На основі цього формується текстовий промпт, подібний до використаного в генерації (див. листинг 2.2), але зі зміненим завданням. Модель обробляє дані та повертає оцінки у вигляді числових значень і коротких коментарів, що пояснюють, чому певне слово є легшим або складнішим. Таку перевірку можна виконувати для кожного слова окремо або для цілого кросворда, якщо потрібно оцінити його середню складність. Отримані результати використовуються для автоматичного визначення рівня складності кросворду перед публікацією.

Експериментальна перевірка валідності оцінки

Для перевірки достовірності такого підходу було проведено експеримент: було вибрано 15 слів із різних рівнів складності та створено до них короткі підказки.

ШІ-модель отримала два завдання:

- оцінити складність самих слів (без урахування підказок);
- оцінити комбінацію “слово + підказка”, тобто наскільки важко гравцю буде розгадати слово за поданим запитанням.

Отримані результати було зіставлено з реальними аналітичними даними гри Crosswordium, які містили:

- кількість користувачів, що успішно пройшли рівень;
- кількість підказок, використаних для кожного слова.

На основі цих даних було розраховано зведений показник складності в 100-бальній шкалі, який потім порівняли з оцінками ШІ.

Таблиця 3.3 – *Приклад порівняння оцінки складності ШІ та аналітичних даних гравців*

Слово	Запитання	Складність з аналітики	ШІ без контексту	Відхи-лення	ШІ з контекстом	Відхи-лення
емірати	Об'єднані Арабські …	0	28	28	4	4
яхта	судно для відпочинку та туризму	2	5	3	6	4
чумак	історичний український торговець сіллю та іншими продуктами	5	42	37	36	31
парфуми	зображення*	8	22	14	18	10
автономія	конституційна особливість Криму відносно областей України	10	34	24	28	18
галерея	художній музей із творами мистецтва для огляду	13	18	5	20	7
атмосфера	зовнішня газова оболонка планети, утримувана силою тяжіння	15	30	15	24	9
талант	видатна природна здібність людини	20	12	8	10	10
рупор	труба у вигляді зрізаного конуса або піраміди для посилення звуку	25	36	11	30	5
куранти	старовинний баштовий або стінний годинник, бій якого супроводжується музикою	30	52	22	44	14
цитоплазма	напіврідкий вміст клітини	45	60	15	56	11
елегія	ліричний вірш задумливого, сумного характеру	60	48	12	42	18
рефлексія	роздуми людини над власним душевним станом	75	72	3	63	12
агностицизм	погляд, що існування Бога та божественного є невідомим та неможливим до пізнання	83	80	3	75	8
комплексність	властивість системи складатися з багатьох взаємопов’язаних елементів	90	83	7	78	12
середнє				**14**		**12**
медіана				**12**		**10**
максимум				**37**		**31**

Порівняння результатів оцінки складності, отриманих від мовної моделі ШІ, із реальними аналітичними даними гравців показало загалом високий рівень відповідності.У більшості випадків штучний інтелект досить точно розпізнає складність слова, особливо коли враховується контекст у вигляді підказки або визначення.

Середнє відхилення між аналітичною оцінкою та результатом ШІ без контексту становило 14 пунктів, а при використанні підказок зменшилося до 12 пунктів. Медіанне відхилення відповідно склало 12 і 10 пунктів, а максимальне — знизилося з 37 до 31. Це свідчить, що включення контексту в запит (аналіз пари “слово + підказка”) дозволяє моделі підвищити точність оцінки, зменшуючи розбіжність із реальними ігровими даними.

ШІ правильно розпізнав основні тенденції: прості, частовживані слова (емірати, яхта, талант) отримали низькі оцінки складності, а абстрактні або наукові терміни (рефлексія, агностицизм, комплексність) — найвищі. Слова чумак і куранти показали більші відхилення, що пояснюється їхньою маловживаністю або історичністю: академічно ШІ може сприймати їх як складні, хоча, наприклад, слово чумак є добре впізнаваним у культурному контексті попри втрату практичного вжитку.

Окремим чинником варіативності є структура самого кросворду — перетини між словами, кількість відомих літер і порядок заповнення можуть зменшувати реальну складність у грі, навіть якщо слово за своєю природою є довгим або рідкісним. Тому певна розбіжність між аналітичними та мовними оцінками є закономірною і не свідчить про помилки моделі.

Загалом результати підтвердили, що оцінки складності, отримані за допомогою ШІ, добре корелюють із фактичними даними. У середньому відхилення не перевищує 10–15 %, що є прийнятним рівнем точності для автоматизованої попередньої перевірки. Такий підхід дозволяє використовувати штучний інтелект як ефективний інструмент для автоматичного визначення складності кросвордів перед їх публікацією, залишаючи остаточне коригування за аналітичними спостереженнями та редактором.

Висновки до розділу

У третьому розділі було розроблено та описано математичне й алгоритмічне забезпечення системи Crosswordium, яке визначає повний цикл створення кросворду — від формування сітки до перевірки готового контенту перед публікацією. Побудовано послідовний алгоритм генерації рівня, що включає етапи ініціалізації параметрів, створення або завантаження сітки, заповнення її словами, формування підказок і збереження результатів. Розроблені модулі (WordLook, QuestionLook, CrossLook) забезпечують розділення функціональності, що спрощує редагування, тестування та подальший розвиток системи.

Описано три методи добору слів:

- локальний пошук, який забезпечує швидкість і стабільність;
- генерація через ШІ, що дає змогу отримувати нові та тематично релевантні слова;
- комбінований підхід, який об’єднує переваги обох і став основним для Crosswordium.

Проведене порівняння моделей ШІ показало, що новіші системи (GPT-5-серії) забезпечують вищу точність і змістовність результатів при помірному збільшенні часу обробки.

Окрему увагу приділено перевірці якості згенерованих слів — етичності, мовній правильності та складності. ШІ продемонстрував високу здатність розпізнавати русизми, застарілі або неетичні терміни, а також надійну кореляцію оцінок складності з реальними даними гравців. У середньому відхилення між автоматичною оцінкою і аналітичними показниками не перевищує 10–15 %, що підтверджує достовірність і практичну придатність підходу.

Таким чином, розроблене алгоритмічне забезпечення забезпечує повну автоматизацію процесу створення та перевірки кросвордів із можливістю адаптації рівнів складності й контролю якості контенту.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Архітектура та загальний опис програмної системи

Середовище та версії. Редактор кросвордів реалізовано як інтегроване розширення в Unity 6000.0.58f2 (Editor tooling всередині проєкту). Для ШІ-функцій використовуються моделі сімейства GPT: GPT-4o-mini (актуальна базова/безкоштовна) та GPT-5 (актуальна платна), що підключаються через модуль GenAI.

Загальна ідея. Редактор працює всередині Unity Editor: має окрему сцену редагування з відображенням сітки (гарантує візуальну відповідність ігровому клієнту) і набір кастомних інспекторів/панелей (Unity Editor UI) для швидкого введення параметрів, роботи зі списками, зв’язків із ресурсами проєкту. Архітектура компонентна (модульна), що спрощує підтримку й розширення.

Причини інтеграції саме в Unity.

1. Зручний Editor UI (Custom Inspectors/EditorWindows) без окремого фреймворку.
2. Пряма робота з асетами проєкту (словники, зображення, конфіги, серіалізовані рівні).
3. Швидкі ітерації: зміни в коді редактора застосовуються миттєво, без зовнішніх збірок.

Основні елементи (скрипти та редакторські модулі)

- Manager: центральний компонент ініціалізації; зв’язує сцену, редакторські панелі й сервісні модулі (WordFind/GenAI/CrossLook), керує загальними параметрами сесії редагування.
- GenAI, GenAIPromptInfo (Додаток А): формування промптів; виклики до GPT-4o-mini / GPT-5; збереження шаблонів і параметрів.

- WordLook, WordLookEditor (Додаток Б): вибір/аналіз слова в поточній сітці, підбір альтернатив (локально/через ШІ), застосування до сітки.
- QuestionLook, QuestionLookEditor (Додаток В): створення/редагування підказок вручну або через ШІ; попередній перегляд і швидке внесення.
- WordFind (Додаток Г): робота з локальними словниками (кеш за мовою/типом, фільтри, підрахунок використань, відповідність шаблонам).
- CrossLook: локальне збереження/завантаження кросвордів (формати, версіонування, метадані).

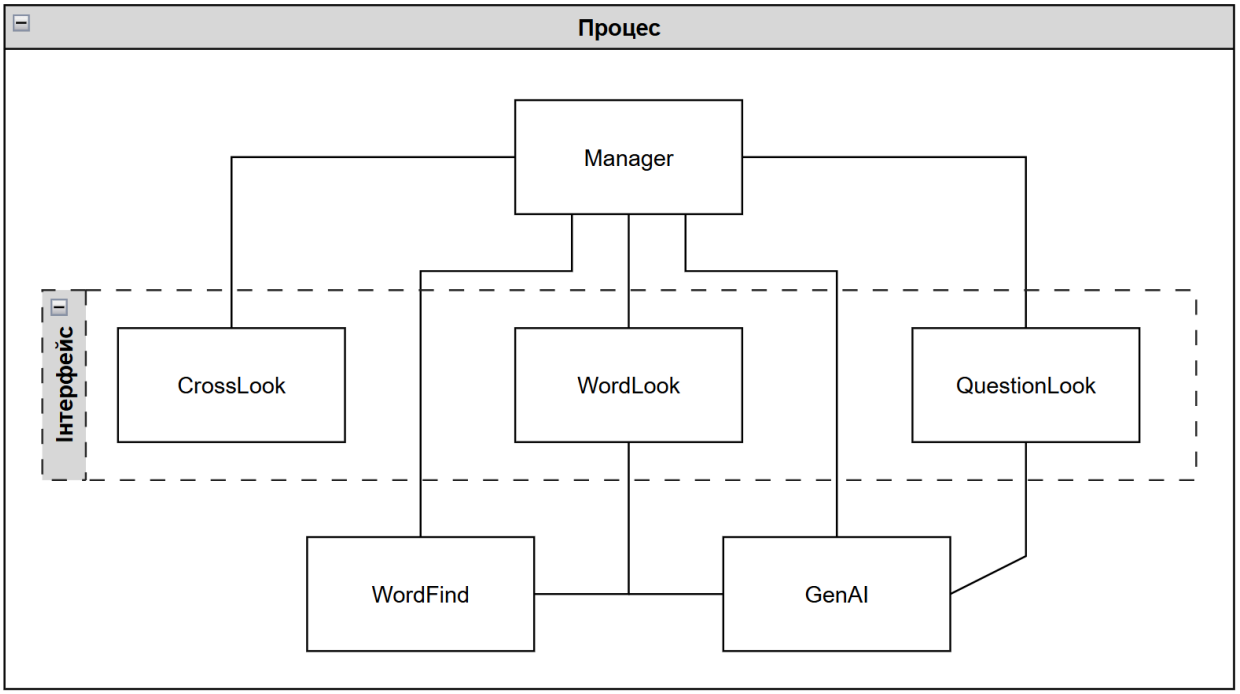

Рисунок 4.1 – *Модульна схема редактора Crosswordium*

Логіка взаємодії модулів (короткий робочий флоу)

1. Основна сцена з сіткою. Редактор бачить реальну сітку рівня, може редагувати блоки або виділити слово.
2. При виділенні слова активні дії в панелях:
    - WordLook — підбір/заміна слова: локально (WordFind) або через ШІ (GenAI).
    - QuestionLook — генерація/редагування підказок (локально/через ШІ).

3. WordFind і GenAI зазвичай попередньо налаштовані (мова, тип словника, параметри промптів) і працюють пасивно — викликаються за потреби з WordLook/QuestionLook.
4. Після підтвердження вибору слово/підказка застосовується до сітки, зміни зберігаються через CrossLook.

Частина дій і службових результатів (зокрема, відповіді ШІ, службові повідомлення про фільтри чи перевірки) можуть виводитись у Unity Console (стандартний інструмент логів). Це зручно для швидкого аудиту промптів/відповідей, трасування помилок і технічних деталей без засмічення основного інтерфейсу.

Редактор надає всі інструменти створення кросворду (пошук слів, генерація підказок, перевірки), але не обмежує сценарії, за якими автор може підготувати частину матеріалів поза редактором (добір тематичних списків, зображень тощо) і просто імпортувати готові елементи.

4.2 Сцена та сітка кросворда

Основним робочим середовищем редактора Crosswordium є сцена редагування, у якій відображається поточний кросворд у вигляді інтерактивної сітки. Саме тут користувач створює, редагує та перевіряє кросворд, додає нові слова й підказки та контролює структуру рівня перед збереженням. Сцена працює безпосередньо в Unity Editor, а візуальна частина побудована на основі елементів Unity UI (Canvas, Grid Layout, Text, Image), що дозволяє легко відтворювати зовнішній вигляд майбутньої ігрової сітки та гнучко змінювати її параметри.

Сітка дозволяє вільно взаємодіяти з клітинками: користувач може натиснути на будь-яку клітинку і, протягнувши курсор у вибраному напрямку, створити слово. Якщо нове слово не конфліктує з існуючими, воно автоматично додається до списку кросворду і відображається в полі. Подібним чином можна створювати поля під

запитання або зображення, що дає змогу швидко формувати повноцінні тематичні завдання.

Робота з сіткою побудована на інтуїтивній моделі: усі дії — створення, редагування або видалення — виконуються безпосередньо у сцені. При виборі слова воно підсвічується у сітці, а відповідний запис у списку слів виділяється автоматично. У цьому списку відображаються актуальні дані: саме слово, його координати, тип і зміст підказки (текстової або графічної). Користувач може швидко змінювати або вилучати слова, після чого редактор зберігає оновлений стан кросворду.

Такий підхід забезпечує зручну, наочну і швидку роботу над кросвордом: розробник бачить не лише структуру сітки, а й контент кожного елемента, що дозволяє підтримувати логічну цілісність і візуальну узгодженість рівня. Інтерактивна сцена відтворює кінцевий вигляд рівня у грі, тому всі зміни, зроблені в редакторі, одразу видно в реальному контексті.

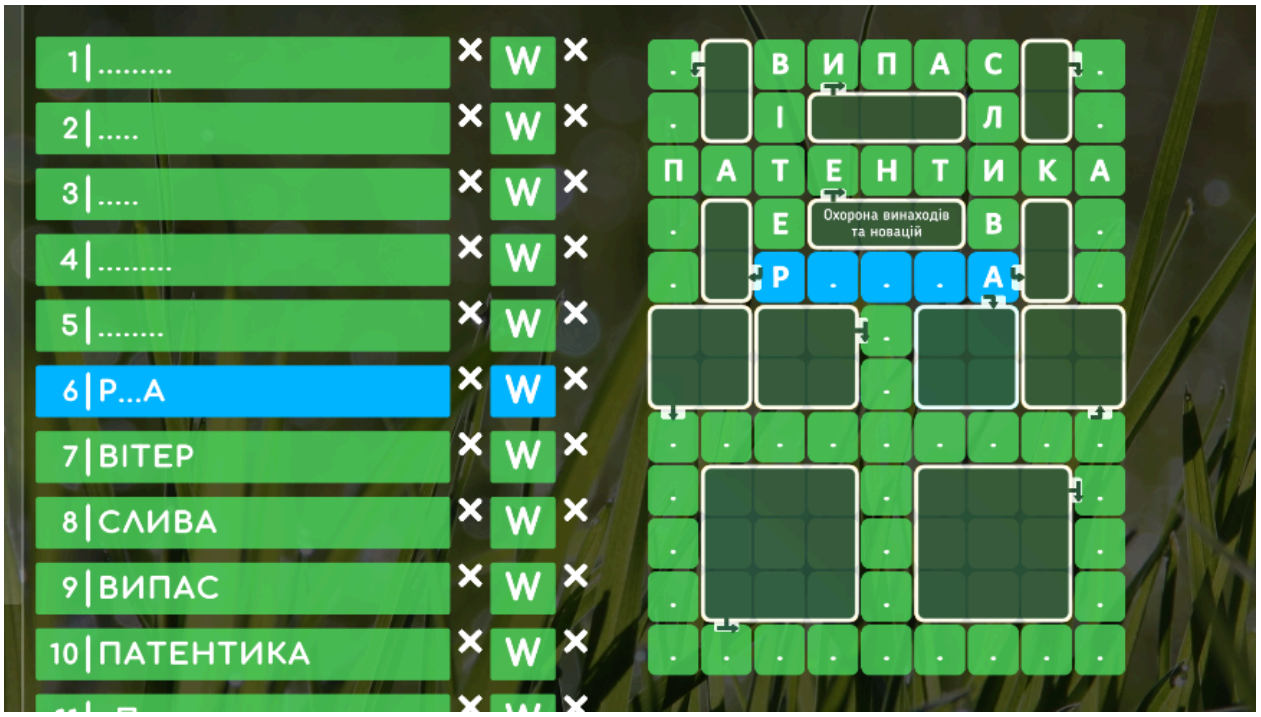

Рисунок 4.2 – Сітка кросворда зі списком слів для вибору й редагування.

Взаємодія між модулями:

1. Користувач у редакторі кросвордів обирає слово або поле, для якого потрібен автоматичний підбір чи генерація підказки.
2. WordLook (разом із WordLookEditor) здійснює запит — або до WordFind (для пошуку серед локальних слів), або через GenAI (для генерації через OpenAI).
3. Якщо потрібно сформулювати підказку — підключається QuestionLook, який також може використати як локальні дані, так і звернення до OpenAI (GenAI).
4. Усі ці модулі пов’язані з інтерфейсом користувача, дозволяючи інтерактивно переглядати й вибирати найкращі результати.

Важливо зазначити, що основна структура редактора кросвордів охоплює також такі компоненти, як відображення сітки кросворда, перегляд і редагування списку всіх слів кросворда, допоміжні інструменти для створення і тестування рівнів тощо. Проте саме підсистема автоматизованого підбору слів та генерації підказок, описана вище, є предметом дослідження і розробки в цій курсовій роботі.

Таке розділення дозволяє чітко виділити та описати інноваційну частину проекту, забезпечити її гнучкість і масштабованість, а також інтеграцію з існуючими інструментами для створення кросвордів.

4.3 Модуль підбору слів

Модуль WordLook є центральним інструментом у процесі створення кросвордів, оскільки саме він забезпечує пошук, відбір і застосування слів як із локальних словників, так і за допомогою штучного інтелекту (ШІ). Його інтерфейс інтегровано безпосередньо в Unity Editor як кастомне розширення, реалізоване на базі класів UnityEditor.Editor (повна реалізація наведена в додатку Б). Завдяки цьому WordLook працює в одному середовищі з основною сценою редагування, що забезпечує швидкий обмін даними між сіткою кросворда, підказками та іншими компонентами (WordFind, GenAI, QuestionLook, CrossLook). Повна структура вікна показана на рисунках 4.3 і 4.4.

Основна функція WordLook — добір слова за заданим шаблоном (наприклад, “..П.Н.”) із урахуванням мови, типу слів, частоти використання та стану поточного рівня.Редактор надає можливість виконувати пошук у двох режимах — локально (через WordFind) або через ШІ (через GenAI).Після отримання результатів користувач може одразу застосувати слово до сітки або залишити його у списку кандидатів для подальшої оцінки.Усі зміни синхронізуються зі сценою в реальному часі, що дає змогу миттєво спостерігати вплив вибраного слова на загальну структуру кросворду.

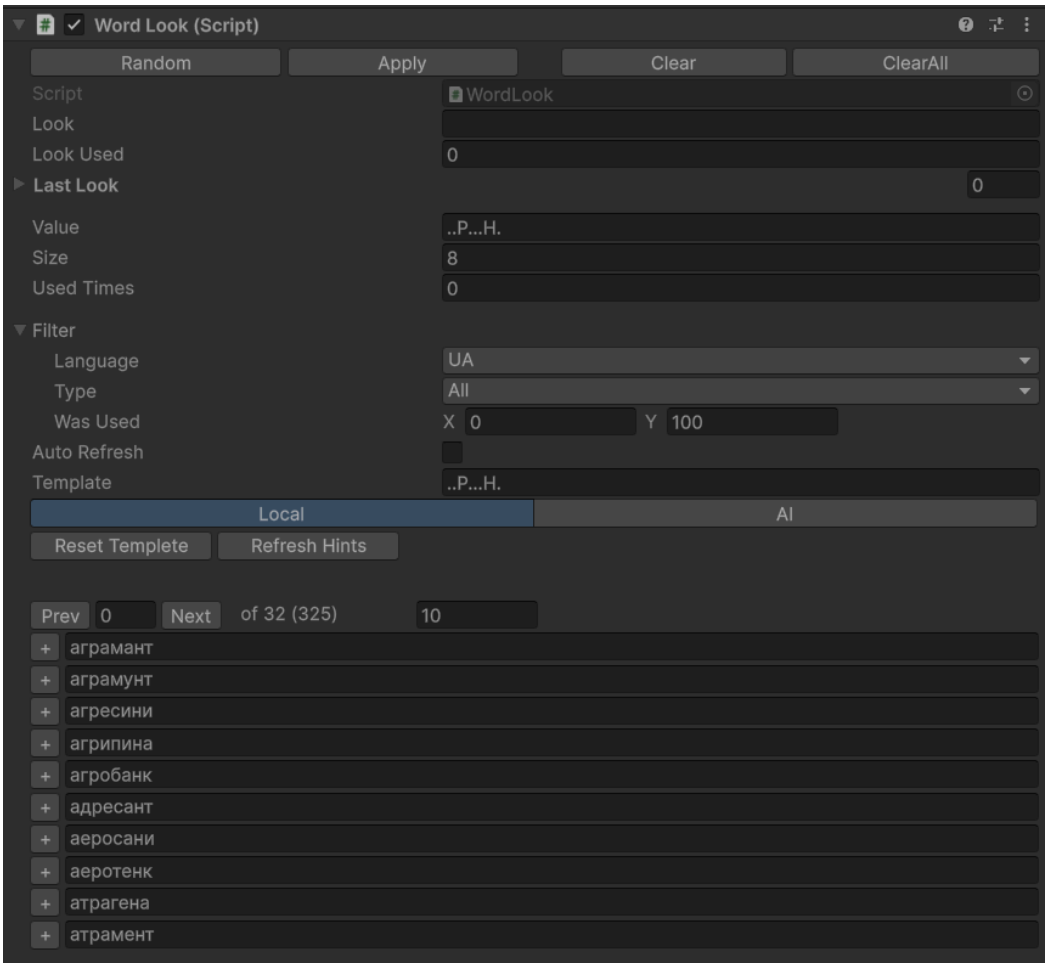

Рисунок 4.3 – *Інтерфейс модуля WordLook у режимі Local: пошук слів у локальному словнику*

Інтерфейс WordLook складається з кількох функціональних блоків:

1. У верхній частині інтерфейсу розташовано набір службових кнопок:
    - Random — вибір випадкового кандидата зі списку;

- Apply — застосування вибраного слова до сітки;
- Clear — очищення поточного значення;
- ClearAll — повне очищення літер у вибраному слові, з можливістю видалити також спільні з іншими словами символи.

2. Look — набір тимчасових кандидатів (словоформ), які розглядаються для поточного слова. Ці значення не зберігаються у файлі, але дозволяють порівнювати кілька варіантів перед прийняттям остаточного рішення.
3. Value — блок поточного слова, вибраного у сітці. Тут відображається саме слово, його довжина, кількість використань та інші службові параметри. Ця інформація динамічно оновлюється при зміні виділення у сітці або при повторному відкритті редактора.
4. Filter — набір фільтрів для формування списків. Користувач може обрати мову (Language), тип слів (Type), стан використання (Was Used), а також активувати автоматичне оновлення результатів (Auto Refresh). Параметри фільтрації дозволяють обмежити пошук до конкретних наборів слів — наприклад, лише українські іменники або лише слова, що ще не використовувалися у жодному кросворді.
5. Режим пошуку (Local / AI) — дві вкладки, які перемикають режим роботи. Нижче розташований список знайдених слів, де кожне можна одразу додати до сітки або залишити для подальшого розгляду.

Режим Local базується на роботі зі словниками, попередньо завантаженими у компоненті WordFind. Кожен словник організований за довжиною слова, що дозволяє швидко знаходити потрібні збіги. Пошук відбувається за принципом посимвольного порівняння з шаблоном — наприклад, “.П..А” означає, що друга літера “П”, а п’ята “А”. Такі запити виконуються практично миттєво, навіть на великих базах (понад 50 000 слів).

Після оновлення (Refresh Hints) результати відображаються внизу у вигляді сторінкового списку, з можливістю перегляду по 10–50 записів. Кожен елемент має кнопку “+” — натискання додає слово до поточної позиції у сітці. Режим Local

гарантує повну відповідність шаблону та максимальну швидкість, але не враховує контекст або тематику, що іноді призводить до появи маловживаних чи діалектних слів.

Режим AI дає змогу використовувати штучний інтелект для розширеного пошуку або генерації нових варіантів.

Він має дві основні функції:

- Select Hints — комбінований режим, у якому ШІ аналізує локальний список і відбирає найкращі за тематикою чи частотою;
- Generate Hints — повна генерація списку нових слів за шаблоном і вказаними параметрами.

Усі параметри (мова, шаблон, тип) автоматично передаються з редактора у GenAI, де формується текстовий промпт до моделі GPT-4o-mini або GPT-5. Результат ШІ очищується від сторонніх символів і перевіряється на відповідність шаблону перед відображенням у списку. Користувач може обрати будь-який із варіантів і застосувати його до сітки.

Перевагою цього режиму є можливість отримати нові або креативні слова, відсутні у локальних джерелах, а також формувати тематично узгоджені списки (наприклад, “морські терміни” або “музичні інструменти”).

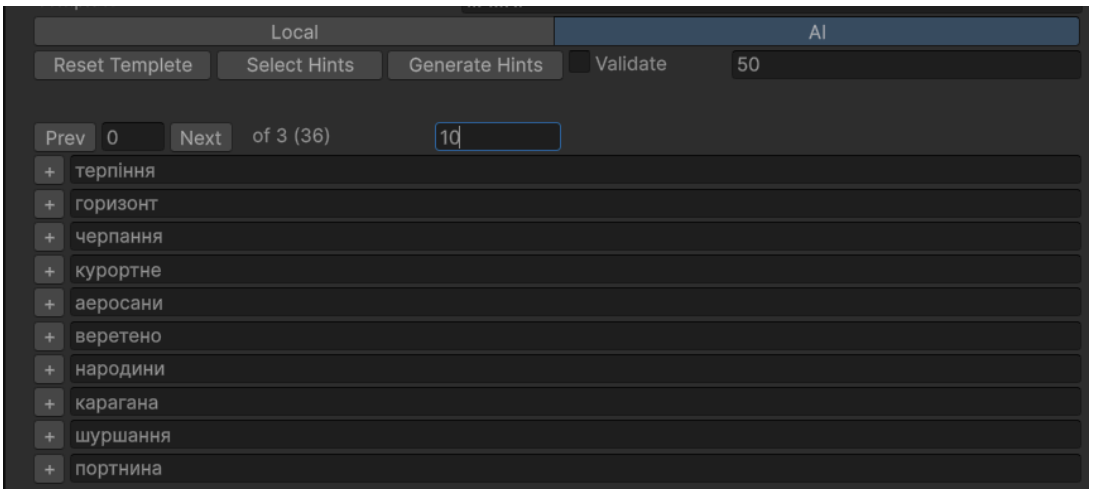

Рисунок 4.4 – *Режим AI у модулі WordLook: генерація та добір слів за допомогою ШІ*

Також передбачені кнопки Reset Template, Refresh Hints (для Local) і Validate (для AI), які оновлюють або перевіряють отримані результати. Усі службові повідомлення, включно з відповідями ШІ, дублюються у Unity Console, що дозволяє відслідковувати деталі виконання запитів і відлагоджувати роботу модуля.

Типовий сценарій використання

1. Редактор виділяє слово у сітці.
2. Вікно WordLook автоматично оновлює Value (текст, розмір, використання).
3. Користувач задає шаблон або активує фільтр.
4. Вибирає режим Local для швидкої перевірки словника або AI для розширеного пошуку.
5. Отримані результати переглядаються у списку; за потреби вибирається оптимальний варіант.
6. Після натискання Apply слово вставляється у сітку, а кросворд оновлюється.

Клас WordLookEditor розширює стандартний Unity Editor, використовуючи API OnInspectorGUI() для динамічного відображення елементів інтерфейсу. Основна логіка взаємодії з іншими модулями здійснюється через посилання на об’єкти сцени (SerializedObject, SerializedProperty) і сервіси (WordFind, GenAI). Завдяки цьому редактор зберігає стан між сесіями та може працювати з будь-яким активним кросвордом без перезапуску Unity.

Переваги реалізації

- Інтеграція безпосередньо в Unity Editor — робота без зовнішніх вікон або плагінів.
- Поєднання локального пошуку і ШІ — швидкість + тематична гнучкість.
- Миттєве оновлення результатів та синхронізація зі сценою.
- Можливість порівняння кандидатів перед вставленням.

- Гнучке керування через єдиний інтерфейс, з підтримкою кількох мов і частин мови.
- Логи у Unity Console, що забезпечують контроль і діагностику.

Таким чином, модуль WordLook є ключовим інструментом інтерактивного добору слів у системі Crosswordium. Він поєднує швидкість локального пошуку з адаптивністю штучного інтелекту, що забезпечує зручність, ефективність і розширюваність у створенні рівнів кросвордів. Повний код скрипта WordFind, а також деталі реалізації основних методів підбору та фільтрації, наведені у додатку Г цієї курсової роботи.

4.4 Модуль підбору підказок

Модуль QuestionLook відповідає за створення, редагування та автоматичний підбір підказок (запитань) до слів у кросворді. Він є частиною інтегрованого редакторського середовища Crosswordium та реалізований як окреме вікно на основі розширення класів UnityEditor.Editor. Інтерфейс і логіка модуля розроблені так, щоб забезпечити однакову зручність як для ручного введення текстів, так і для генерації варіантів за допомогою штучного інтелекту.

QuestionLook автоматично синхронізується з поточним словом, вибраним у сітці. Редактор може створити нове запитання вручну або отримати варіанти від ШІ.

Основні поля інтерфейсу поділені на декілька груп:

- Question Info → Text — основне текстове поле, де вводиться або редагується формулювання підказки.
- Is Image / Image Name — перемикач і поле для створення візуальних підказок (зображень). Якщо позначено Is Image, підказка у грі відображатиметься як картинка.
- CC License поля (Author, License, Changes, Source) — блок для введення даних Creative Commons у випадках, коли використовується зображення

з відкритою ліцензією. Це дозволяє дотримуватись правил авторського права та фіксувати джерело.

Після редагування натискання кнопки Refresh Visual оновлює відображення підказок у сцені, щоб одразу побачити результат у контексті кросворду.

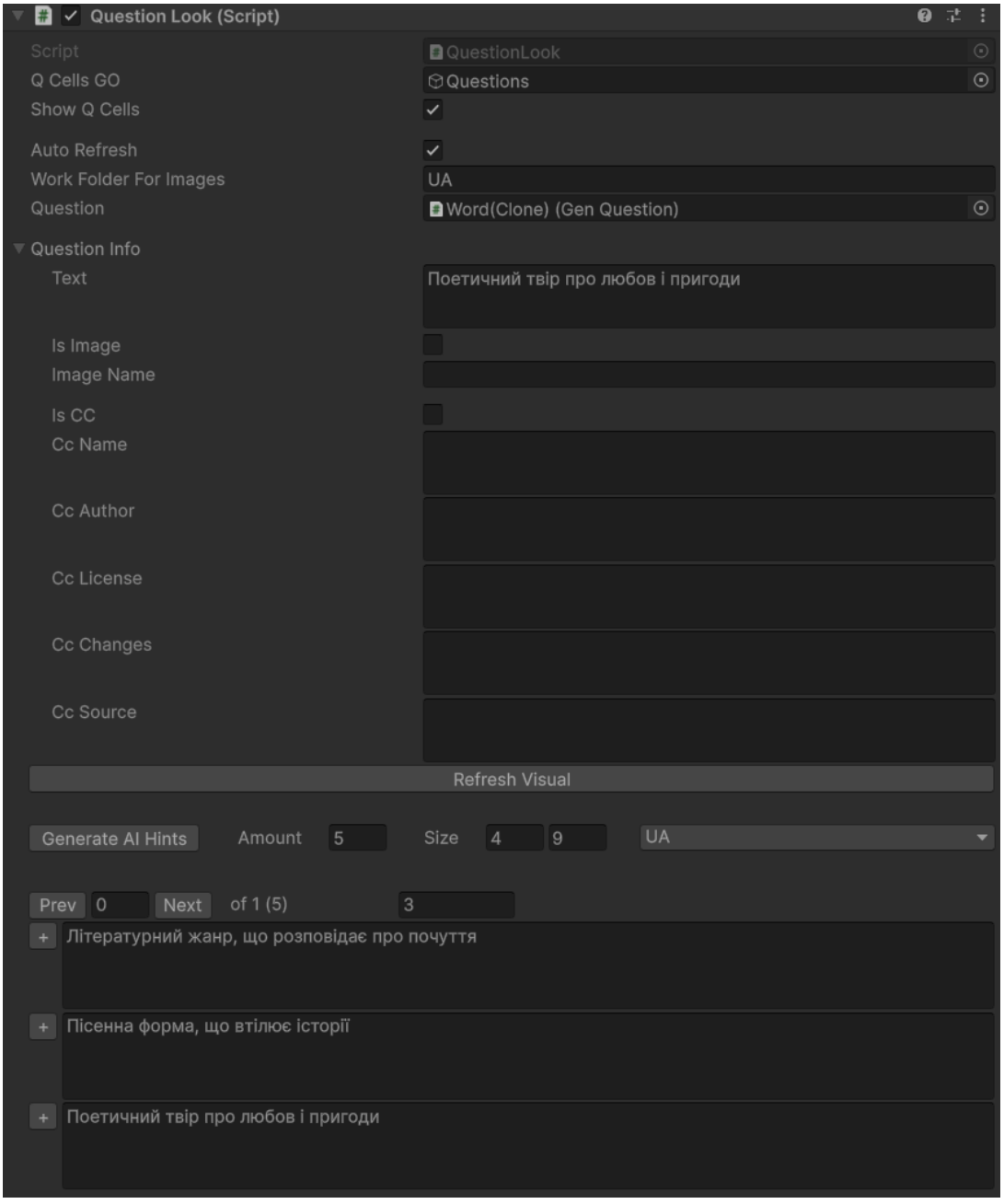

Рисунок 4.5 – *Інтерфейс QuestionLook: створення та генерація підказок через ШІ*

У нижній частині інтерфейсу розташований блок Generate AI Hints, який дозволяє отримати варіанти підказок із використанням мовної моделі GPT (через

модуль GenAI). Редактор вказує кількість бажаних варіантів (Amount) та орієнтовну довжину речення (Size), після чого система надсилає запит на генерацію.

ШІ повертає список можливих формулювань, які відображаються у таблиці нижче. Кожен варіант можна одразу застосувати до активного слова натисканням кнопки “+”. Зазвичай ШІ генерує кілька коротких і нейтральних підказок, що відрізняються за стилем.

Типовий сценарій роботи

1. Користувач у сцені виділяє слово.
2. Модуль QuestionLook автоматично підтягує поточне слово в поле Question.
3. Редактор може:
    - вручну ввести текст підказки у полі Text,
    - або натиснути Generate AI Hints для отримання кількох варіантів.
4. Обраний варіант застосовується натисканням “+”.
5. Якщо підказка є зображенням, за потреби заповнюються відповідні CC-поля (автор, джерело, ліцензія).

Особливості реалізації та переваги

- Єдина структура для текстових і графічних підказок.
- Автоматична синхронізація з активним словом у сцені.
- Можливість роботи в офлайн-режимі (ручне введення) або з ШІ (онлайн).
- Збереження метаданих Creative Commons, що полегшує роботу з відкритими зображеннями.
- Генерація кількох варіантів дає змогу швидко підібрати найбільш точне чи стилістично доречне запитання.

Модуль QuestionLook забезпечує повний цикл створення підказок — від ручного написання до автоматичної генерації. Він поєднує простоту редакторського

інтерфейсу з потужністю мовних моделей, дозволяючи швидко отримувати якісні формулювання, адаптовані до контексту слова. Таке рішення значно скорочує час підготовки контенту та підтримує цілісність кросворду за рахунок уніфікованого підходу до всіх типів підказок.

Висновки до розділу

У цьому розділі представлено програмну реалізацію системи Crosswordium — інтегрованого редактора кросвордів, створеного у середовищі Unity 6000.0.58f2 з використанням Unity Editor API та моделей GPT-4o-mini / GPT-5 для автоматизованого підбору контенту.

Редактор має модульну структуру, що поєднує сцену з сіткою кросворду та кілька спеціалізованих компонентів:

- WordLook — для добору слів (локально або через ШІ),
- QuestionLook — для створення текстових і візуальних підказок,
- WordFind — для пошуку у локальних словниках,
- GenAI — для взаємодії з мовними моделями,
- CrossLook — для збереження й завантаження рівнів.

Комбінований підхід до генерації слів і підказок дозволив поєднати швидкість локального пошуку з креативністю ШІ, а також забезпечив стабільну роботу редактора без виходу за межі Unity. Інтерфейси на основі кастомних редакторів забезпечують зручність, миттєве оновлення даних і можливість розширення функціоналу. Розроблена система довела свою ефективність і практичність: вона забезпечує швидку генерацію, перевірку та збереження кросвордів і може слугувати базою для подальшого розвитку інтелектуальних ігрових інструментів.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Ідея стартапу

Стартап-проєкт Crosswordium — це мобільна гра нового покоління, що поєднує класичну форму кросвордів із сучасними технологіями штучного інтелекту. Основна ідея полягає у використанні AI-інструментів для створення та оновлення контенту, що дозволяє суттєво скоротити час розробки рівнів і підвищити якість ігрового досвіду.

На відміну від традиційних словесних ігор, у Crosswordium реалізовано власний редактор рівнів, який може автоматично добирати слова, формувати підказки та оцінювати складність із використанням мовних моделей GPT. Редактор наразі використовується розробником для генерації нових рівнів, але в перспективі може бути адаптований до спрощеної версії для самих гравців.

Інноваційність стартапу полягає у поєднанні:

- класичної мобільної гри зі стабільною механікою;
- ШІ-редактора, що автоматизує створення кросвордів;
- потенційної UGC-екосистеми (User Generated Content), у якій користувачі зможуть створювати власні кросворди безпосередньо в грі, користуючись спрощеним ШІ-інтерфейсом.

Це розширює модель гри від звичайного контенту до динамічної платформи, де оновлення рівнів і залучення користувачів відбуватиметься природно, без потреби постійного ручного оновлення.

5.2 Технологічна реалізація

Гра Crosswordium розроблена у середовищі Unity 6000.0.58f2, що забезпечує кросплатформність, високу продуктивність і підтримку Android- та iOS-платформ.

Для генерації текстових даних використовується OpenAI API із моделями GPT-4o-mini (оперативна генерація) та GPT-5 (аналітична обробка).

Технологічна інфраструктура проекту також включає Firebase, який використовується як серверна платформа для:

- зберігання готових кросвордів і словників;
- обробки запитів користувачів до ШІ (через проміжний API-сервіс);
- керування оновленнями контенту та поширення нових рівнів;
- автентифікації користувачів і ведення статистики активності.

Firebase надає повний набір інструментів для хмарного зберігання, баз даних у реальному часі та інтеграції з аналітикою, що робить його оптимальною основою для масштабування Crosswordium у майбутньому. Така архітектура забезпечує гнучкість — при потребі можна змінювати модель, тип запиту або систему постобробки без оновлення клієнтської частини гри.

Сучасна архітектура гри вже містить усі необхідні технічні основи для розширення функціональності. На базі існуючих AI-компонентів передбачено можливість реалізації внутрішньоігрового редактора, який дозволятиме користувачам створювати власні кросворди безпосередньо у Crosswordium. Це може супроводжуватися серверною обробкою AI-запитів, перевіркою контенту на відповідність мовним і етичним нормам та збереженням користувацьких рівнів у хмарному сховищі. Таким чином, наявна інфраструктура створює реалістичну технологічну базу для подальшого розвитку гри у напрямі динамічної AI-екосистеми.

5.3 Аналіз ринкових можливостей

Ринок словесних ігор стабільно зростає: за даними AppMagic та SensorTower, у 2024 році мобільний сегмент “Word & Puzzle Games” перевищив $1,2 млрд із темпом зростання 8–10 % щороку. Популярність таких ігор пояснюється їхньою універсальністю, освітнім ефектом та можливістю коротких ігрових сесій.

Однак більшість сучасних рішень (Wordscapes, CodyCross, Crossword Puzzle Free тощо) мають статичні рівні, створені вручну, без можливості автоматичного оновлення або залучення користувачів до створення контенту.

Crosswordium вирізнятиметься завдяки:

- AI-редактору, що дозволяє постійно створювати нові рівні без зовнішніх ресурсів;
- підтримці української мови, якої практично немає серед конкурентів;
- потенціалу спільнотного розвитку, де гравці зможуть створювати та ділитися власними кросвордами.

Потенційні користувачі:

- гравці словесних ігор (широка масова аудиторія 18–45 років);
- викладачі, які використовують кросворди як навчальний інструмент;
- контент-креатори, які працюють з AI-моделями для лінгвістичних ігор.

5.4 Ринкова стратегія

На початковому етапі Crosswordium позиціонується як класична мобільна гра з AI-генерацією рівнів у фоновому режимі. Основний маркетинговий акцент — на якісному контенті, що постійно оновлюється, та на україномовній підтримці.

Стратегія розвитку передбачає три послідовні етапи:

1. Поточний етап: активна підтримка гри та регулярне додавання нових рівнів, створених за допомогою оновленого редактора;
2. Середньострокова перспектива: розроблення внутрішнього редактора рівнів для гравців, що дозволить їм створювати власні кросворди;
3. Довгострокова мета: формування UGC-екосистеми, де користувачі зможуть ділитися своїми кросвордами, оцінювати контент інших і змагатися у спільноті.

Модель монетизації:

- безкоштовна базова версія з рекламою;
- внутрішні покупки (додатковий набір рівнів, підказки);
- можливе введення передплати на розширені можливості редактора (у майбутньому UGC-режимі).

5.5 Висновки до розділу

Гра Crosswordium є прикладом практично реалізованого стартап-проєкту, який поєднує класичну ігрову модель та інноваційну AI-підсистему створення контенту. Редактор, що вже використовується під час розробки рівнів, забезпечує унікальну швидкість оновлення гри та високий рівень якості словникового матеріалу.

Подальший розвиток передбачає розширення функціоналу до вбудованого користувацького редактора, що дозволить гравцям створювати власні кросворди у грі. Це відкриває перспективу UGC-екосистеми, у якій AI стане посередником між гравцем-творцем і спільнотою, підтримуючи баланс між креативністю та якістю контенту. Таким чином, Crosswordium має не лише готовий продукт, але й чітку інноваційну траєкторію розвитку, що поєднує ігрову індустрію та сучасні інструменти штучного інтелекту.

ВИСНОВКИ

У дипломній роботі було досліджено, спроєктовано та реалізовано систему автоматизованого створення кросвордів із використанням технологій штучного інтелекту на прикладі гри Crosswordium.

Метою роботи було створення інструменту, який поєднує класичні алгоритмічні підходи з можливостями сучасних мовних моделей для генерації, перевірки та оцінки контенту.

У ході виконання роботи:

- проведено аналіз проблемної області та традиційних методів формування кросвордів;
- обґрунтовано доцільність використання AI-моделей для добору слів і підказок, а також для оцінки складності й етичності контенту;
- розроблено алгоритмічне та програмне забезпечення системи Crosswordium Editor, що включає модулі WordLook, QuestionLook, WordFind, GenAI та CrossLook;
- реалізовано інтеграцію з OpenAI API для автоматичного створення й аналізу слів, що дало змогу поєднати швидкість локального пошуку з креативністю ШІ;
- здійснено практичну оцінку точності генерації різними моделями (GPT-4o-mini, GPT-5 тощо) та досліджено ефективність AI-оцінки складності українських слів на основі реальної ігрової аналітики;
- створено повноцінний редактор рівнів у середовищі Unity, який забезпечує зручну роботу з сіткою кросворду, підказками й AI-запитами.

Розроблена система довела, що поєднання локальних словників і мовних моделей ШІ є ефективним підходом до створення ігрового контенту:
локальні методи гарантують коректність і швидкодію, тоді як ШІ забезпечує розширення словникового запасу, тематичність і стилістичну гнучкість. Оцінювання якості показало, що сучасні мовні моделі здатні коректно визначати складність і

придатність слів у ≈85 % випадків, що відкриває перспективу для подальшого використання таких систем у геймдеві та освіті.

У межах проєкту Crosswordium створено не лише гру, але й AI-редактор, який спрощує розробку рівнів і відкриває можливості для подальшого розвитку продукту як стартапу. Інфраструктура на базі Firebase і OpenAI API забезпечує технічну основу для масштабування контенту, а в перспективі — для реалізації внутрішньоігрового редактора та системи користувацьких кросвордів (UGC-екосистеми).

Отже, поставлену мету роботи досягнуто. Система Crosswordium демонструє практичну інтеграцію штучного інтелекту в процес розробки ігор, що дозволяє автоматизувати рутинні завдання, покращити якість контенту та створити передумови для появи нових форматів гейміфікованого навчання. Отримані результати підтверджують актуальність теми та перспективність подальших досліджень у напрямі AI-асистованої генерації контенту в ігровій індустрії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. OpenAI. API Documentation. – 2024. – https://platform.openai.com/docs
2. Levy, R. Crossword Construction Algorithms. – Springer, 2018.
3. Vaswani, A., Shazeer, N., Parmar, N., et al. Attention Is All You Need. // Advances in Neural Information Processing Systems. – 2017. – Vol. 30.
4. Gregory, J. Game Engine Architecture. – CRC Press, 2019.
5. Unity Technologies. Unity Scripting API. – 2024. – https://docs.unity3d.com/ScriptReference/
6. Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach. – Pearson, 2021.
7. Liu, J., et al. Evaluation of Large Language Models. // Proceedings of the Association for Computational Linguistics (ACL). – 2023.
8. Андрущенко, Т. В., Костюк, Г. В. Словникові ресурси для інтелектуальних лінгвістичних систем: сучасний стан та перспективи розвитку. // Комп’ютерна лінгвістика та інтелектуальні технології. – 2022. – № 25. – С. 41–55.
9. Васильченко, Д. В. Інтеграція штучного інтелекту у лінгвістичні ігри: тенденції та виклики. // Інформаційні технології і засоби навчання. – 2021. – № 4(88). – С. 101–116.
10. Ермаков, В. О. Застосування нейронних мереж для генерації лінгвістичних задач у навчальних програмах. // Науковий вісник Черкаського університету. Серія: Інформатика. – 2022. – № 3. – С. 49–57.
11. Firebase Documentation. Google Cloud Platform. – 2024. – https://firebase.google.com/docs

ДОДАТКИ

ДОДАТОК А

Скрипти GenAI та GenAIPromptInfo — модулі для роботи з OpenAI API

```
using Newtonsoft.Json;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using UnityEngine;

namespace Generator
{
    public class GenAI : MonoBehaviour
    {
        [SerializeField] private string apiKey;
        [SerializeField] private string model;

        [SerializeField] private GenAIPromptInfo promptsEN;
        [SerializeField] private GenAIPromptInfo promptsUA;

        [SerializeField] private bool debugResults;

        private string apiUrl = "https://api.openai.com/v1/chat/completions";


           private GenAIPromptInfo GetPromptInfo(Language language) => language
switch
        {
            Language.EN => promptsEN,
            Language.UA => promptsUA,
            _ => null
        };

        private string BetterPattern(string basePattern)
        {
            basePattern = basePattern.Replace('.', '?');
            var betterPattern = string.Empty;
            for (int i = 0; i < basePattern.Length; i++)
            {
                betterPattern += $"[{basePattern[i]}]";
            }
            return betterPattern;
        }

          public async Task<List<string>> GetWords(Language language, int amount,
string pattern)
        {
            var prompts = GetPromptInfo(language);
            if (prompts == null)
            {
```

```
                    Debug.LogError($"Unsuported language for prompts! {language}");
                    return new();
                }

                    var response = await SendPrompt(string.Format(prompts.getWord,
amount, pattern.Length, BetterPattern(pattern)));

                if (debugResults)
                    Debug.Log("AI result:\n" + response);

                return GetWordsFromRespond(response, language, pattern);
            }

               public async Task<List<string>> SelectWords(Language language, string
pattern, List<string> variants)
            {
                var prompts = GetPromptInfo(language);
                if (prompts == null)
                {
                    Debug.LogError($"Unsuported language for prompts! {language}");
                    return new();
                }

                   var response = await SendPrompt(string.Format(prompts.selectWord,
BetterPattern(pattern), string.Join(", ", variants)));

                if (debugResults)
                    Debug.Log("AI result:\n" + response);

                return GetWordsFromRespond(response, language, pattern);
            }

             public async Task<List<string>> GetHints(Language language, int amount,
int sizeMin, int sizeMax, string word)
            {
                var prompts = GetPromptInfo(language);
                if (prompts == null)
                {
                    Debug.LogError($"Unsuported language for prompts! {language}");
                    return new();
                }

                    var response = await SendPrompt(string.Format(prompts.getHint,
amount, sizeMin, sizeMax, word));

                if (debugResults)
                    Debug.Log("AI result:\n" + response);

                return GetTextsFromRespond(response, language);
            }

             private async Task<string> SendPrompt(string prompt, float temperature =
0.6f)
            {
                using (var client = new HttpClient())
                {
                            client.DefaultRequestHeaders.Add("Authorization", $"Bearer
{apiKey}");

                    var requestBody = new
                    {
                        model = model,
                        messages = new[]
                        {
```

```
                        new { role = "user", content = prompt }
                    },
                        temperature = temperature,
                        max_tokens = 256
                    };

                                                                var content = new
StringContent(JsonConvert.SerializeObject(requestBody), Encoding.UTF8,
"application/json");

                    if (debugResults)
                        Debug.Log(prompt);

                    var response = await client.PostAsync(apiUrl, content);
                    var responseString = await response.Content.ReadAsStringAsync();

                    if (debugResults)
                        Debug.Log(responseString);

                    // Дістанемо текст з JSON
                    dynamic json = JsonConvert.DeserializeObject(responseString);
                    string aiText = json.choices[0].message.content;
                    return aiText.Trim();
                }
            }

                 private List<string> GetWordsFromRespond(string response, Language
language, string pattern)
            {
                var lines = response.Split('\n');
                var words = new List<string>();
                                                        var wordRegex = new
Regex(@"[\d\.\-\)]*\s*([a-яA-ЯіїєґІЇЄҐa-zA-Z'\-]+)", RegexOptions.IgnoreCase);

                foreach (var line in lines)
                {
                    var match = wordRegex.Match(line.Trim());
                    if (match.Success)
                    {
                        string wordRaw = match.Groups[1].Value.Trim();

                        // Очищаємо слово: лишаємо тільки букви відповідної мови
                        string cleanWord = "";
                        if (language == Language.UA)
                             cleanWord = Regex.Replace(wordRaw, @"[^a-яA-ЯіїєґІЇЄҐ]",
"");
                        else if (language == Language.EN)
                            cleanWord = Regex.Replace(wordRaw, @"[^a-zA-Z]", "");
                        else
                                                    cleanWord = Regex.Replace(wordRaw,
@"[^a-zA-Za-яA-ЯіїєґІЇЄҐ]", "");

                        // Довжина слова має співпадати з патерном
                        if (cleanWord.Length == pattern.Length)
                            words.Add(cleanWord);
                    }
                }
                string regexPattern = "^" + pattern + "$";

                var result = new List<string>();
                foreach (var word in words)
                {
                      if (Regex.IsMatch(word, regexPattern, RegexOptions.IgnoreCase |
RegexOptions.CultureInvariant))
```

```
                        result.Add(word);
                }

                if (debugResults)
                    Debug.Log("AI format:\n" + string.Join('\n', result));

                return result;
            }

             private List<string> GetTextsFromRespond(string response, Language
language)
            {
                var result = new List<string>();
                var lines = response.Split('\n');

                    // Основний патерн: може бути цифра+крапка, дефіс, кружечок або
просто текст
                var regex = new Regex(@"^\s*(\d+\.\s*|\-\s*|•\s*)?(?<text>.+)$");

                foreach (var line in lines)
                {
                    string trimmed = line.Trim();
                    // Пропускаємо явно порожні рядки або службовий текст
                                        if (string.IsNullOrEmpty(trimmed) ||
trimmed.ToLower().StartsWith("ось") || trimmed.ToLower().Contains("підказ"))
                        continue;

                    var match = regex.Match(trimmed);
                    if (match.Success)
                    {
                        string text = match.Groups["text"].Value.Trim().TrimEnd('.');
                        // Ігноруємо дуже короткі рядки, які схожі на службові
                        if (text.Length > 2)
                            result.Add(text);
                    }
                }
                return result;
            }
        }
    }

    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;


    namespace Generator
    {
        [CreateAssetMenu(menuName = "AI/Prompt Template")]
        public class GenAIPromptInfo : ScriptableObject
        {
            [Multiline(8)]
            public string getWord;
            [Multiline(8)]
            public string selectWord;
            [Multiline(8)]
            public string getHint;
        }
    }
```

ДОДАТОК Б

Скрипти WordLook та WordLookEditor — логіка й інтерфейс підбору слова

```
using System.Collections;
using System.Collections.Generic;
using System.Threading.Tasks;
using UnityEngine;

namespace Generator
{
    public class WordLook : MonoBehaviour
    {
        private GenManager manager;
        private WordFind wordFind;
        private GenAI genAI;
        bool inited;

        public string look;
        public int lookUsed;
        public List<string> lastLook = new List<string>();

        [Space]
        public string value;
        public int size;
        public int usedTimes;

        [Space]
        public WordFilter filter;
        public bool autoRefresh;
        public string template;

        private WordFilter lastFilter;
        private string lastTemplate;

        [HideInInspector]
        public int FindCount;
        [HideInInspector]
        public List<string> FindResult = new List<string>();

        private GenWord Word;

        private bool selecting;

        public void Init(GenManager manager, WordFind find, GenAI genAI)
        {
            this.manager = manager;
            this.wordFind = find;
            this.genAI = genAI;

            inited = true;
        }

        public void SetWord(GenWord word)
        {
            Word = word;
            if (!word)
            {
                Clear();
                return;
            }
```

```
                size = word.CellsPos.Length;
                value = "";
                foreach (var pos in word.CellsPos)
                    value += manager.Letters[pos.x, pos.y];
                usedTimes = wordFind.UsedTimes(value, filter);

                template = value;
                lastTemplate = value;
                RefreshResult();
            }

            private async void RefreshResult()
            {
                if (selecting)
                    return;
                selecting = true;
                        //Task<List<string>> task = new Task<List<string>>(() =>
wordFind.Select(template, filter));
                await Task.Run(() =>
                {
                    var result = wordFind.Select(template, filter);
                    FindResult.Clear();
                    FindResult.AddRange(result);
                });
                lastTemplate = template;
                lastFilter = filter;
                FindCount = FindResult.Count;
                selecting = false;
            }

            private void Clear()
            {
                value = "";
                size = 0;
                template = "";
                usedTimes = 0;
            }

            public void ResetTemplate()
            {
                template = value;
                lastTemplate = value;
                RefreshResult();
            }

            public void TryRefreshResult()
            {
                if (template == lastTemplate && filter.Equals(lastFilter))
                    return;

                if(!Word)
                {
                    RefreshResult();
                    return;
                }

                if (template.Length != value.Length)
                {
                    template = lastTemplate;
                    return;
                }
                for (int i = 0; i < size; i++)
                    if (value[i] != GenData.DOT && value[i] != template[i])
```

```
                {
                    template = lastTemplate;
                    return;
                }

            RefreshResult();
        }

        private void OnValidate()
        {
            if (!inited || !autoRefresh)
                return;
            lookUsed = wordFind.UsedTimes(look, filter);
            TryRefreshResult();
        }

        public void LookRandom()
        {
            if(FindResult.Count == 0)
                return;

            var r = Random.Range(0, FindResult.Count);
            look = FindResult[r];
            lookUsed = wordFind.UsedTimes(look, filter);
            lastLook.Add(look);
            if (lastLook.Count > 5)
                lastLook.RemoveAt(0);
            //lookUsed = wordFind.UsedTimes(look, filter);
        }

        public void LookSelect(string s)
        {
            look = s;
            lookUsed = wordFind.UsedTimes(look, filter);
            lastLook.Add(look);
            if (lastLook.Count > 5)
                lastLook.RemoveAt(0);
        }

        public void LookApply()
        {
            if (!Word)
                return;

            if (look.Length != size)
                return;

            for (int i = 0; i < size; i++)
                    if (value[i] != GenData.DOT && char.ToUpper(value[i]) !=
char.ToUpper(look[i]))
                {
                    Debug.Log("[WordLook] can't Apply with pattern!");
                    return;
                }

            Word.SetValue(look.ToUpper());
            value = look;
            usedTimes = wordFind.UsedTimes(value, filter);
        }

        public void LookClear()
        {
            if (!Word)
                return;
```

```
            manager.WordClearValue(Word.CellsPos);
            SetWord(Word);
        }

        public void LookClearAll()
        {
            if (!Word)
                return;

            manager.WordClearAllValue(Word.CellsPos);
            SetWord(Word);
        }

        #region AI

        [HideInInspector]
        public List<string> FindAIResult = new List<string>();

        public void TryRefreshAIResult(int amount = 10)
        {
            if (!Word)
                return;

            if (string.IsNullOrEmpty(template))
                return;

            RefreshAIResult(amount);
        }

        private async void RefreshAIResult(int amount = 10)
        {
                    FindAIResult = await genAI.GetWords(filter.language, amount,
template);
        }

        public void TrySelectAIResult(int amount = 10)
        {
            if (!Word)
                return;

            if (string.IsNullOrEmpty(template))
                return;

            SelectAIResult(amount);
        }

        private async void SelectAIResult(int amount = 10)
        {
            RefreshResult();
            while (selecting)
            {
                await Task.Delay(10);
            }

            if (FindResult.Count == 0)
            {
                Debug.LogError("No Local Words!");
                return;
            }

            var list = new List<string>(FindResult);
            var selected = new List<string>();
            for (int i = 0; i < amount; i++)
```

```
                {
                    var r = Random.Range(0, list.Count);
                    selected.Add(list[r]);
                    list.RemoveAt(r);

                    if (list.Count == 0)
                        break;
                }

                if (selected.Count == 0)
                {
                    Debug.LogError("No Selected Words From Local!");
                    return;
                }
                else if (selected.Count == 1)
                {
                    FindAIResult = selected;
                    return;
                }

                   FindAIResult = await genAI.SelectWords(filter.language, template,
selected);
            }

            #endregion
        }
    }

    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using UnityEditor;

    namespace Generator
    {
        [CustomEditor(typeof(WordLook), true)]
        public class WordLookEditor : Editor
        {
            public enum LookType
            {
                Local,
                AI,
            }

            private LookType lookType;
            private int resultPage;
            private int PC = 20;
            private int aiHints = 10;

            public override void OnInspectorGUI()
            {
                var t = (WordLook)target;

                GUILayout.BeginHorizontal();

                if (GUILayout.Button("Random"))
                {
                    t.LookRandom();
                }

                if (GUILayout.Button("Apply"))
                {
                    t.LookApply();
                }
```

```
                GUILayout.Space(25);

                if (GUILayout.Button("Clear"))
                {
                    t.LookClear();
                }

                if (GUILayout.Button("ClearAll"))
                {
                    t.LookClearAll();
                }

                GUILayout.EndHorizontal();

                base.OnInspectorGUI();

                                lookType  =  (LookType)GUILayout.Toolbar((int)lookType,
System.Enum.GetNames(typeof(LookType)));

                if (lookType == LookType.Local)
                {
                    DrawLocalSearch();
                }
                else if (lookType == LookType.AI)
                {
                    DrawAISearch();
                }

                void DrawLocalSearch()
                {
                    GUILayout.BeginHorizontal();
                    if (GUILayout.Button("Reset Templete", GUILayout.MaxWidth(120)))
                    {
                        t.ResetTemplate();
                    }
                    if (GUILayout.Button("Refresh Hints", GUILayout.MaxWidth(120)))
                    {
                        t.TryRefreshResult();
                    }
                    GUILayout.EndHorizontal();

                    if (t.FindResult.Count > 0)
                    {
                        GUILayout.Space(25);
                        DrawSearchList(t.FindResult);
                    }
                }

                void DrawAISearch()
                {
                    GUILayout.BeginHorizontal();
                    if (GUILayout.Button("Reset Templete", GUILayout.MaxWidth(120)))
                    {
                        t.ResetTemplate();
                    }
                    if (GUILayout.Button("Select Hints", GUILayout.MaxWidth(120)))
                    {
                        t.TrySelectAIResult(aiHints);
                    }
                    if (GUILayout.Button("Generate Hints", GUILayout.MaxWidth(120)))
                    {
                        t.TryRefreshAIResult(aiHints);
                    }
```

```
                    aiHints = Mathf.Clamp(EditorGUILayout.IntField(aiHints), 1, 40);
                    GUILayout.EndHorizontal();

                    if (t.FindAIResult.Count > 0)
                    {
                        GUILayout.Space(25);
                        DrawSearchList(t.FindAIResult);
                    }
                }

                void DrawSearchList(List<string> result)
                {
                    var length = result.Count;

                    GUILayout.BeginHorizontal();
                    if (GUILayout.Button("Prev", GUILayout.MaxWidth(40)))
                    {
                        resultPage--;
                        if (resultPage < 0)
                            resultPage = (length - 1) / PC;
                    }
                                resultPage  =  EditorGUILayout.IntField(resultPage,
GUILayout.MaxWidth(40));
                    if (resultPage > (length - 1) / PC)
                    {
                        resultPage = (length - 1) / PC;
                    }
                    if (GUILayout.Button("Next", GUILayout.MaxWidth(40)))
                    {
                        resultPage++;
                        if (resultPage > (length - 1) / PC)
                            resultPage = 0;
                    }
                       GUILayout.Label($"   of {(length - 1) / PC} ({length})",
GUILayout.MaxWidth(120));
                                    PC  =  Mathf.Clamp(EditorGUILayout.IntField(PC,
GUILayout.Width(80)), 1, 1000);
                    GUILayout.EndHorizontal();

                    for (int i = resultPage * PC; i < Mathf.Min(length, (resultPage +
1) * PC); i++)
                    {
                        GUILayout.BeginHorizontal();
                        if (GUILayout.Button("+", GUILayout.Width(20)))
                        {
                            t.LookSelect(result[i]);
                            t.LookApply();
                        }
                        EditorGUILayout.TextField(result[i]);
                        GUILayout.EndHorizontal();
                    }
                }
            }
        }
    }
```

ДОДАТОК В

Скрипти QuestionLook та QuestionLookEditor — логіка й інтерфейс підбору питання

```
using System.Collections;
using System.Collections.Generic;
using Tool.Maker;
using UnityEngine;


namespace Generator
{
    public class QuestionLook : MonoBehaviour
    {
        private GenManager manager;
        private GenAI genAI;
        bool inited;

        public GameObject qCellsGO;
        public bool showQCells;
        [Space]
        public bool autoRefresh;
        public GenQuestion Question;
        [Space]
        public QuestionInfo QuestionInfo;

        public void Init(GenManager manager, GenAI genAI)
        {
            this.manager = manager;
            this.genAI = genAI;

            inited = true;
        }

        public void SetQuestion(GenQuestion question)
        {
            Question = question;
            if (!question)
            {
                QuestionInfo = null;
                return;
            }
            QuestionInfo = question.qInfo;

        }

        private void Update()
        {
            if(qCellsGO.activeSelf != showQCells)
                qCellsGO.SetActive(showQCells);
        }

        private void OnValidate()
        {
            //qCellsGO.SetActive(showQCells);

            if (!autoRefresh)
                return;
```

```
                Refresh();
            }

            public void Refresh()
            {
                if (!inited || !Question)
                    return;

                Question.Refresh();
            }

            [HideInInspector] public int aiResultPage;
            [HideInInspector] public int aiPC = 5;
            [HideInInspector] public int aiHints = 5;
            [HideInInspector] public int aiSizeMin = 4;
            [HideInInspector] public int aiSizeMax = 5;
            [HideInInspector] public Language language;

            [HideInInspector]
            public List<string> FindAIResult = new List<string>();

            public async void TryGenerateAIHint(int amount, int sizeMin, int sizeMax)
            {
                if(Question == null)
                {
                    Debug.LogError("Empty Selection!");
                    return;
                }

                var genWord = Question.GetComponent<GenWord>();
                genWord.TryGetCrossWord(out var word);
                var w = word.Value;

                if (string.IsNullOrEmpty(w) || w.Contains(GenData.DOT))
                {
                    Debug.LogError("Wrong word!");
                    return;
                }

                     FindAIResult = await genAI.GetHints(language, amount, sizeMin,
sizeMax, w);
            }

            public void HintSelect(string s)
            {
                if (!inited || !Question)
                    return;

                if (QuestionInfo == null)
                    QuestionInfo = new QuestionInfo();
                QuestionInfo.text = s;
            }
        }
    }

    using System.Collections;
    using System.Collections.Generic;
    using UnityEditor;
    using UnityEngine;

    namespace Generator
    {
        [CustomEditor(typeof(QuestionLook), true)]
        public class QuestionLookEditor : Editor
```

```
        {
            public override void OnInspectorGUI()
            {
                var t = (QuestionLook)target;
                base.OnInspectorGUI();
                if (GUILayout.Button("Refresh Visual"))
                {
                    t.Refresh();
                }
                GUILayout.Space(20);

                DrawAISearch();

                void DrawAISearch()
                {
                    GUILayout.BeginHorizontal();
                                        if  (GUILayout.Button("Generate  AI  Hints",
GUILayout.MaxWidth(120)))
                    {
                        t.TryGenerateAIHint(t.aiHints, t.aiSizeMin, t.aiSizeMax);
                    }
                    GUILayout.Space(20);
                    GUILayout.Label("Amount", GUILayout.Width(60));
                          t.aiHints = Mathf.Clamp(EditorGUILayout.IntField(t.aiHints,
GUILayout.Width(40)), 1, 40);
                    GUILayout.Space(20);
                    GUILayout.Label("Size", GUILayout.Width(40));
                      t.aiSizeMin = Mathf.Clamp(EditorGUILayout.IntField(t.aiSizeMin,
GUILayout.Width(40)), 1, 50);
                      t.aiSizeMax = Mathf.Clamp(EditorGUILayout.IntField(t.aiSizeMax,
GUILayout.Width(40)), 1, 50);
                    GUILayout.Space(20);
                    t.language = (Language)EditorGUILayout.EnumPopup(t.language);
                    GUILayout.EndHorizontal();

                    if (t.FindAIResult.Count > 0)
                    {
                        GUILayout.Space(25);
                        DrawSearchList(t.FindAIResult);
                    }
                }

                void DrawSearchList(List<string> result)
                {
                    var length = result.Count;

                    GUILayout.BeginHorizontal();
                    if (GUILayout.Button("Prev", GUILayout.MaxWidth(40)))
                    {
                        t.aiResultPage--;
                        if (t.aiResultPage < 0)
                            t.aiResultPage = (length - 1) / t.aiPC;
                    }
                           t.aiResultPage = EditorGUILayout.IntField(t.aiResultPage,
GUILayout.MaxWidth(40));
                    if (t.aiResultPage > (length - 1) / t.aiPC)
                    {
                        t.aiResultPage = (length - 1) / t.aiPC;
                    }
                    if (GUILayout.Button("Next", GUILayout.MaxWidth(40)))
                    {
                        t.aiResultPage++;
                        if (t.aiResultPage > (length - 1) / t.aiPC)
                            t.aiResultPage = 0;
```

```
                    }
                        GUILayout.Label($"  of {(length - 1) / t.aiPC} ({length})",
GUILayout.MaxWidth(120));
                                t.aiPC = Mathf.Clamp(EditorGUILayout.IntField(t.aiPC,
GUILayout.Width(80)), 1, 1000);
                    GUILayout.EndHorizontal();

                        for (int i = t.aiResultPage * t.aiPC; i < Mathf.Min(length,
(t.aiResultPage + 1) * t.aiPC); i++)
                    {
                        GUILayout.BeginHorizontal();
                        if (GUILayout.Button("+", GUILayout.Width(20)))
                        {
                            t.HintSelect(result[i]);
                        }
                        EditorGUILayout.TextArea(result[i], GUILayout.Height(60));
                        GUILayout.EndHorizontal();
                    }
                }
            }
        }
    }
```

ДОДАТОК Г

Скрипт WordFind — робота з локальними словниками

```
using Res;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using UnityEngine;

namespace Generator
{
    public enum WordFilterType
    {
        all = 0,
        noun = 1,
        most = 2
    }

    [Serializable]
    public struct WordFilter
    {
        public Language language;
        public WordFilterType type;
        //public bool onlyNoun;
        public Vector2Int wasUsed;
    }
    public class WordFind : MonoBehaviour
    {
        public class WData
        {
            public string[][] wordsAll;
            public string[][] wordsNoun;
            public string[][] wordsMost;
            public Dictionary<string, int>[] wordsUses;

            public string[][] GetWords(WordFilterType type)
            {
                if (type == WordFilterType.noun)
                    return wordsNoun;
                else if (type == WordFilterType.most)
                    return wordsMost;
                else
                    return wordsAll;
            }
        }

        #region word data

        [SerializeField] private int maxWordLength;
        [SerializeField] private bool loadOnAwake;

        private WData wDataUA;
        private WData wDataEN;
        private WData wDataRU;

        private void Awake()
        {
            if (loadOnAwake)
```

```
                RefreshWordData();
            }

            [ContextMenu("RefreshWordData")]
            private void RefreshWordData()
            {
                LoadWordSet(Language.UA, ref wDataUA);
                LoadWordSet(Language.EN, ref wDataEN);
                LoadWordSet(Language.RU, ref wDataRU);
            }

            private void LoadWordSet(Language language, ref WData wd)
            {
                string[] txts = new string[] {"", "", "", ""};

                var p = $"WordFinder/words_{language}_";
                var p_base = $"{p}base";
                var p_noun = $"{p}noun";
                var p_most = $"{p}most";
                var p_used = $"{p}used";


                    if (Loader.Exists(Loader.PathType.Root, Loader.PatternType.txt,
p_base))
                        txts[0] = Loader.LoadText(Loader.PathType.Root,
Loader.PatternType.txt, p_base);

                    if (Loader.Exists(Loader.PathType.Root, Loader.PatternType.txt,
p_noun))
                        txts[1] = Loader.LoadText(Loader.PathType.Root,
Loader.PatternType.txt, p_noun);

                    if (Loader.Exists(Loader.PathType.Root, Loader.PatternType.txt,
p_most))
                        txts[2] = Loader.LoadText(Loader.PathType.Root,
Loader.PatternType.txt, p_most);

                    if (Loader.Exists(Loader.PathType.Root, Loader.PatternType.txt,
p_used))
                        txts[3] = Loader.LoadText(Loader.PathType.Root,
Loader.PatternType.txt, p_used);

                SetWordData(txts, ref wd);
            }

            private void SetWordData(string[] ts, ref WData wd)
            {
                wd = new WData();

                //All
                var dict0 = new List<string>[maxWordLength + 1];
                for (int i = 0; i <= maxWordLength; i++)
                    dict0[i] = new List<string>();

                        var s0 = ts[0].Split(new string[] { "\r\n", "\n" },
StringSplitOptions.RemoveEmptyEntries);
                for (int i = 0; i < s0.Length; i++)
                {
                    var key = s0[i].Length;
                    if (key > maxWordLength)
                        continue;
                    dict0[key].Add(s0[i]);
                }
```

```
                wd.wordsAll = new string[maxWordLength + 1][];
                for (int i = 0; i <= maxWordLength; i++)
                    wd.wordsAll[i] = dict0[i].ToArray();

                //Noun
                var dict1 = new List<string>[maxWordLength + 1];
                for (int i = 0; i <= maxWordLength; i++)
                    dict1[i] = new List<string>();

                        var s1 = ts[1].Split(new string[] { "\r\n", "\n" },
StringSplitOptions.RemoveEmptyEntries);
                for (int i = 0; i < s1.Length; i++)
                {
                    var key = s1[i].Length;
                    if (key > maxWordLength)
                        continue;
                    dict1[key].Add(s1[i]);
                }

                wd.wordsNoun = new string[maxWordLength + 1][];
                for (int i = 0; i <= maxWordLength; i++)
                    wd.wordsNoun[i] = dict1[i].ToArray();

                //Most
                var dict2 = new List<string>[maxWordLength + 1];
                for (int i = 0; i <= maxWordLength; i++)
                    dict2[i] = new List<string>();

                        var s2 = ts[2].Split(new string[] { "\r\n", "\n" },
StringSplitOptions.RemoveEmptyEntries);
                for (int i = 0; i < s2.Length; i++)
                {
                    var key = s2[i].Length;
                    if (key > maxWordLength)
                        continue;
                    dict2[key].Add(s2[i]);
                }

                wd.wordsMost = new string[maxWordLength + 1][];
                for (int i = 0; i <= maxWordLength; i++)
                    wd.wordsMost[i] = dict2[i].ToArray();

                //Used
                wd.wordsUses = new Dictionary<string, int>[maxWordLength + 1];
                for (int i = 0; i <= maxWordLength; i++)
                    wd.wordsUses[i] = new Dictionary<string, int>();

                 var s_used = ts[ts.Length - 1].Split(new string[] { "\r\n", "\n" },
StringSplitOptions.RemoveEmptyEntries);
                for (int i = 0; i < s_used.Length; i++)
                {
                    var key = s_used[i].Length;
                    if (key > maxWordLength)
                        continue;
                    if (!wd.wordsUses[key].ContainsKey(s_used[i]))
                        wd.wordsUses[key].Add(s_used[i], 0);
                    wd.wordsUses[key][s_used[i]]++;
                }
            }

            #endregion

            private List<string> result = new List<string>();
```

```
            public List<string> Select(string pattern, WordFilter filter)
            {
                var st = System.DateTime.Now;

                                                //Debug.Log($"[WFA]   Start   Refresh   :
{st.ToLongTimeString()}:{st.Millisecond}");

                result.Clear();

                if (!TryGetWData(filter.language, out var wd))
                    return result;

                var words = wd.GetWords(filter.type);

                var lenght = pattern.Length;

                if (lenght < words.Length)
                {
                    foreach (var w in words[lenght])
                    {

                        bool skip = false;
                        for (int i = 0; i < lenght; i++)
                        {
                            if (pattern[i] == GenData.DOT)
                                continue;
                            if (char.ToLower(pattern[i]) != char.ToLower(w[i]))
                            {
                                skip = true;
                                break;
                            }
                        }
                        if (skip)
                            continue;

                        if (filter.wasUsed.y > 99)
                        {
                            result.Add(w);
                            continue;
                        }

                        var ut = UsedTimes(w, filter);
                        if(ut >= filter.wasUsed.x && ut <= filter.wasUsed.y)
                            result.Add(w);
                    }
                }

                return result;

                var et = System.DateTime.Now;
                                                Debug.Log($"[WFA]   End   Refresh   :
{et.ToLongTimeString()}:{et.Millisecond}");
                Debug.Log($"[WFA] Refresh Time : {(et - st).TotalMilliseconds}");
                Debug.Log($"[WFA] Total Result Words : {result.Count}");
            }

            public int UsedTimes(string word, WordFilter filter)
            {
                word = word.ToLower();
                //Debug.Log($"w : {word}");

                if (!TryGetWData(filter.language, out var wd))
                    return 0;
```

```
            if (wd.wordsUses[word.Length].TryGetValue(word, out var v))
                return v;
            else
                return 0;
        }

        private bool TryGetWData(Language language, out WData wd)
        {
            if (language == Language.UA)
                wd = wDataUA;
            else if (language == Language.EN)
                wd = wDataEN;
            else if (language == Language.RU)
                wd = wDataRU;
            else
            {
                wd = new WData();
                return false;
            }

            return true;
        }
    }
}
```