Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)

Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

Здобувача вищої освіти Вернигора Олександр Олександрович
академічної групи 124-21-1
спеціальності 124 Системний аналіз
за освітньо-професійною програмою Системний аналіз

на тему: «Аналіз метрик залученості та утримання гравців у грі жанру 'ферма': визначення ключових показників для прийняття рішень»

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**Інституційною**	
кваліфікаційної роботи	*к.ф.-м.н., доц. Хом'як Т.В.*			
розділів:				
Інформаційно-аналітичний	*к.ф.-м.н., доц. Хом'як Т.В.*			
Спеціальний розділ	*к.ф.-м.н., доц. Хом'як Т.В*			
Рецензент	*д.т.н.,проф. Алексєєв М.А.*			
Нормоконтролер	*к.ф.-м.н., доц. Хом'як Т.В.*			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)
«____» _____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

здобувачу вищої освіти *Вернигорі О. О.* **академічної групи** *124- 21-1*
спеціальності: *124 Системний аналіз*
за освітньо-професійною програмою *Системний аналіз*
на тему «Аналіз метрик залученості та утримання гравців у грі жанру 'ферма': визначення ключових показників для прийняття рішень»
затверджену наказом ректора НТУ «Дніпровська політехніка» від *05.05.2025 р. №336-с*

Розділ	Зміст	Терміни виконання
1. Інформаційно-аналітичний розділ	*Проаналізувати ігровий проект, його структуру, механіки, та дані, отримані під час тестового запуску.*	10.03.2025 – 01.05.2025
2.Спеціальний розділ	*Визначити додаткові події, які необхідно збирати, щоб покращити якість та кількість даних для майбутнього аналізу, розробити систему для віддаленого налаштування балансу*	01.05.2025 – 10.06.2025

Завдання видано _____ *доц. Хом'як Т. В.*
(підпис) (прізвище, ініціали)

Дата видачі: *06.03.2025 р.*

Дата подання до екзаменаційної комісії: _____

Прийнято до виконання _____ Вернигора О. О.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 58 с., 17 іл., 1 табл., 14 додатки, 7 джерел.

Об’єктом дослідження в роботі є мобільна гра жанру «ферма» із трьома тематичними зонами: городнє поле, індустріальне поле та селищна зона.

Предметом дослідження є аналіз ключових метрик engagement та retention для оцінювання залученості й утримання гравців у контексті внутрішньоігрової економіки, онбордингу та систем мотивації.

Метою кваліфікаційної роботи є проведення аналізу ключових метрик engagement та retention, розроблення рекомендацій для оптимізації ігрового балансу, онбордингу й систем мотивації в мобільній грі жанру «ферма», систем та рекомендовані дані для збору аналітики та розробка і впровадження системи віддаленого редагування конфігів для подальшого проведення A/B тестувань.

Методи дослідження:

- Порівняльний аналіз екранних прототипів — для оцінювання відмінностей інтерфейсних рішень у різних ігрових зонах;
- Експертне оцінювання інтерфейсних рішень — для виявлення стилістичних і юзабіліті-проблем;
- Кореляційно-регресійний аналіз змодельованих телеметричних даних — для виявлення зв’язків між параметрами економіки гри та показниками Day 1 і Day 7 retention;
- Аналіз відгуків тестової групи користувачів — для оцінки прийнятності нових механік мотивації та онбордингу.

В інформаційно–аналітичному розділі наведено аналіз ігрового процесу в трьох тематичних зонах (городнє поле, індустріальне поле, селищна зона). Визначено ключові проблеми залученості й утримання гравців, сформульовано завдання дослідження.

У спеціальному розділі сформовано рекомендації що до збору даних для аналізу показників залученості та утримання, розроблено та реалізовано інструментарій для A/B-тестування внутрішньоігрової економіки, онбордингових сценаріїв та систем мотивації.

Галузь застосування охоплює розробку та оптимізацію мобільних ігор соціального та казуального сегментів, зокрема тих, що базуються на циклічних механіках «ферми» або схожих жанрів.

Значення роботи для науки полягає в універсальності розробленої методики поєднання кількісного й якісного аналізу, що може бути застосована навіть за обмеженого обсягу реальних даних завдяки моделюванню сценаріїв та сегментації аудиторії. Практична значущість полягає в підвищенні конкурентоспроможності продукту за рахунок оптимізованого геймбалансу, покращеного онбордингу й ефективних стимулів для повернення гравців після першого сеансу.

Прогнозні припущення вказують на подальший розвиток адаптивних систем персоналізації контенту та впровадження методів машинного навчання для точнішого прогнозування поведінки гравців і підвищення показників retention.

Ключові слова: ІНГЕЙДЖМЕНТ, РІТЕНШН, ЕКОНОМІКА ГРИ, ОНБОРДИНГ, МОБІЛЬНІ ІГРИ, АНАЛІЗ ДАНИХ, A/B-ТЕСТУВАННЯ, REMOTE CONFIG, UI/UX, ФЕРМА.

ABSTRACT

Explanatory Note: 58 pages, 17 illustrations, 1 table, 14 appendices, 7 sources.

The object of the study is a mobile game of the "farm" genre with three thematic zones: crop field, industrial field, and village area.

The subject of the study is the analysis of key engagement and retention metrics to assess player involvement and retention in the context of in-game economy, onboarding, and motivation systems.

The aim of the qualification work is to conduct an analysis of key engagement and retention metrics, develop recommendations for optimizing game balance, onboarding, and motivation systems in a mobile farm genre game, design and implement systems and data pipelines for analytics collection, and develop and integrate a remote configuration editing system for conducting A/B testing.

Research methods:

- Comparative analysis of screen prototypes — to evaluate differences in interface solutions across different game zones;
- Expert evaluation of interface solutions — to identify stylistic and usability issues;
- Correlation-regression analysis of simulated telemetry data — to identify relationships between game economy parameters and Day 1 and Day 7 retention rates;
- Analysis of feedback from a test user group — to assess the acceptability of new motivation and onboarding mechanics.

The information-analytical section presents an analysis of gameplay in the three thematic zones (crop field, industrial field, village area). Key issues of player engagement and retention are identified, and research objectives are formulated.

The special section provides recommendations for data collection to analyze engagement and retention indicators, as well as the development and implementation of tools for A/B testing of the in-game economy, onboarding scenarios, and motivation systems.

The field of application covers the development and optimization of mobile games in the social and casual segments, particularly those based on cyclical mechanics of the "farm" or similar genres.

The scientific significance of the work lies in the universality of the developed methodology combining quantitative and qualitative analysis, which can be applied even with limited real data through scenario modeling and audience segmentation. The practical significance lies in increasing the product’s competitiveness through optimized game balance, improved onboarding, and effective incentives to encourage players to return after the first session.

Forecast assumptions indicate the further development of adaptive content personalization systems and the implementation of machine learning methods for more accurate prediction of player behavior and improvement of retention indicators.

Keywords: ENGAGEMENT, RETENTION, GAME ECONOMY, ONBOARDING, MOBILE GAMES, DATA ANALYSIS, A/B TESTING, REMOTE CONFIG, UI/UX, FARM.

Зміст

ВСТУП

У сучасних умовах індустрія відеоігор демонструє стрімкий розвиток як у світовому, так і вітчизняному контексті. Особливої уваги заслуговують соціальні та мобільні проєкти жанру «ферма», які завдяки своїй доступності та гнучким механікам приваблюють широкий загал гравців. Проте зростаюча конкуренція вимагає від розробників не лише креативних ідей, а й чіткої стратегії утримання аудиторії. Аналіз ключових метрик залученості та утримання гравців дозволяє виявити сильні та слабкі сторони ігрового процесу, обґрунтувати необхідність оптимізації механік і підвищити ефективність прийняття управлінських рішень в ігровій розробці. Це особливо актуально для вітчизняних студій, які прагнуть підвищити конкурентоспроможність своїх проєктів на глобальному ринку.

Актуальність теми

У грі жанру «ферма» взаємодія гравця із механіками базується на повторюваних діях і поступовій мотивації через досягнення, що робить особливо важливим аналіз індикаторів, які відображають рівень залученості та лояльності аудиторії. Незважаючи на значну кількість досліджень у сфері гейміфікації та утримання гравців у різних жанрах (Rossi et al., 2020), для жанру «ферма» лишаються недостатньо вивченими вплив окремих механік на довгострокове утримання. Вітчизняні розробники потребують системного підходу до збору та інтерпретації метрик, що обґрунтує рішення щодо подальшої розробки та монетизації, а розроблена система віддаленого редагування конфігів полегшить процес A/B тестування. Тож проведення дослідження сприятиме формуванню методології аналізу метрик у грі жанру «ферма» та підвищенню якості продукту, над яким працює наша команда.

Мета і завдання дослідження

Метою цієї роботи є аналіз метрик залученості та утримання гравців у грі жанру «ферма» з визначенням ключових показників, що забезпечують своєчасне й обґрунтоване прийняття рішень для оптимізації геймплею та розробка системи віддаленого редагування конфігів для подальшого впровадження A/B тестувань.

Для досягнення поставленої мети вирішуються такі завдання:

1. Проаналізувати основні ігрові механіки об’єкта дослідження та описати їх вплив на поведінку гравця.
2. Провести кореляційний та регресійний аналіз зібраних даних отриманих під час відкритого тестування.
3. Визначити перелік ключових метрик залученості та утримання (загальний час у грі, середній час сесії, частота використання механік, швидкість прогресу тощо) та події, які доцільно збирати.
4. Розробити систему для віддаленого редагування конфігів, для можливості легкого впровадження A/B тестування, яка буде легко розширюватись та доповнюватись.
5. Сформулювати рекомендації щодо вдосконалення ігрового процесу на основі отриманих результатів та подальшого збору аналітики.

Об’єкт дослідження

Об’єктом дослідження є гра Mobile Farm: cultivate and build, та її механіки, над якими буде проведено аналіз.

Предмет дослідження

Предметом дослідження виступають окремі метрики залученості та утримання гравців (часові, частотні, поведінкові показники) та їхня взаємозалежність із ключовими механіками геймплею.

Методи дослідження

У роботі застосовуватимуться такі методи:

- **Описово-аналітичний** — для систематизації та класифікації ігрових механік;
- **Експериментальний** — для збору даних у тестових сесіях або моделювання сценаріїв;
- **Кореляційно-регресійний аналіз** — для виявлення взаємозв’язків між показниками;
- **Метод експертних оцінок** — для підтвердження висновків щодо впливу механік на мотивацію гравців;

- **Метод кейс-стаді** — для порівняння з існуючими практиками інших розробників.

Наукова новизна отриманих результатів

Наукова новизна роботи полягає в розробці цілісної методики аналізу взаємозв’язків між вибраними метриками залученості та утримання в контексті гри жанру «ферма». Вперше запропоновано комбінувати як якісні (експертні оцінки), так і кількісні (регресійні моделі) підходи для виявлення найбільш впливових механік. Крім того, запропоновані рекомендації щодо оптимізації ігрового процесу базуються на глибинному аналізі реальних даних, що дозволяє розширити практичні знання у сфері гейм-аналізу.

Практичне значення отриманих результатів

Практична цінність дослідження полягає в тому, що розроблена система віддаленого редагування конфігів може бути імплементована в будь яку гру без значних змін, а рекомендації щодо коригування механік забезпечать підвищення лояльності та довготривалого утримання гравців, що сприятиме зростанню комерційних показників проєкту.

РОЗДІЛ 1 ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ

1.1 Стан досліджень залученості та утримання в іграх жанру «ферма»

Жанр «ферма» у мобільних та соціальних іграх відзначається специфічними механіками повторюваних дій (посадка, збір урожаю, розвиток інфраструктури), що формують довгострокову залученість користувачів. У той же час факт монотонності рутинних операцій створює підвищений ризик відтоку аудиторії після першого тижня гри. Саме тому глибоке розуміння показників залученості (engagement) і утримання (retention) стало темою низки наукових та галузевих досліджень.

Перші академічні праці, присвячені гейміфікації та мотиваційним моделям у відеоіграх (Hamari & Koivisto, 2015; Meschtscherjakov et al., 2017), не орієнтувалися на жанр «ферма» як окрему нішу. Водночас останні кілька років у фокусі уваги опинилися дослідження впливу циклічних механік на довгострокове утримання, зокрема на прикладах популярних проєктів Hay Day (Supercell) і TownShip (Playrix).

1.2 Ігрові механіки «ферми» та їхній вплив на поведінку гравців

Ігрова механіка — це сукупність алгоритмізованих правил і взаємодій, які формують середовище та можливості для гравця всередині проєкту. Саме через механіки реалізуються основні виклики, винагороди та стимули, що керують поведінкою користувача. У жанрі «ферма» механіки набувають особливої ролі: циклічні дії (посадка, вирощування, збір урожаю) створюють відчуття прогресу й персональної участі в розвитку віртуального господарства, тоді як таймери і ресурси вводять елемент нетерпіння й очікування. Саме поєднання цих елементів визначає унікальну динаміку залученості та утримання в «фермах» і стає предметом детального аналізу в нашому дослідженні.

1.3 Загальний огляд цільового продукту

Mobile Farm: Cultivate and Build – представник гри жанру «Ферма». Ігри жанру *ферма* (farm games) мають набір характерних механік, які визначають

геймплей і роблять їх впізнаваними та привабливими для гравців. Нижче наведено основні ключові механіки, притаманні цьому жанру:

- **Посів і збір урожаю (Planting & Harvesting):**

  Гравець вирощує рослини, які мають цикл зростання (час на дозрівання), після чого їх можна зібрати для отримання ресурсів чи прибутку.

- **Розведення тварин:**

  Утримання домашніх тварин, які дають ресурси (молоко, яйця, шерсть тощо), зазвичай з потребою догляду: годування, чищення, лікування.

- **Будівництво та розвиток ферми:**

  Розширення ферми шляхом будівництва нових будівель (сараї, млини, склади), покращення існуючих об’єктів, розширення території.

- **Крафтінг і переробка ресурсів:**

  Можливість створювати продукти з вирощених ресурсів (наприклад, борошно з пшениці, сир з молока), що додає глибини ігрі.

- **Торговельна система / Продаж продукції:**

  Продавати зібраний врожай або виготовлені товари на ринку або замовникам для отримання ігрової валюти.

- **Система енергії або часу:**

  Обмеження дій (наприклад, кількість енергії або час дозрівання рослин), що стимулює гравця заходити до гри регулярно.

- **Завдання та квести:**

  Наявність місій або замовлень від NPC (персонажів), що мотивують до розвитку ферми та відкривають нові функції.

- **Декорування та кастомізація:**

  Гравець може прикрашати ферму, вибирати розташування об’єктів, що додає елемент творчості та індивідуальності.

- **Соціальні функції:**

  Можливість взаємодіяти з фермами друзів, дарувати подарунки, брати участь у кооперативних подіях або змаганнях.

- **Сезонність і події:**

Введення змін у геймплей у залежності від пори року чи свят (наприклад, святкові врожаї, тематичні квести).

Цільовий продукт має майже всі ключові механіки, які притаманні для жанру, які розділені на 3 тематичні поля. Таке розділення створює певний цикл геймплею від посадки, збору, переробки та продажу ресурсів для отримання прибутку.

1.4 Ігровий процес: аналіз та структура

У грі, аналіз якої проводиться в цій роботі, реалізовано більшість базових механік, притаманні жанру «ферма» з деякими особливостями в реалізації.

Досліджувана гра умовно поділена на три тематичні зони, які включають в себе частину ключових механік:

- **Городнє поле**: тут гравець сіє, доглядає та збирає врожай, формуючи базові ресурси для подальшої роботи.
- **Індустріальне поле**: на цій ділянці зводяться й удосконалюються фабрики, які переробляють зібрану сировину в більш цінні товари.
- **Селищна зона**: гравець відбудовує житлові та господарські будівлі, виконує замовлення мешканців і продає готову продукцію у власній крамничці.

Таке зонування створює чітку послідовність геймплею – від вирощування сировини до її переробки та реалізації, що одночасно мотивує гравця розвивати всі три «галузі» ферми.

1.4.1 Городнє поле

Городнє поле (рис. 1.1) є стартовою зоною ферми, де гравець видає завдання фермеру, який безпосередньо працює з рослинами та накопичує базові ресурси. У цій частині гри реалізовані такі ключові дії:

- **Посадка, догляд і збір врожаю**

  Фермер сіє насіння, поливає культуру та чекає на дозрівання, після чого збирає врожай.
- **Розширення грядок і садових клумб**

За внутрішню валюту гравець може придбати нові ділянки землі та плодові клумби, що дозволяє збільшити одночасний обсяг посадок, які стають доступними з підвищенням рівня.

- **Покращення складу**

  Розбудова та апгрейд складських приміщень підвищують ліміт зберігання врожаю, запобігаючи втраті ресурсів і створюючи запас для подальшої переробки.

- **Купівля нових інструментів фермера**

Швидкість збору та догляду за рослинами та грядками залежить від рівня інструментів. Із часом техніка зношується, тому регулярний апгрейд зменшує затримки в ігровому процесі.

Такий набір механік забезпечує базовий цикл «робота–очікування–нагорода», формуючи у гравця відчуття прогресу і мотивацію повертатися до гри.

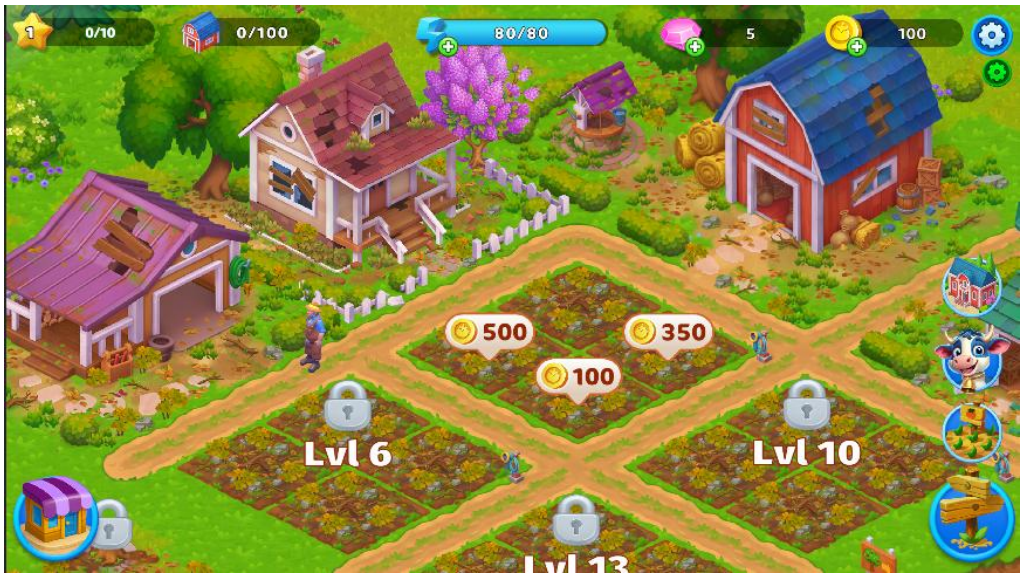


Рисунок 1.1 – Загальний вигляд першого поля.

1.4.2 Індустріальне поле

Індустріальне поле (рис 1.2) відповідає за переробку зібраних ресурсів у готові товари та формує середню ланку виробничого циклу. Тут гравець виконує такі ключові дії:

- **Придбання ділянок під фабрики**

За внутрішню валюту гравець може відкрити нові слоти для побудови виробничих будівель, збільшуючи виробничі можливості ферми.

- **Виготовлення товарів за рецептами**

  Кожна фабрика приймає певний набір сировини (наприклад, пшеницю або овочі) та виробляє кінцевий продукт згідно із заданим рецептом (борошно, хліб, сир, молоко тощо).

- **Покращення фабрик**

  Апгрейд збільшує внутрішню ємність фабрики для зберігання сировини та скорочує час виготовлення продукції.

- **Відкриття нових рецептів**

  Деякі фабрики при досягненні певного рівня розблоковують додаткові рецепти, що розширюють асортимент продукції та дають змогу отримати цінніші товари.

Такий набір механік стимулює гравця планувати розміщення фабрик і черговість їх покращення, водночас удосконалюючи виробничі ланцюжки та підвищуючи глибину залученості й утримання в грі.

Рисунок 1.2 – Загальний вигляд другого поля.

1.4.3 Селищна зона

Селищна зона (рис. 1.3) завершує виробничий ланцюжок, поєднуючи економічні та сюжетні елементи гри. Основні механіки цієї зони:

- **Центральні об’єкти**
  - **Крамничка** — місце роздрібного продажу продукції: гравець може виставити обмежену кількість товарів, які через певний час куплять мешканці.
  - **Машина скупника** — швидкий спосіб оптового збуту надлишкових ресурсів за лояльними цінами.
- **Замовлення від мешканців**
  - На початку доступні дві стартові будівлі, які видають замовлення на виготовлення певних товарів (наприклад, випічки чи молочні продукти).
  - Виконання замовлень приносить будівельні матеріали для відновлення нових об’єктів.
- **Відновлення села та розвіювання туману**
  - З отриманням нового рівня, гравцю відкриваються нові зони з занедбаними будівлями, які йому необхідно відновити.
  - Після накопичення достатньої кількості матеріалів із замовлень гравець може відбудовувати нові будівлі.
  - Кожен відбудований об’єкт розбиває фрагмент туману й відкриває доступ до нових замовлень і декорацій.
- **Динаміка продажу в крамничці**
  - Мешканці села ходять площею і мають випадково згенеровані бажання, що створює різноманітність у швидкості купівлі товарів.

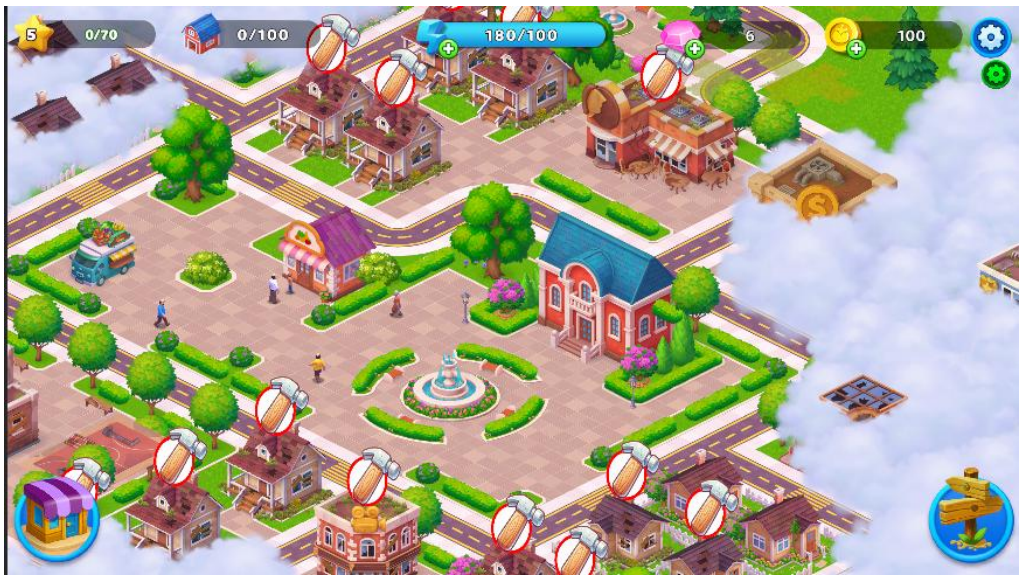


Рисунок 1.3 – Загальний вигляд третього поля.

Таке поєднання механік забезпечує гравцю широкий спектр дій — від економічного планування до сюжетного розвитку села — і водночас підтримує інтерес через постійне відкриття нового контенту та різноманітні цільові завдання.

У сукупності три тематичні зони — городнє поле, індустріальне поле та селищна зона — утворюють замкнений виробничий ланцюг «ферми»: від первинного збору сировини до її переробки й фінальної реалізації. Кожна зона пропонує власний набір викликів і винагород, що підтримує в гравця відчуття прогресу:

- Городнє поле закладає базу, формуючи звичку регулярних візитів через циклічні дії та таймери.
- Індустріальне поле занурює в стратегію планування й оптимізації виробництва, збільшуючи глибину залученості.
- Селищна зона додає сюжету та соціальної мотивації через відновлення споруд, виконання замовлень і взаємодію з NPC.

Разом вони створюють послідовний, але різноманітний геймплей, який стимулює як короткострокову активність (щоденні завдання, продажі в крамничці), так і довгострокове утримання. (рис. 1.4)

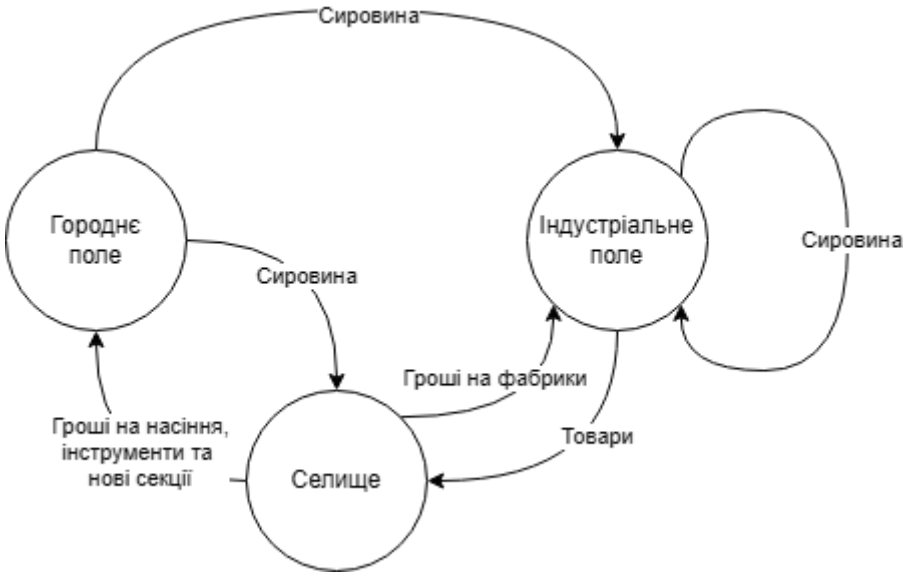


Рисунок 1.4 – Схема взаємодії між полями

1.5 Узгодженість інтерфейсів та візуального стилю

Візуальна презентація гри демонструє стилістичну фрагментованість між окремими частинами ігрового процесу. Основний HUD, який зустрічає гравця на першому полі, виконаний у яскравих, насичених кольорах із чітко вираженим казуальним стилем. Проте подальші інтерфейси гри — як всередині першого поля, так і на інших локаціях — не відповідають цій стилістиці, що створює враження неузгодженості.

На першому полі гравець може взаємодіяти з об'єктами для отримання доступу до інтерфейсів: складу (Warehouse), магазину (Shop), інтерфейсу насіння та саджанців (Seed and Sapling). Їх можна відкрити як натисканням на відповідні іконки, так і взаємодією з об’єктами на полі. Проте структура доступу не завжди інтуїтивна: зокрема, інтерфейс саджанців викликається через натискання на сховище, що неочевидно для нових гравців. Також деякі значення, наприклад, числа біля мішечків насіння, є незрозумілими за відсутності підказок чи пояснень. Усі ці інтерфейси оформлені у дерев’яних тонах, що візуально відрізняється від основного HUD, з яким гравець взаємодіє найперше. Візуальне оформлення – приємне, а занедбані будівлі корелюють з сюжетом, та дають гравцю ціль та натяк на прогрес, що будівлі будуть відновлюватись з часом.

На другому полі, де розташовані фабрики, інтерфейси мають вигляд технічних заглушок. Хоча це може бути допустимим на стадії публічного

тестування, у фінальному релізі такий вигляд викликає враження незавершеності. Стилістично інтерфейси фабрик зберігають коричневі тони, що узгоджується з інтерфейсами першого поля, але все ще не гармонізує з основним HUD.

На третьому полі спостерігається покращення загальної візуальної презентації — будівлі реагують на взаємодію з гравцем, що додає динаміки. Присутність NPC, які ходять полем і підходять до прилавку, вносить живість у, здебільшого, статичну картинку. Візуально будівлі на полі мають зруйнований вигляд, що підтримується хінтом у вигляді молотка. З інтерфейсами ситуація подібна: інтерфейси тут вже не виглядають як заглушки, але залишають контраст із оформленням першого поля. Загальний вигляд села деталізований, але хінти на недобудовані будівлі виглядають як тимчасові або технічні позначки, що не мають належної графічної реалізації. Інтерфейси тут також витримані у коричневих тонах, продовжуючи лінію попередніх полів, але так і не зводяться до єдиного візуального стандарту з HUD.

Додатково варто звернути увагу на **квест-бар** та **діалогові вікна**: їхня реалізація також залишає відчуття недопрацьованості. Квест-бар не має чіткої візуальної ієрархії або привабливого дизайну, а діалогове вікно виглядає сирим — що може негативно вплинути на імерсію та враження від сюжету. Якщо врахувати, що частина гравців повідомляла про баг з «квестом продажу», що блокує подальше проходження, це підсилює загальне відчуття незавершеності проєкту та потенційно сприяє відтоку аудиторії.

Попри різнотипність інтерфейсів, позитивним аспектом є те, що всі внутрішні вікна мають анімацію появи — спливаючий pop-up ефект, який створює приємне враження взаємодії. Проте сам по собі ефект не компенсує відсутність візуальної та функціональної узгодженості між основними елементами UI.

У підсумку, відсутність єдиного візуального підходу до оформлення інтерфейсів у різних частинах гри створює відчуття стилістичної незавершеності. Для підтримання занурення та підвищення рівня довіри до

продукту, важливо уніфікувати візуальні рішення: як щодо кольорової палітри, так і щодо стилю іконок, шрифтів та способу виклику інтерфейсів.

1.6 Ігровий баланс

Ігровий баланс є ключовим аспектом будь-якої гри, оскільки від нього залежить мотивація гравця до подальшого розвитку та взаємодії з ігровим світом. У представленій грі, однак, баланс фактично відсутній, що створює низку проблем, які негативно впливають на ігровий досвід.

Основним економічним ресурсом є монети, отримати які можна лише двома шляхами: підвищенням рівня або продажем товарів в крамниці. Відсутність монет у якості винагороди за виконання замовлень, що є основним способом розвитку села, створює суттєвий конфлікт. Гравець опиняється у ситуації, коли він змушений розриватися між необхідністю виконувати замовлення, що дають матеріали та досвід, але не приносять монет, та потребою накопичувати монети для того, щоб взагалі мати можливість виконувати ці замовлення.

Прогрес у розвитку, який реалізується через підвищення рівня, також виявляється малопомітним і не завжди стимулює до активної гри. Досвід, необхідний для переходу на новий рівень, переважно здобувається за сюжетні квести, кількість яких на рівні обмежена, а їх виконання потребує монетних вкладень, при цьому не повертаючи гравцеві монети у вигляді винагороди. Така система породжує замкнене коло, коли для виконання завдань потрібні монети, які отримати можна лише продажем, що гальмує природний розвиток ігрового процесу.

Ще однією проблемою є переповнення складу матеріалами, які гравець отримує внаслідок виконання замовлень. Оскільки витратити ці матеріали без підвищення рівня, а отже без відкриття нових будівель і регіонів, зрештою стає практично неможливо, накопичення ресурсів стає проблемою. Відсутність можливості ефективно позбуватись надлишку матеріалів призводить до потенційного зниження зацікавленості в грі. До того ж, для відбудови нових

об’єктів потрібні конкретні матеріали, яких може бракувати, що додатково ускладнює прогрес і формує відчуття застою.

Наявна економічна модель гри демонструє низку вузьких місць, які слід усунути для поліпшення ігрового балансу. Рекомендується впровадити додаткові джерела надходження монет, зокрема за виконання замовлень, що допоможе гравцю більш плавно управляти ресурсами. Також доцільно розширити спектр активностей, які дозволять отримувати винагороди, і розробити механізми утилізації або обміну надлишкових матеріалів, щоб уникнути переповнення складу та надати додатковий прибуток. Такий підхід сприятиме більш динамічному і збалансованому розвитку ігрового процесу, що позитивно вплине на залучення та утримання гравців.

1.7 Онбординг

Ключовим елементом успішності мобільної гри є те, як вона зустрічає нового гравця та пояснює основні механіки. Як зазначається в офіційних рекомендаціях Apple, грамотний онбординг повинен бути інтуїтивно зрозумілим, поступовим і вмонтованим у сам ігровий процес, аби не перевантажувати користувача та водночас зацікавити його з перших хвилин взаємоді. Саме такий підхід реалізовано в грі, що розглядається в цій роботі.

Гра розпочинається діалогом двох братів — головного героя Поло та його брата Марко (рис. 1.5), які повертаються на ферму своїх батьків. Цей сюжетний вступ задає емоційний тон і створює контекст для подальших дій гравця. Після вступного діалогу гравець одразу переходить до першого навчального завдання — розчищення території для посадки культур. Подальше навчання включає посадку, пришвидшення росту та збір врожаю. Кожен етап супроводжується діалогами, які в доступній формі пояснюють наступні дії, що відповідає рекомендаціям BBC щодо навчання через інтегровану історію, а не окремі інструкції.

Рисунок 1.5 – Перший діалог

Після завершення базових аграрних дій, гра поступово веде гравця до нових механік. На індустріальному полі гравець навчається будівництву фабрик та переробці ресурсів, а далі — продажу продукції на сільському ринку. У цьому процесі відбувається знайомство з новими персонажами — мером Артуром та Філіпом, які в подальшому виступають як NPC, що видають квести. Така побудова вступного етапу гри відповідає загальним принципам ефективного онбордингу: поступове введення нових систем, знайомство з ключовими персонажами, та надання нагороди за кожне успішно виконане завдання.

Таким чином, розглядувана гра реалізує поетапне, сюжетно обґрунтоване залучення гравця до ключових механік, що підвищує рівень утримання користувачів та сприяє загальній успішності проєкту.

Аналіз даних першого публічного тестування підтверджує важливість ефективного онбордингу. Згідно з діаграмою (Рис. 1.6.), лише близько 38% гравців завершили весь цикл навчання. Найбільше відсіву відбулося на етапах, пов’язаних із будівництвом та переробкою ресурсів, що вказує на можливі труднощі з інтерфейсом чи сприйняттям ігрових механік. Це підкреслює необхідність оптимізації навчального процесу для підвищення утримання. нових гравців.

Деякі гравці повідомляли про труднощі з подальшим проходженням сюжету через некоректну роботу квесту з продажем ресурсів. Це, ймовірно,

стало однією з причин втрати частини аудиторії після проходження навчання. Подібна ситуація вказує на недостатній рівень попереднього тестування та потребу у більш ретельній перевірці критично важливих ігрових механік перед публічним релізом.

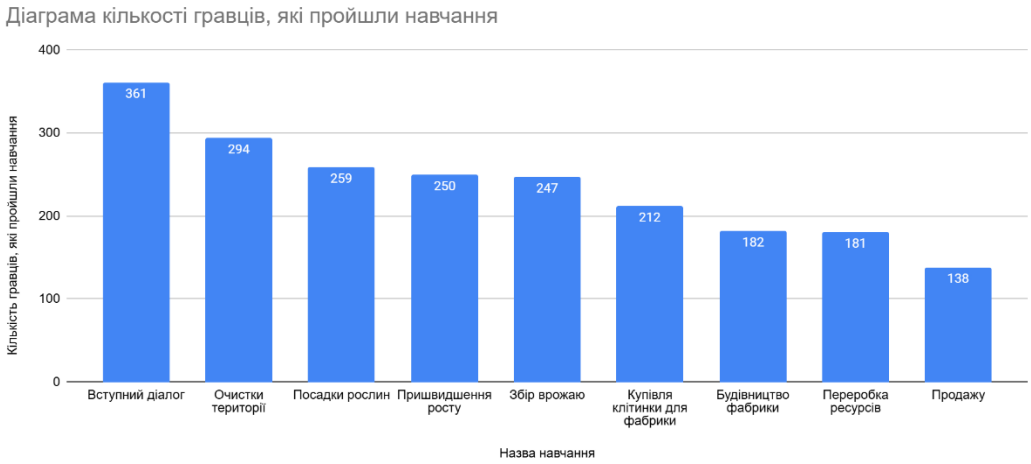


Рисунок 1.6 – Діаграма кількості гравців, які закінчили навчання.

Загальний огляд діаграми

- Найбільше гравців почали навчання, але не всі дійшли до кінця:
  - Вступний діалог пройшли 361 гравець, але до етапу "Продажу" дійшло лише 138 — тобто лише ~38% завершили весь навчальний цикл. Це вказує на значну втрату користувачів у процесі онбордингу.
- Найбільші втрати — після етапів, пов’язаних із будівництвом і переробкою:
  - Після завдання “Купівля клітинки для фабрики” (212 гравців) кількість гравців різко зменшується: на “Будівництві фабрики” лишається лише 182, а на “Переробці ресурсів” — 181. Це може свідчити про ускладнення геймплею або неінтуїтивність інтерфейсу на цьому етапі.
- Перші етапи проходять значно більше гравців:
  - Етапи “Очищення території” (294), “Посадка рослин” (259), “Прискорення росту” (250), “Збір врожаю” (247) мають відносно

невелике зниження. Це означає, що початкові механіки прості, добре пояснені та не викликають труднощів.

- Продаж — найменш досягнутий етап:
    - Етап "Продажу" пройшли тільки 138 гравців, що може свідчити або про недостатню мотивацію гравців дійти до фіналу навчання, або про занадто довгий/складний шлях до нього.

1.8 Аналіз поведінки гравців після проходження онбордингу

Після завершення навчання (онбордингу), активність гравців демонструє помітне зниження. Як видно з діаграми (рис. 1.7), проходження сюжетних квестів після останнього етапу навчання супроводжується каскадним спадом кількості гравців. Така динаміка може свідчити про складність перших самостійних завдань, недостатню мотивацію або втрату відчуття прогресу після завершення чіткого скриптованого вступу.

Рисунок 1.7 – Загальна кількість пройдених сюжетних нод.

Додатково тестування гри виявило суттєві недоліки у системі нагород. Лише перший квест надає монети як винагороду, натомість наступні завдання — лише досвід або невелику кількість алмазів. Така ситуація порушує баланс між зусиллям та нагородою, що негативно впливає на мотивацію гравця продовжувати проходження. Відсутність відчутного прогресу або винагороди вже на ранніх етапах може бути однією з причин вибування великої кількості гравців на перших сюжетних квестах після онбордингу.

1.9 Основні метрики залученості: час у грі, час сесії, частота взаємодій

У цьому підрозділі розглянемо ключові кількісні показники, що відображають рівень залученості гравців у грі жанру «ферма». До таких метрик належать загальний час у грі, середня тривалість однієї сесії та частота взаємодій

із основними елементами інтерфейсу. Аналіз цих показників дає змогу оцінити, наскільки гравців «затягує» геймплей, виявити пікові моменти активності та зрозуміти, які аспекти інтерфейсу або механік потребують оптимізації. У подальших розділах ми визначимо методи збору цих даних, опишемо способи їх обробки та проаналізуємо, які висновки можна зробити на їх основі для підвищення engagement.

1.9.1 Загальний час у грі (Total Play Time)

- **Визначення**: сукупна тривалість усіх сесій одного гравця за певний період (день, тиждень, місяць).
- **Значення**: показує, наскільки гра в цілому “затягує” аудиторію; довгий кумулятивний час свідчить про сильну залученість.
- **Збір даних**: телеметрія клієнта фіксує початок і кінець кожної сесії, а бекенд агрегує дані.
- **Аналіз**:
    - **Розподіл по когортах**: порівняння нових і досвідчених гравців.
    - **Тренди**: зростання чи зниження середнього часу за релізні патчі чи оновлення контенту.
    - **Кореляція з монетизацією**: чи схильні “тяжкі” гравці більше витрачати в магазині.

1.9.2 Середній час однієї сесії (Average Session Length)

- **Визначення**: середня тривалість однієї ігрової сесії у хвилинах.
- **Значення**: дозволяє оцінити, наскільки гра заточена на довгі або короткі епізоди.
- **Збір даних**: як і Total Play Time, але ділять на число сесій.
- **Аналіз**:
    - **Гістограма сесій**: чи є чіткий “піковий” час (5–10 хв)?
    - **Сегментація**: новачки vs ветерани.

- **Вплив подій**: як довжина сесії змінюється під час сезонних івентів чи акцій.

1.9.3 Частота взаємодій (Interaction Frequency)

- **Визначення**: кількість ключових подій (тапи посадки, збору, відкриття меню, кліків по завданнях) за одиницю часу (наприклад, подій за хвилину).
- **Значення**: показує, наскільки активно гравець взаємодіє з ігровим простором; високі значення можуть свідчити як про сильну залученість, так і про можливе перевантаження інтерфейсу.
- **Збір даних**: SDK аналітики генерує подію на кожну взаємодію; бекенд логує timestamps.
- **Аналіз**:
  - **Time-series**: як змінюється частота взаємодій у межах однієї сесії (найактивніші 1–2 хвилини).
  - **Когорти**: порівняння нових гравців (1–3 день) і досвідчених (>14 днів).
  - **Зв'язок із retention**: чи вищі interaction rates у гравців, які повертаються на Day 7.

1.10 Метрики утримання: ретеншн, LTV, churn rate

У цьому підрозділі розглянемо головні показники, що оцінюють здатність гри утримувати гравців на різних етапах після першого входу. До таких метрик належать:

- **Retention (ретеншн)** — відсоток гравців, які повернулися в гру на певний день після встановлення (Day 1, Day 7, Day 30 тощо).
- **LTV (Lifetime Value)** — сумарний дохід від одного гравця за весь час його активності в грі.
- **Churn rate** — частка гравців, які перестали грати (не здійснювали жодної сесії) у встановлений період.

1.10.1 Retention

- **Визначення:**

$$\text{Retention}_{\text{Day N}} = \frac{\text{кількість гравців, що зіграли в День N}}{\text{кількість нових гравців у День 0}} \times 100\%.$$

- **Значення:** дозволяє виміряти, наскільки гра затягує гравців після знайомства.
- **Методика збору:** через телеметрію фіксують дати останніх сесій гравців; когорти за датою реєстрації.
- **Аналіз:**
- Сформувати когорти нових гравців і відстежити їхній повернення на Day 1, 7, 30.
- Порівняти retention до та після ключових оновлень (нові механіки, івенти).

1.10.2 Lifetime Value (LTV)

- **Визначення:**

$$\text{LTV} = \sum_{t=0}^{T} \frac{R_t}{(1+d)^t}$$

де R_t — дохід від гравця в день t, d — ставка дисконту, T — період аналізу.

- **Значення:** показує економічну цінність гравця та допомагає коригувати маркетингові витрати на залучення (CAC).
- **Методика збору:** агрегування покупок і внутрішніх транзакцій у профілі користувача.
- **Аналіз:**
    - Прогнозування LTV на основі ранніх метрик engagement і retention (наприклад, за допомогою регресійних моделей).

- Сегментація гравців за LTV (низький, середній, високий) для таргетованих кампаній.

1.11 Аналіз ключових метрик зібраних в ході тестового запуску.

У цьому підрозділі наведено детальний огляд і інтерпретацію основних аналітичних показників, отриманих під час тестового запуску додатку. Аналіз базується на графіках користувацької активності, метриці утримання (retention), середньому часу взаємодії, а також на розподілі користувачів за географією, пристроями та подіями, що генеруються в процесі використання.

1.11.1 Користувацька активність за часом

На графіку “User activity over time” (рис. 1.8.) видно еволюцію кількості активних користувачів у різні часові проміжки:

- **30-денних активних користувачів** (синя крива) наприкінці періоду становить **423** осіб.
- **7-денних активних користувачів** (фіолетова крива) сягає максимуму близько **17** осіб.
- **1-денних активних користувачів** (рожева крива) у пікові дні — **2** особи.

Зі спостережень:

1. Під час першого тижня запуску (приблизно 4–11 травня) відбувся різкий стрибок кількості 30-денних активних користувачів — з нульових значень до понад 400. Це свідчить про успішне залучення аудиторії на етапі анонсу/релізу.
2. У той же самий період кількість 7-денних активних користувачів підскочила до позначки близько 17, що вказує на короткостроковий інтерес і початкове тестове охоплення.
3. Після піка у середині травня спостерігається поступове згасання кривих: кількість нових користувачів стабілізується, а інтенсивність щоденної (1-денної) активності залишається на дуже низькому рівні (приблизно 1–2 користувачі).

Отже, тестовий запуск залучив значну групу зацікавлених користувачів на початку, проте вже через кілька днів активність суттєво знизилась, що може свідчити про недостатню утримувальну здатність застосунку.

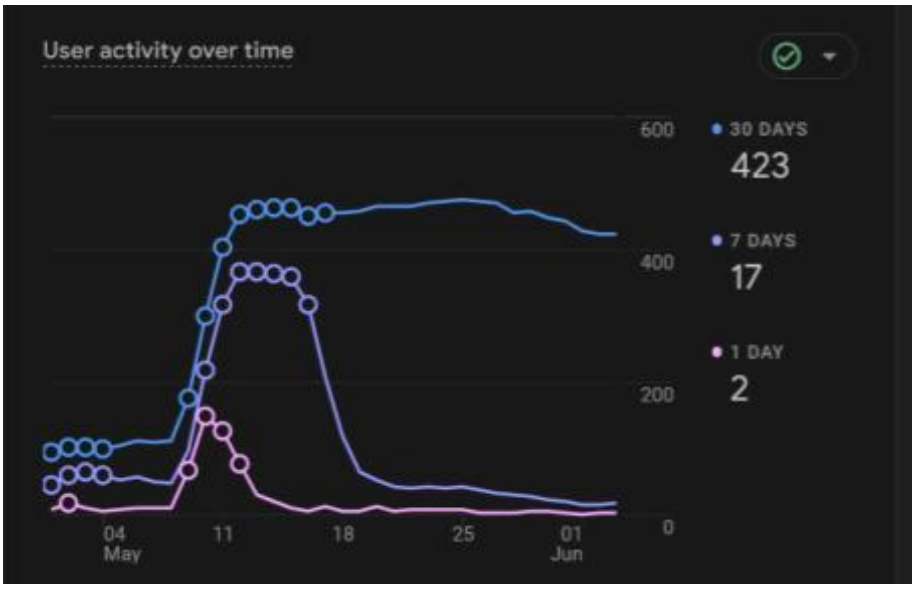


Рисунок 1.8 – Користувацька активність за часом

1.11.2 Події (Events) та інтенсивність їхнього виконання

Таблиця 1.1 демонструє перелік основних подій (event name), загальну кількість фіксацій (event count), кількість унікальних користувачів, які їх викликали (total users), а також середню кількість подій на одного активного користувача (event count per active user). Вона дає узагальнене уявлення про загальну кількість завершених нод, піднятих рівнів, тощо.

Таблиця 1.1

Події (Events) та інтенсивність їхнього виконання

Назва події	Кількість подій	Унікальних користувачів	Подій на одного активного користувача
m_node_completed	5174	358	14,49
screen_view	1722	451	3,82
user_engagement	1591	382	4,16
session_start	1110	452	2,46
m_level_up	899	221	4,13
first_open	442	442	1,00

Ключові висновки:

1. **m_node_completed (пройдено/завершено елемент/ноду):**
   - Найбільша кількість подій (5174) і високе середнє значення (≈ 14,5 подій/користувач).
   - Це свідчить про те, що користувачі, які продовжували користуватися додатком, активно проходили контент (уроки/рівні/модулі). Проте кількість унікальних користувачів (358) менша, ніж загальне число тих, хто встановив (442), тобто приблизно 80 % загальної аудиторії виконали хоча б одну подію цього типу.
2. **screen_view (перегляд екрану):**
   - 1722 події, залучено 451 унікального користувача.
   - Середньо ~3,8 переглядів на одного активного користувача. Це означає, що майже всі, хто відкрив додаток хоча б раз, переглядали кілька екранів/секцій, але не всі дійшли до завершення вузлів (m_node_completed).
3. **user_engagement (взаємодія користувача):**
   - 1591 фіксація, 382 унікальних користувачі.
   - Середнє ≈ 4,16 подій на користувача вказує на помірний рівень залучення (натискання, скролінг, інтеракції з інтерфейсом), але приблизно 60 % встановивших (382 з 442) справді «взаємодіяли» із застосунком протягом сесій.
4. **session_start (старт сесії):**
   - 1110 запусків сесії (452 унікальних користувачів).
   - У середньому кожен активний користувач розпочинав сесію ≈ 2,46 рази за весь період тесту. Це означає, що лише менша частина користувачів поверталася повторно.
5. **m_level_up (підняття рівня):**
   - 899 подій підвищення рівня у 221 користувача (≈ 4,1 на кожного).
   - Сигналізує про те, що близько 50 % усіх встановивших дійшли до моменту «підняття рівня» (Level Up), однак подальше залучення

було обмеженим, адже ці 221 із 442 становлять половину стартової аудиторії.

6. **first_open (первинне відкриття додатку):**
    - 442 події, ровно по одній на кожного встановившого.
    - Це підтверджує, що саме 442 користувачі інсталювали й відкрили додаток хоча б одного разу.

Висновок за подіями. Хоча факт першого відкриття був високим (442 користувачі), уже значно менша кількість доходила до завершення вузлів та підняття рівня, а ще менше—поверталося на повторні сесії. Це вказує на достатньо слабку утримувальну здатність після початкового інтересу.

1.11.3 Середній час взаємодії (Average engagement time)

На графіку “Average engagement time per active user” (рис 1.9) відображено середню тривалість активного користувача (у хвилинах і секундах) та кількість сесій на користувача за різні дати (квітень–травень):

- **Станом на середину періоду** (кінець квітня) середній час взаємодії досяг піка близько **50–55 хвилин**.
- Після цього (кінець квітня — початок травня) показник стабілізувався в межах **30–40 хвилин**, а ближче до серединитравня впав до приблизно **16 хв 40 с**.
- В середньому за весь відтинок часу застосунок демонструє показник ≈ **16 хв 52 с** активного користувача (як зазначено у заголовку графіка).
- Кількість задіяних (engaged) сесій на одного активного користувача становить **2**.

Інтерпретація:

1. У пікові дні (частково припадають на тиждень після релізу) користувачі проводили значний час у додатку — близько години, що свідчить про високу зацікавленість контентом під час першого знайомства.
2. Надалі середній час поступово зменшився, що може бути пов’язано зі зниженням новизни продукту та зменшенням інтенсивності залучення

(багато користувачів більше не поверталися або досліджували додаток поверхово).

3. Середня кількість сесій (2) вказує, що користувачі, які повернулися, зазвичай проводили у додатку дві взаємодії за аналізований період, що знову підтверджує низьку частоту повторних візитів.

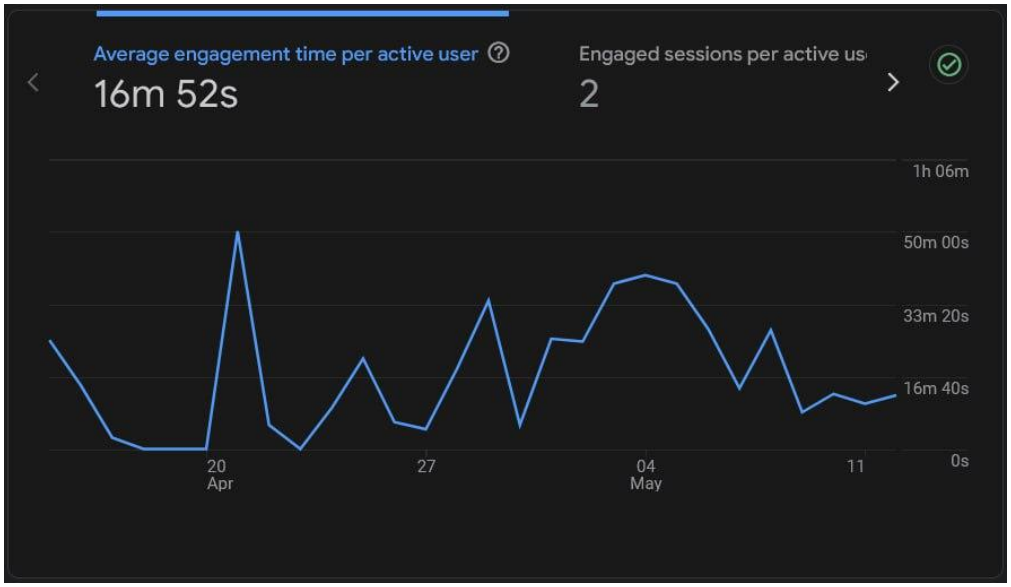


Рисунок 1.9 – Середній час взаємодії

1.11.4 Утримання користувачів (User retention)

Графік “User retention” (рис 1.10) демонструє, яку частку користувачів, що встановили додаток у певний день (Day 0), вдалося утримати на наступні дні:

- **Day 1 (день після встановлення):** утримання складає **15,8 %**.
- **Day 7:** показник опускається майже до **0 %**.
- Надалі, аж до **Day 40**, утримка залишається в межах **1–3 %**, фактично стабілізуючись близько до **0 %** для більшості когорти.

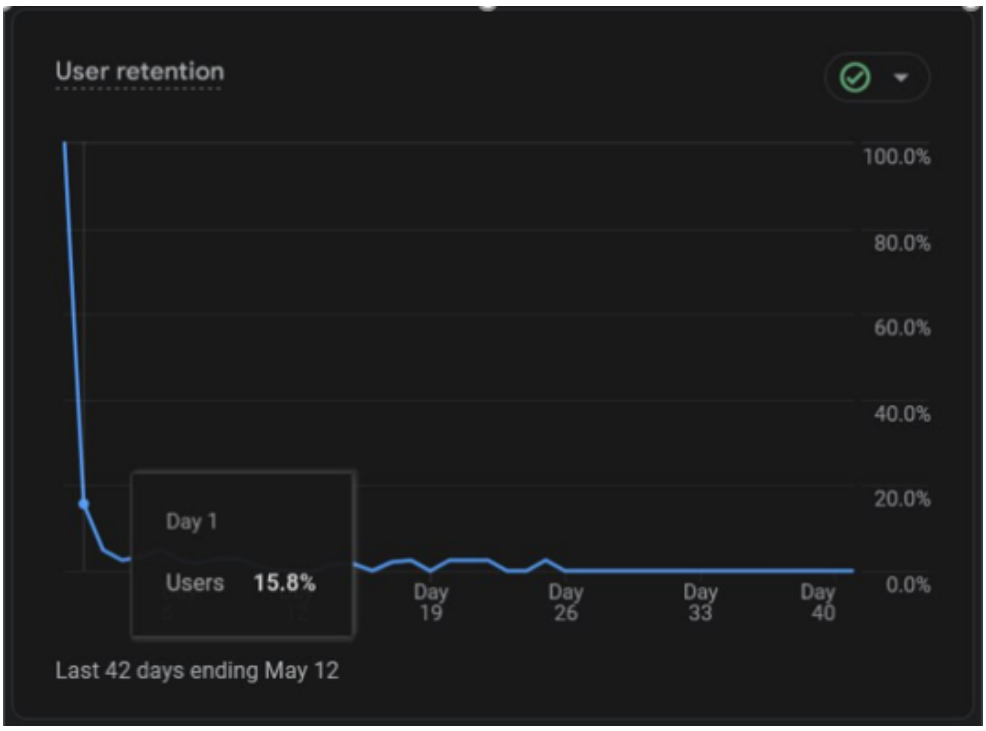


Рисунок 1.10 – Середній час взаємодії

Висновок:

1. **Низький показник Day 1 (15,8 %)** говорить про те, що більшість користувачів не поверталася до застосунку навіть наступного дня після встановлення/першого відкриття. Це свідчить про слабку первинну залученість: контент або UX/UI не спонукали до повторного використання одразу.
2. **Практично повне «висихання» (drop-off)** до Day 7–Day 10 відображає критично низьке довгострокове утримання — лише поодинокі користувачі повертались для подальших сесій.
3. Загальна форма кривої retention є типовою для продуктів, які не пропонують явної причини повертатися (напр. push-повідомлень, мотиваційних елементів тощо). Це вказує на необхідність опрацювання механік рети-менту (нагороди, гейміфікація, повідомлення).

1.11.5 Розподіл активних користувачів

Нижній правий блок (рис 1.11) відображає, з яких країн надійшла аудиторія, та які пристрої (моделі смартфонів) найбільш поширені серед активних користувачів.

Активні користувачі за країнами (Active users by Country)

- **United States: 347** активних користувачів.
- **Ukraine: 87** користувачів.
- **Germany: 3**, **Czechia: 2**, **Poland: 2**, **Spain: 1**, **Micronesia: 1**.

Висновок: приблизно **82 %** усієї аудиторії (347 із 423 30-денних активних) припадає на США, ≈ **21 %** (87 із 423) — на Україну. Решта країн практично не впливають на статистику. Це може бути результатом таргетингу реклами чи локалізаційних налаштувань тестової кампанії.

Активні користувачі за моделями пристроїв (Active users by Device model)

- **Redmi Note 5: 30** користувачів.
- **CPH2069: 26** користувачів.
- **CPH2481: 13** користувачів
- **Moto G 5G 2024: 13** користувачів
- **SM-S908U: 12** користувачів
- **Redmi Note 7: 9** користувачів
- **SM-S906U: 9** користувачів

Висновок: серед тестових користувачів домінують Android-пристрої середнього сегмента (Redmi Note 5 і схожі). Це може означати, що тестери переважно використовують Android-пристрої середнього цінового діапазону.

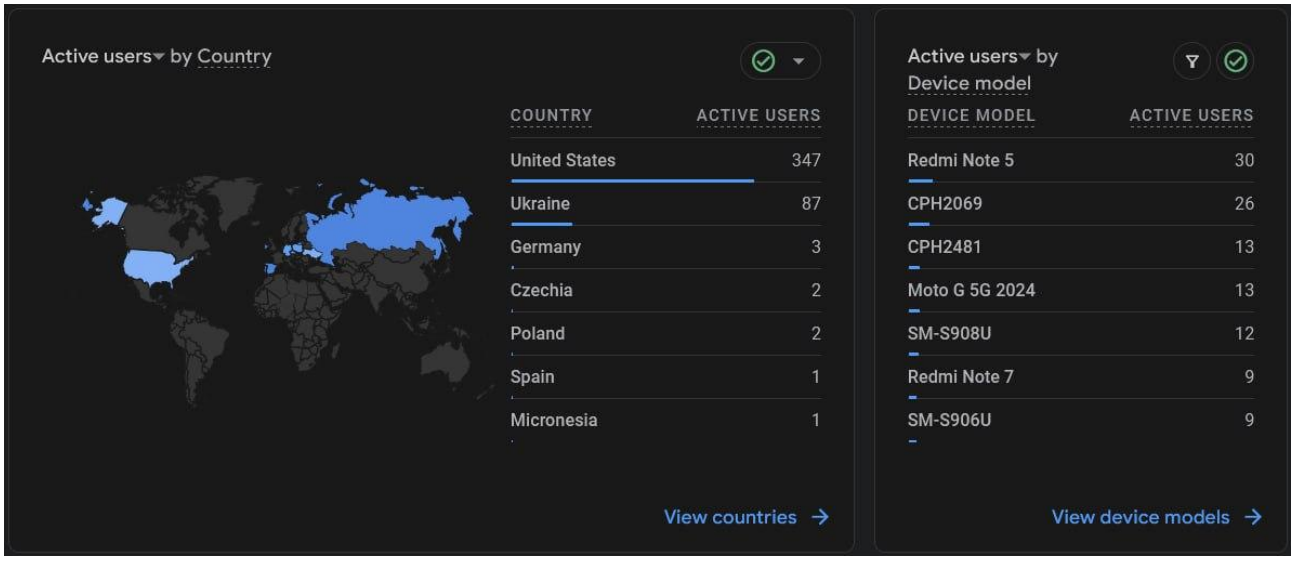


Рисунок 1.11 – **Розподіл активних користувачів**

1.11.6 Загальна інтерпретація та рекомендації

1. **Аудиторія та охоплення.**
   - **Сильний початковий інтерес:** понад 440 встановлень і 423 активні протягом 30 днів.
   - **Географічна концентрація:** основна частина аудиторії залишилась у США та Україні. Для розширення впливу варто розглянути локалізацію інших мов та більш точний таргетинг.
2. **Залученість (engagement) та активність.**
   - Максимальний середній час використання досягав півгодини — години, але в середньому склав 16–17 хв.
   - Невисока частота повторних сесій (≈ 2 на користувача за весь тест), що разом із низьким рівнем подій типу session_start і user_engagement вказує на слабкі тригери для повернення в додаток.
   - Середня кількість переглядів екранів (~ 3,8) і кількість завершених вузлів (~ 14,5) доводять, що найбільш зацікавлені користувачі проходили контент досить активно, але це було невелике відсоткове співвідношення від загальної кількості встановлень.
3. **Утримання користувачів.**
   - **Day 1 = 15,8 %** — критично низький показник утримання вже на наступний день.
   - Практично повне «вимивання» користувачів до Day 7.
   - Беручи до уваги ці дані, очевидно, що під час тестового періоду контент або функціонал додатку не був достатньо привабливим для того, щоб утримати користувачів мінімально необхідний термін.
4. **Технічні характеристики.**
   - Майже всі користувачі мобільні, при цьому помітна домінація Android-пристроїв середнього класу.
   - Серед найактивніших моделей: Redmi Note 5, CPH2069, Moto G (сезону 2024). Це означає, що оптимізація й тестування продуктивності варто зосередити саме на цих пристроях.

РОЗДІЛ 2 СПЕЦІАЛЬНИЙ

У цьому розділі ми деталізуємо, які саме події й показники потрібно фіксувати в кожній із трьох зон гри. Сформуємо логіку та розробимо систему дистанційного оновлення конфгів для балансу, сюжету тощо, щоб полегшити в майбутньому впровадження А/Б тестувань.

2.1 Система збору даних. Визначення подій (event taxonomy)

Для того, щоб отримати глибоке розуміння поведінки гравців у трьох зонах гри, необхідно чітко класифікувати всі події, які фіксуються під час гри. Запроваджена систему подієвого таксономіє сприятиме уніфікації збору даних, спростить їхню обробку та аналіз.

1. **Мікро-події**

   Ці події відображають найдрібніші взаємодії користувача з інтерфейсом та елементами гри. Вони допомагають зрозуміти, як гравець навігується грою та які екрани чи елементи привертають його увагу.

- **Приклади:**
  - tap_screen — торкання будь-якої точки екрана
  - open_menu — відкриття головного меню, картки будівлі, крамнички
  - view_popup — перегляд інформаційного вікна (підказка, туторіал, івент)
  - scroll_inventory — прокрутка списку інвентаря
- **Значення для аналізу:**
  - Виявлення «вузьких місць» інтерфейсу (UI/UX)
  - Оцінка активності в межах однієї сесії

2. **Макро-події**

   Це ключові бізнес- події, які безпосередньо впливають на ігровий процес і внутрішню економіку. Фіксація макро-подій критично важлива для обчислення основних ігрових метрик (DAU, retention, ARPU).

- **Приклади:**
  - plant_crop — посадка культури на городньому полі
  - harvest_crop — збір врожаю з грядки

- complete_order — виконання замовлення мешканця
- purchase_upgrade — покупка або апгрейд інструменту, фабрики, складу
- Частота перемикання між полями

- **Значення для аналізу:**
  - Розрахунок загального часу на макро-дозавдання
  - Оцінка конверсійних шляхів (funnels)
  - Виявлення механік із найвищим/найнижчим KPI

3. **Контекстні атрибути**

До кожної події додаються атрибути, що дозволяють аналізувати поведінку в різних умовах і сегментувати користувачів за ситуаційними характеристиками.

- **Основні атрибути:**
  - field_type — тип поля (город, фабрика, село)
  - player_level — рівень гравця на момент події
  - session_time_of_day — час доби (ранок, день, вечір, ніч)
  - event_active — позначка про наявність поточного івенту або сезонної події
  - device_type — платформа (iOS, Android)
- **Значення для аналізу:**
  - Порівняння активності в різних зонах гри
  - Виявлення кореляцій між рівнем гравця та вибором механік
  - Аналіз ефективності івентів у різні години

2.2 Інструменти й SDK

Для забезпечення надійного та гнучкого збору аналітичних даних у досліджуваній грі інтегровані дві основні платформи — Firebase Analytics і Facebook Analytics — з можливістю підключення додаткових сервісів (GameAnalytics, Amplitude) за потреби глибшого аналізу.

Firebase Analytics

Firebase Analytics використовується як базова платформа для збору як

стандартних, так і кастомних ігрових подій:

- **Реєстрація стандартних подій**
  - first_open — перше відкриття додатку після інсталяції
  - session_start — початок кожної ігрової сесії
  - level_up — досягнення нового рівня гравцем
- **Додавання власних подій**
  - plant_seed — посадка культури
  - harvest_crop — збір врожаю
  - factory_upgrade — покращення фабрики
    Кожна подія передає додаткові параметри (player_level, field_type, session_time_of_day), що дозволяє сегментувати аудиторію за контекстом.
- **Remote Config для A/B-тестування**

За допомогою Remote Config динамічно змінюються параметри таймерів дозрівання та інтенсивності замовлень у селищній зоні. Це дає змогу проводити A/B-тести без необхідності випуску нового клієнтського патчу.

  - **Facebook Analytics / Facebook Events**
  - Facebook Analytics (через Facebook SDK) доповнює Firebase даними про рекламні кампанії й соціальну взаємодію:
- **Трекінг переходів із рекламних оголошень**
  - fb_mobile_activate_app — активація додатку після кліку на рекламний банер
  - Purchase — реєстрація внутрішніх покупок, корелювання з джерелом трафіку
- **Створення аудиторій для ретаргетингу**

На основі подій plant_seed, complete_order та factory_upgrade формуються кастомні аудиторії в Ads Manager. Це дозволяє націлювати рекламу на гравців, які, наприклад, не завершили сезонний івент або мають низький Day-7 retention.

2.3 Розробка системи для Remote Config

У сучасних умовах розробки мобільних ігор надзвичайно важливо мати можливість оперативно коригувати ключові параметри геймплею (баланс економіки, умови онбордингу, налаштування мотиваційних механік тощо) без необхідності випускати нові збірки застосунку. Саме для цього реалізується система віддаленого конфігурування (Remote Config), яка дозволяє змінювати значення параметрів у реальному часі та розгортати А/Б-тестування без втручання в клієнтський код. Для початку розглянемо ключові системи, з якими буде працювати Remote Config.

2.3.1 Огляд системи предметів, та її реалізації

Усі об’єкти – від предметів та техніки до рослин, товарів, будівель і фабрик – реалізовано у вигляді конфігураційних ScriptableObject, що забезпечує зручне редагування й оновлення параметрів безпосередньо в редакторі Unity. Найбільш загальна база для таких елементів представлена класом BaseItem (Додаток Ж), який інкапсулює основні властивості: тип об’єкта (ItemType), контейнер, у якому він може зберігатися, рівень доступності для гравця, посилання на додаткові конфігурації, іконку та ключ локалізації. Такий підхід дозволяє уніфікувати механізм роботи з різними видами ігрових об’єктів і забезпечити швидке завантаження нових версій конфігів через Remote Config. У подальшій частині подано опис структури класу BaseItem та приклади його використання (код наведено в додатку), а також ілюстрацію того, як ці об’єкти відображаються в інспекторі Unity (рис. 2.1).

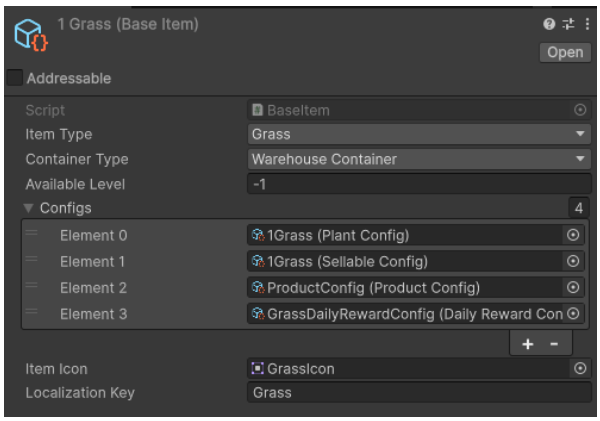


Рисунок 2.1 – Приклад конфігу предмету в Інспекторі Unity

Клас BaseItem – це ScriptableObject, який слугує базовою конфігурацією для будь-якого ігрового об’єкта (предмета, будівлі, рослини тощо). Головні принципи його роботи:

1. **Зберігання загальних властивостей**
    1. Поле _itemType — тип предмета (ItemType).
    2. containerType — контейнер, у якому його можна зберігати (ContainerType).
    3. availableLevel — рівень, з якого предмет доступний гравцю.
    4. _itemIcon — іконка предмета.
    5. localizationKey — ключ для підвантаження локалізованого імені/опису.
2. **Масив Configs і словник configsDict**

- У полі Configs зберігається масив об’єктів різних конфігурацій (нащадків Config), які визначають поведінку цього предмета (наприклад, SellableConfig, PlantConfig, FactoryConfig тощо).
- У методі Init() масив розбивається на словник configsDict, де ключем є ObjectType (наприклад, Plant, Factory, Tool тощо), а значенням – відповідний об’єкт Config. Це дозволяє швидко шукати потрібну конфігурацію за типом.

3. **Універсальний доступ до конкретного Config**

- Метод GetConfig<T>() дозволяє за допомогою дженерика отримати потрібну підконфігурацію.
- Усередині він перевіряє, якому типу T потрібно повернути конфіг (наприклад, якщо T = PlantConfig, то шукає ObjectType.Plant у словнику). Якщо конфіг не знайдено або тип не відповідає, виводиться Debug.LogError.

4. **Додаткові геттер-методи**

- GetAvailableLevel() – повертає availableLevel.
- GetItemIcon() – повертає _itemIcon.
- GetObjectTypes() – повертає перелік ключів (ObjectType) із словника configsDict, тобто всі типи конфігурацій, якими володіє цей предмет.

- GetItemType(), GetContainerType(), GetLocalizationKey() – повертають відповідні поля.
- IsObjectType(objectType) – перевіряє, чи міститься в словнику конфіг із заданим ObjectType.

Завдяки такій організації BaseItem уніфікує зберігання всіх основних даних про ігровий об’єкт і дозволяє динамічно додавати або змінювати різні поведінкові конфігурації через словник configsDict, що ідеально підходить для подальшого віддаленого оновлення через Remote Config.

На рисунку 1.1 наведено конфігурацію об’єкта «Трава» в інспекторі Unity. Зокрема:

- **Item Type: Grass** — цей предмет позначено як «Трава».
- **Container Type: Warehouse Container** — «Трава» може зберігатися на складі;
- **Available Level: –1** — доступна гравцю з самого початку (відсутнє обмеження за рівнем).

У блоці **Configs** прикріплено чотири різні конфігурації, що визначають властивості цієї «Трави»:

- **PlantConfig** («1Grass (Plant Config)») — описує всі ключові параметри як рослини: час росту, потребу у воді/добриві, ґрунтові обмеження тощо.
- **SellableConfig** («1Grass (Sellable Config)») — задає економічні характеристики предмета: базову ціну продажу, частоту запитів на ринку, розмір партії, у якій покупці можуть замовляти цю траву, та інші умови торгівлі.
- **ProductConfig** («ProductConfig (Product Config)») — сервісний конфіг, спільний для всіх продуктів; використовується, наприклад, для фільтрації чи групування товарів, але сам по собі не містить жодних специфічних даних.
- **DailyRewardConfig** («GrassDailyRewardConfig (Daily Reward Config)») — визначає, що «Трава» може випадати як щоденна нагорода, із налаштуваннями ймовірності та кількості одиниць за один розіграш.

У результаті такий набір налаштувань (itemType, containerType, availableLevel та набір конфігів) є базовим визначенням для всіх рослин у грі: кожен «Plant»-об’єкт містить аналогічні поля

У грі реалізовано велику кількість окремих конфігурацій, кожна з яких містить власний набір параметрів для налаштування поведінки ігрових елементів. Частина цих конфігів безпосередньо пов’язана з базовими об’єктами (BaseItem), але є й такі, що функціонують самостійно та відповідають за глобальні налаштування гри (наприклад, рівні гравця, параметри зон, системи винагород тощо).

Важливо зазначити, що для кожного конфігу створено окрему підсистему його обробки – від парсерів Google Sheets до модулів у клієнтському коді. Це полегшує розробку, оскільки нові типи конфігів можна підключати без значних змін у вже існуючій архітектурі, і одночасно дозволяє чітко розділити відповідальність кожного модуля.

Нижче наведено повний перелік наявних у проєкті конфігів:

- RegionConfig
- BuildingConfigs
- ResourceToDiamondConfig
- StartingResourceConfig
- BaseItemConfig
- ObstacleConfig
- WarehouseConfig
- WarehouseLevelConfig
- WareConfig
- ToolConfig
- GardenAreasConfig
- DailyRewardConfigs
- OrderTableAwardConfigs
- SellableConfigs

- PlantConfigs
- RecipeConfigs
- FactoryConfigs
- FarmerShopConfig
- CustomersConfig
- PlayerLevelsConfig

Завдяки такому розмаїттю та модульній структурі конфігів можна тонко коригувати майже всі аспекти балансу: від поведінки покупців і цін на різні зони в GardenField до системи досвіду, складу початкових ресурсів, механік крафту й щоденних бонусів. Крім того, централізоване зберігання налаштувань у вигляді Google-таблиць і динамічне завантаження через Remote Config забезпечують гнучкість і швидкість впровадження змін під час А/Б-тестування без потреби випуску нових збірок.

2.3.2 Огляд системи сюжету та квестів

У проєкті система сюжету й квестів побудована за аналогією з системою предметів: замість BaseItem центральним елементом є клас PlotNode (Додаток И), а всі налаштування сюжету зосереджені в одному конфігу AllPlotNode. Кожна окрема нода (PlotNode) містить базове визначення свого стану та поведінки й керується двома ключовими списками:

1. **TriggerInfos** — набір тригерів, які стежать за певними умовами (досягненням гравцем рівня, виконанням попереднього квесту, потребою поливу/підживлення клітини тощо). Як тільки всі з тригерів спрацьовують, нода запускається.
2. **Actions** — послідовність дій, які виконує ця нода. Основними діями є ShowDialogAction та WaitCompleteQuestAction, які слугують для показу діалогів, або початку та очікуванню завершення квесту. Окрім універсальних дій, є спеціалізовані для проведення туторіалу для гравця.

У грі не надто багато типів тригерів, але кожен із них чітко відповідає за активацію відповідної ноди сюжету. Наприклад:

- **CellNeedWateringTrigger** та **CellNeedFertilizerTrigger** використовуються в навчальному блоці, щоб підказати гравцю догляд за рослиною;
- **CompleteNodeTrigger** чекає на завершення попереднього квесту й активує наступний етап сюжету;

Таким чином, коли спрацьовують всі тригери, PlotNode по черзі «програє» всі Actions — від показу діалогу до очікування завершення завдання чи натискання кнопки. Після проходження всіх Actions ця нода вважається завершеною, і до роботи готується наступний PlotNode згідно з конфігом AllPlotNode. Такий підхід дозволяє динамічно змінювати сюжетні ланцюжки через конфігурації, та визначати коли треба почати виконувати ту чи іншу ноду. Як приклад: туторіал поливу (рис. 2.2) покажеться тоді, коли є грядка, яку треба полити.

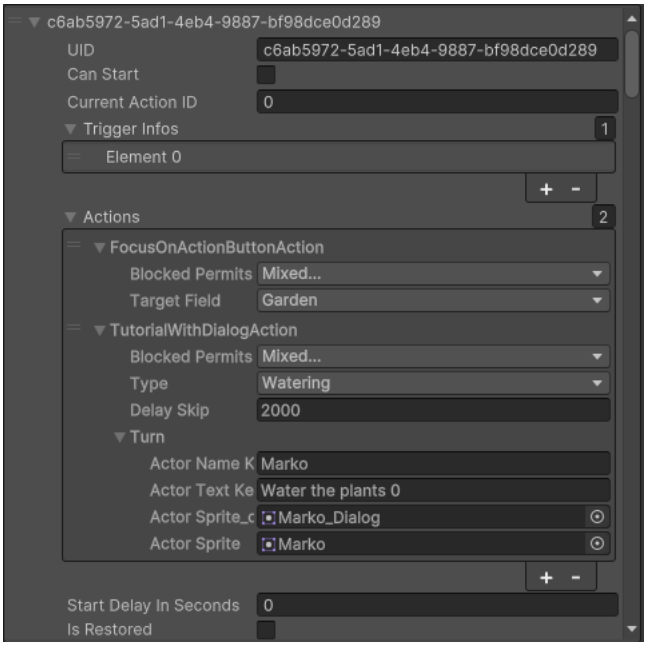


Рисунок 2.2 – Конфіг туторіалу поливу в Інспекторі Unity.

А керує всіма нодами – StoryAndQuestSystem, ініціалізуючи всі ноди зі збережень, за допомогою uid, та запускаючи активні ноди (Додаток К.)

2.3.3 Програмна реалізація Remote Config

У нашому проєкті Remote Config написано мовою C#, оскільки ігровий клієнт базується на Unity та використовує логіку, реалізовану в .NET-середовищі. Основна ідея полягає в тому, щоб зберігати всі налаштування у

вигляді Google-таблиць, до яких система звертається через API Google Sheets під час завантаження гри (якщо є оновлення в таблицях). Використовуючи REST-запити, парсери отримують «сирі» дані з захищених таблиць, перетворюють їх у відповідні об’єкти конфігурації й зберігають у локальному кеші. Такий підхід гарантує динамічність і гнучкість: при потребі можна додати нові колонки з параметрами, змінити структуру таблиці або розширити функціональність парсерів без суттєвих змін у вже написаному коді. У подальшому розділі наведено опис архітектури системи, огляд ключових класів та методів, продемонстровано приклади роботи з API Google Sheets й організації парсерів, а також показано, як отримані дані інтегруються в механізм Remote Config всередині клієнтського застосунку.

У наведених схемах (рис, рис, рис) зображено, як у нашому проєкті побудовано архітектуру завантаження та розбору віддалених конфігурацій через Google Sheets. Нижче подано пояснення головних компонентів і порядку їхнього взаємозв’язку.

2.3.4 Загальна ідея та етапи завантаження конфігів

Перевірка увімкненого віддаленого завантаження

При запуску гри, система спочатку перевіряє, чи дозволено віддалене завантаження (Remote Config). Якщо ця опція вимкнена, жодної додаткової роботи не відбувається, і застосунок працює зі вбудованими налаштуваннями (локальними ScriptableObject). Але якщо Remote Config увімкнено, під час ініціалізації викликається один метод: GoogleSheetProvider.LoadAndParseAllMainParsers(...)

Саме він виступає «точкою входу» для завантаження абсолютно всіх віддалених конфігів.

GoogleSheetProvider виступає як центральний «диригент» парсерів

Клас GoogleSheetProvider — це ScriptableObject у Unity (рис. 2.3), який:

- Знає про всі основні парсери, що реалізують інтерфейс IMainGoogleSheetParser.

- Утримує в інспекторі Unity словник (SerializedDictionary<string, string> spreadsheets), де кожному названню класу-парсера (ключ — parserType.Name) відповідає рядок із ID або URL відповідної Google-таблиці.
- За допомогою рефлексії (метод GetParsersAssembly()) знаходить усі класи, що реалізують IMainGoogleSheetParser, і автоматично додає їхні імена до словника spreadsheets, якщо раніше їх там не було.

Рисунок 2.3 – Файл конфігурації парсерів GoogleSheetProvider в Інспекторі Unity

Програмна реалізація представлена в додатку Л.

Послідовне завантаження «головних» парсерів

Коли гравець заходить у гру, у методі LoadAndParseAllMainParsers(loadCallback) відбуваються такі кроки:

1. Через GetParsersAssembly() збирається список усіх Type parserType, які є некласами IMainGoogleSheetParser.
2. Визначається totalParser = кількість знайдених parserType, а currentParser початково = 0.
3. У циклі для кожного parserType викликається LoadAndParseMainParser(parserType, loadCallback). Після завершення одного парсера currentParser збільшується на 1, і через loadCallback можна оновити UI/лог.

Внутрішня робота GoogleSheetProvider

При першому виклику (або під час оновлення об’єкта в редакторі) метод OnValidate() викликає рефлексію

- **Концепція «Assembly Definition»**: Поле selectedAssemblyDefinition (string) містить ім’я збірки, де лежать усі наші класи-парсери.
- Реалізувавши пошук цих типів, ми автоматично бачимо у інспекторі Unity всі «ключі» (імена класів), яким залишилося лише підставити конкретний Google-Sheet ID у відповідному полі словника spreadsheets.

Метод LoadAndParseAllMainParsers (...)

Цей метод відповідає за запуск парсингу всіх головних парсерів та їхніх залежностей. Алгоритм виского рівню буде представлений у додатку М

2.3.5 Архітектурна схема взаємодії компонентів

Нижче наведена спрощена діаграма компонентів (рис 2.4):

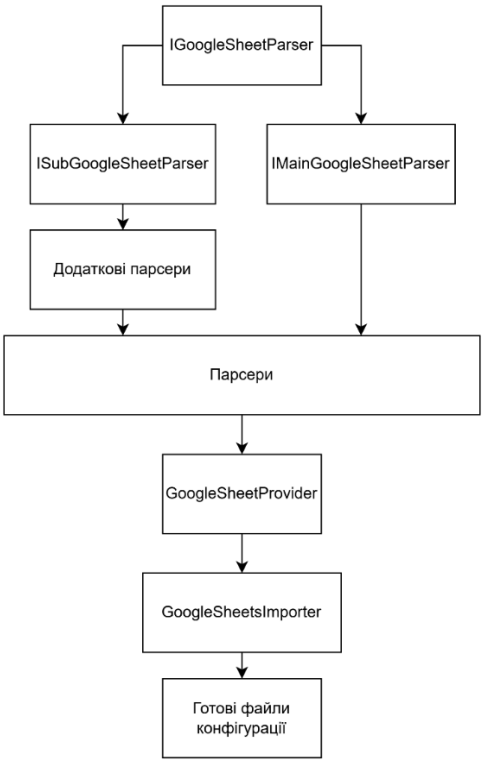


Рисунок 2.4 – Схематична взаємодія в системі.

На схемі показано взаємодію ключових компонентів, які разом реалізовують механізм віддаленого завантаження та розбору конфігурацій із Google Sheets. Принцип роботи можна розбити на кілька рівнів:

1. Інтерфейсні шари парсерів

1.1. IGoogleSheetParser

Це загальний інтерфейс (або «контракт») для усіх парсерів, що працюють із таблицями Google. Він не містить жодної реалізації, а лише декларує методи для зчитування діапазону (GetRange()), обробки даних (Parse(...)) та опціонально — для передачі залежностей.

1.2. IMainGoogleSheetParser

Спадкоємець IGoogleSheetParser, який відповідає за розбір «основних» листів: наприклад, повний перелік PlayerLevels, BaseItemConfig, WarehouseConfig тощо. Клас, що імплементує цей інтерфейс, містить методи:

1.2.1. GetRange() – повернути назву листа (або діапазону клітинок) у Google Sheets.

1.2.2. GetDependencies() – за потреби повернути список типів (Type[]), які представлятимуть допоміжні (підлеглі) парсери.

1.2.3. SetDependencies(Dictionary<Type, ISubGoogleSheetParser> deps) – отримати вже розпарсені результати з усіх залежностей і далі використовувати їх під час обробки головного листа.

1.2.4. Parse(DataTable rows) (позначено через базовий контракт) – отримати «сирі» рядки і з них побудувати внутрішні об’єкти конфігурації.

1.3. ISubGoogleSheetParser

Також спадкоємець IGoogleSheetParser, але «міні»-версія для допоміжних таблиць (наприклад, LocalizationKeysParser, TranslationsParser, RegionDependenciesParser тощо). Відрізняється від головних тим, що:

1.3.1. Не містить у собі GetDependencies() (або повертає null/пустий масив).

1.3.2. Використовується виключно для первинного зчитування частини даних (наприклад, список ключів, перелік категорій, основні xml-перетини), які потрібно передати у головний парсер через SetDependencies(...).

2. Рівень «Додаткові парсери»

Будь–який клас, що реалізує ISubGoogleSheetParser, автоматично потрапляє в підрядок обробки перш ніж запуститься відповідний IMainGoogleSheetParser. Це може бути **DialogParser** (зчитує діалоги та ключі до них) для StoryAndQuestParser.

У діаграмі всі такі класи об’єднані блоком «Додаткові парсери».

3. Рівень «Парсери»

Після того, як усі залежності (підпарсери) розібрались — кожен у своєму порядку — настає черга основних парсерів (IMainGoogleSheetParser). У схемі їх об’єднано в загальний прямокутник «Парсери». Саме вони отримують результати від підлеглих парсерів, обробляють головні листи, створюють кінцеві об’єкти конфігурацій ScriptableObject і передають їх далі.

4. GoogleSheetProvider

Цей скриптабл-об’єкт у Unity виступає єдиним «оркестратором» або «диспетчером» роботи всіх парсерів. Його основні обов’язки:

4.1. За допомогою рефлексії (метод GetParsersAssembly()) знаходить у вказаній збірці (selectedAssemblyDefinition) усі класи, що реалізують IMainGoogleSheetParser.

4.2. Автоматично заповнює (у методі OnValidate) словник spreadsheets, якщо в інспекторі Unity з'явилися нові парсери без заданих ID/URL таблиці.

Послідовно викликає:

4.2.1. Спочатку — усі класи з ISubGoogleSheetParser, якщо вони є у списку залежностей у конкретного головного парсера.

4.2.2. Потім — сам IMainGoogleSheetParser, передаючи йому зібрані раніше «підпорядковані» результати через SetDependencies(...).

4.3. Виконує асинхронні виклики кожного парсера (через await LoadAndParseMainParser(...)), щоб не блокувати потік Unity.

4.4. Завдяки цьому «теразуванню» GoogleSheetProvider може в одному місці запустити одночасно всі необхідні парсери, відстежувати їхні результати та виводити інформацію про хід процесу (через loadCallback).

5. GoogleSheetsImporter

Нижче за логікою йде компонент, який безпосередньо звертається до Google API:

5.1. Отримує на вхід об'єкти (парсери) та дані для аутентифікації (credentials.json, що міститься у TextAsset).

5.2. За допомогою REST-запитів до Google Sheets API завантажує «сирі» значення (табличні рядки).

5.3. Передає їх у метод parser.Parse(...):

5.3.1. Для підпарсерів — виклик відбувається у загальному порядку, визначеному методом GetDependencies().

5.3.2. Для головних парсерів — після встановлення залежностей (через SetDependencies(...)) вони отримують уже готову «підмножину» даних і можуть сформувати фінальні структури конфігів.

6. Готові файли конфігурації

Після того, як кожен основний парсер (та його підлеглі) обробили свої таблиці, результатом роботи системи з’являються локальні об’єкти конфігурацій - Unity-скриптабл-об’єкти (.asset), які легко розгорнути під час гри;

Ці «готові файли» вже можна безпосередньо використовувати в коді гри (наприклад, звернутися до PlantConfigs.All, WarehouseLevelConfig.Levels чи PlayerLevelsConfig.ExperienceThresholds).

2.3.6 Загальний вигляд парсерів та таблиць

Система парсерів покладається на GoogleSheets. Ключовими елементами таблиці є Перший рядок з вказаного діапазону, який виступає в якості заголовків до змінних а всі подальші рядки – значення цих змінних для кожного елементу. Хоча в проекті існує велика кількість різних конфігів, для прикладу було обрано BaseItem (рис. 2.5) та Node (StoryAndQuestSystem) (рис. 2.6), які виглядають наступним чином:

	A	B	C	D	E
1	ConfigName	Type	Container Type	Available Level	Localization Key
2		Coin	ValuteContainer	0	
3		Diamond	ValuteContainer	0	
4		Score	LevelContainer	0	
5		EnergyDrink10	EnergyContainer	0	
6		EnergyDrink25	EnergyContainer	0	
7		EnergyDrink50	EnergyContainer	0	
8	Warehouse 0	Warehouse	NullContainer	0	Warehouse
9	Warehouse 1	Warehouse	NullContainer	4	Warehouse
10	Warehouse 2	Warehouse	NullContainer	9	Warehouse
11		ToolShovel	NullContainer	0	Showel
12		ToolGrinder	NullContainer	8	Grinder
13		ToolTractor	NullContainer	24	Tractor
14		ToolSickle	NullContainer	0	Sickle
15		ToolMotoblock	NullContainer	8	Motoblock
16		ToolCombine	NullContainer	15	Combine

Рисунок 2.5 – Таблиця BaseItemConfig

	A	B	C	D	E	F	G	H	I
1	UID	startDelay	Ir Triggers	Actions	QuestTask	QuestDescriptionKey	QuestAward	ActorQuestIcon	
2	c6ab5972-5ad1-4eb4-9887-bf98dce0d289		CellNeedWateringTrigger	FocusOnActionButtonAction Garden					
3				TutorialWithDialogAction typeTutorial = Watering delaySkip = 2000 DialogTurn = c47e6d63-800d-42cd-bf19-83fbb9c652a0					Тутор поливу
4	fa5790cb-2012-4c72-9d4f-ed504121acc7		CellNeedFertilizerTrigger	FocusOnActionButtonAction Garden					
5				TutorialWithDialogAction typeTutorial = Digging delaySkip = 2000 DialogTurn = 42f866cc-520d-4b54-8d6b-99f569da46c8					Тутор перекопки
6	9c348c97-ac1d-4c74-8e2f-5f18d2b44cd7			FocusOnActionButtonAction Garden					
7				ShowDialogAction DialogTurns: f5829bd5-a140-4ee3-aad0-16065d1d5683 5fb87be6-d945-4caf-a27b-b6370fd43c53 3b826b02-209f-489e-972e-fcf8868753fd 93e8a7fc-a695-47b0-810b-20ddd3ad1c94 69d6a920-780a-42a4-9f6f-5be45a17e99a					
8	96db817f-46e5-4865-bd37-4e7c94d90567	0,5	CompleteNodeTrigger 9c348c97-ac1d-4c74-8e2f-5f18d2b44cd7	FocusOnActionButtonAction Garden					
9				TutorialWithDialogAction typeTutorial = ClearObstacle delaySkip = 2000 DialogTurn = 4e6eacde-1b23-4294-9d9f-56755db36776					Тутор прибирання перешкод
10				ClearObstacleTutorialAction areaID = 0 sectionID = 0					

Рисунок 2.6 – Таблиця Node

Безпосередньо розбір (парсинг) будь-якої Google-таблиці (зокрема, таких, як BaseItemConfig чи NodeConfig) відбувається за допомогою класу

GoogleSheetsImporter. Суть полягає в тому, що перший рядок листа вважається «заголовком» (headers), тобто списком імен полів, а всі наступні рядки обробляються послідовно «зліва направо» та «зверху вниз»: спочатку значення із клітинок другого рядка, потім третього і так далі. Програмна реалізація GoogleSheetsImporter представлена в Додатку Н (Метод DownloadAndParseSheet було доповнено коментарями).

Покроковий опис алгоритму

1. **Перевірка діапазону (range)**

Якщо рядок range (назва листа або діапазон у форматі "Sheet1!A1:E100") пустий або null, метод одразу повертається, записуючи попередження в консоль. Це запобігає випадковому спробі зробити запит без точної адреси.

2. **Підготовка парсера**

Метод parser.LoadTargetConfigs() викликається на самому початку, щоб дати парсеру шанс підготувати власні колекції або створити порожні об’єкти перед тим, як почати вносити в них значення. Наприклад, BaseItemConfigParser може виділити пам’ять для списку всіх BaseItemConfig або створити порожній словник, у який потім додаватиме нові елементи.

3. **Формування URL для запиту**

Ми використовуємо константу GoogleSheetsApiUrl, у яку підставляємо:

- ID Google-таблиці (_sheetId)
- назву діапазону/аркуша (range)

Потім додаємо ?key= + наш API_KEY (або власні облікові дані), щоб аутентифікувати запит. У результаті отримаємо щось на кшталт:

- https://sheets.googleapis.com/v4/spreadsheets/1A2B3C4D5E/values/BaseItemConfig!A:E?key=MY_API_KEY

4. **Асинхронне виконання HTTP-запиту**

Використовуємо Unity-клас UnityWebRequest.Get(url) для отримання JSON-відповіді. Оскільки це асинхронний запит, ми не блокуємо головний потік Unity – решта гри залишається чутливою до подій в UI.

5. **Обробка відповіді**

- Якщо **успішно** (request.result == UnityWebRequest.Result.Success), спочатку парсимо рядок JSON у JObject.
- Далі беремо поле "values" (масив рядків і стовпців, List<List<object>>), саме воно містить усі комірки у вигляді вкладених списків.
- Перевіряємо, чи в масиві tableArray є хоч один рядок (тобто справді існують дані). Якщо ні – логуємо попередження.

6. **Визначення заголовків (headers)**
   - Перший рядок tableArray[0] вважаємо списком назв полів.
7. **Прохід по всім рядкам даних**
   - Ми починаємо з i=1, бо i=0 — це заголовки.
8. Збереження результатів (лише в Unity Editor)
9. Після того, як усі рядки було оброблено методом parser.Parse(...), у блоці #if UNITY_EDITOR виконується збереження змін в конфігах. Це означає, що якщо парсер десь викликав ScriptableObject.CreateInstance<BaseItemConfig>() чи оновив якийсь .asset у папці, то ці зміни відразу зберігаються на диск – і розробник бачить нові або оновлені об’єкти відразу після завершення процесу парсингу.
10. В додатку П та додатку Р буде представлено код реалізації парсерів для BaseItemConfig а також PlotParser в якості прикладів.

ВИСНОВКИ

Виконане дослідження демонструє, що для успішної розробки та підтримки гри жанру «ферма» критично важливим є комплексний підхід до аналізу залученості й утримання гравців. Поєднання якісних і кількісних методів, застосованих у роботі, дало змогу виокремити ключові аспекти, що впливають на поведінку аудиторії протягом усього життєвого циклу гри. У підсумковому висновку можна виокремити кілька основних думок, які узагальнюють отримані результати та дають напрямок для подальших покращень.

По-перше, аналіз ігрових механік і структури інтерфейсів показав, що у розглянутому проєкті існує невідповідність візуальних елементів між окремими зонами гри. Хоча кожна з трьох тематичних ділянок (городнє поле, індустріальне поле, селищна зона) має власну функціональну спрямованість і певну естетику, відсутність єдиного стилістичного підходу призводить до дисонансу в сприйнятті. Наприклад, інтерфейси першого поля виглядають завершеними й витриманими у яскравих тонах, тоді як друге поле демонструє «заглушки» без остаточного оформлення, а третє поле, хоча і є деталізованішим, усе одно не синхронізоване з основним HUD. Така фрагментація створює відчуття недопрацювань і знижує довіру користувачів, що в умовах високої конкуренції мобільних проектів жанру «ферма» може призвести до раннього відтоку аудиторії. З огляду на це рекомендується уніфікувати кольорову палітру, іконографію та підходи до розміщення елементів інтерфейсу, аби кожен перехід між полями сприймався гравцем як частина єдиного цілісного світу.

По-друге, аналіз ігрового балансу виявив фундаментальні невідповідності, що формують «витки» ресурсного дефіциту. Основна валюта (монети) отримується лише за підвищення рівня або продаж продукції в крамничці, тоді як виконання замовлень — одного з ключових завдань у селищній зоні не дає монет, а лише матеріали для відбудови будівель. Така модель змушує гравця постійно перемикатись між накопиченням коштів для покращення ферми та змогою вирощувати рослини та потребою виконувати сюжетні й тематичні

завдання, що породжує замкнене коло: без монет неможливо купувати нові рослини, грядки, заводи, а без цього стає неможливо виконувати сюжетні завдання, чи замовлення, щоб відбудовувати село чи піднімати рівень. Додатково проблема накопичення зайвих ресурсів, які через обмеження складу не можуть бути своєчасно утилізовані чи продані, створює відчуття застою. Це має прямий негативний вплив на мотивацію гравця й може стати однією з причин низького коефіцієнта утримання. Зважаючи на це, доцільно впровадити додаткові джерела надходжень монет — зокрема, винагороду за виконання замовлень у селищній зоні, а також розробити механізми обміну або продажу надлишкових матеріалів за вигіднішими умовами. Подібні зміни сприятимуть збалансованому економічному циклу, у якому гравець не стикатиметься з тупиковими ситуаціями.

По-третє, онбординг — процес першого знайомства новачка з грою — виявився доволі добре структурованим та сюжетно обґрунтованим, що є позитивним фактором. Наприклад, вступна сцена з братами створює емоційний зв’язок і пояснює базові механіки через геймплей. Проте статистика тестового запуску показала, що лише близько 38 % гравців доходять до фіналу навчального циклу: найбільший відтік відбувається на етапах, пов’язаних із будівництвом фабрик і переробкою ресурсів. Це свідчить про недостатню інтуїтивність інтерфейсу або про складність самих завдань. Крім того, повідомлення гравців про помилки в квесті з продажем ресурсів вказує на необхідність ретельного тестування критичних механік ще до публічного релізу. Щоб підвищити коефіцієнт завершення онбордингу, слід оптимізувати навчальні підказки, знизити кількість одночасно вводимих нових систем та додати можливість негайного отримання монет за виконанні початкові завдання. Це допоможе гравцю відчути прогрес уже в перші хвилини та зменшить відсів у процесі знайомства.

Четвертий напрямок висновків стосується базових метрик залученості й утримання, отриманих у тестовому запуску. Дані телеметрії свідчать, що, незважаючи на успішний початковий «приманковий» ефект (442 першочергових

відкриття, 5174 події проходження вузлів), лише 15,8 % користувачів повернулися на другий день, а до сьомого дня майже всі залишилися. Середній час взаємодії в пікові дні сягав 50–55 хвилин, але вже до середини травня падіння цього показника призвело до середньої активності лише 16 хвилин 40 секунд. Такі низькі показники Day 1 і Day 7 retention вказують на слабкі стимули для повторного залучення: у грі бракує регулярних нагадувань (push-повідомлень), ціннісних винагород за повернення та короткочасних івентів, які змушують зайти знову. Відповідно, розробникам варто впровадити систему щоденних бонусів, сезонних подій з обмеженим терміном ігрового контенту, а також налаштувати таргетовані повідомлення, які нагадуватимуть гравцю про незавершені завдання та можливість отримати цінні винагороди. Це допоможе згладити «провал» між Day 1 і Day 7 і збільшить кількість активних сесій.

П’ятий висновок стосується загальної архітектури збору та обробки даних. У роботі запропоновано використовувати Firebase Analytics разом із Facebook Analytics для відстеження як стандартних (first_open, session_start, level_up), так і кастомних подій (plant_seed, harvest_crop, factory_upgrade тощо). Запроваджена подієва таксономія дозволяє деталізовано аналізувати шлях гравця в усіх трьох зонах. Однак, щоб отримати найбільш достовірні дані, потрібно налаштувати коректну передачу контекстних атрибутів (field_type, player_level, session_time_of_day, event_active, device_type). Це дозволить сегментувати аудиторію за різними характеристиками і виявити специфічні потреби новачків і ветеранів. Збір «сирих» даних є лише початковим етапом: надалі слід імплементувати A/B тестування, для якого було розроблено систему віддаленого редагування конігів та почати проводити регулярне A/B-тестування змін механік і візуальних елементів, що дозволить оцінювати вплив нововведень у реальному часі без необхідності випускати новий патч.

Таким чином, комплексний висновок доводить: для підвищення конкурентоспроможності мобільного проєкту «ферма» необхідно одночасно працювати над єдиним візуальним стилем, збалансувати внутрішню економіку, оптимізувати процес онбордингу, упровадити систему мотиваційних нагород і

вдосконалити збір та аналіз даних. Лише так можна домогтися стійкого підвищення показників залученості й утримання, а відтак — комерційного успіху гри. У майбутньому рекомендується регулярно аналізувати дані телеметрії, проводити глибинні опитування гравців і тестувати зміни в ігровому середовищі, щоб динамічно адаптувати продукт до потреб цільової аудиторії та тенденцій ринку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Apple Developer. Onboarding for Games. URL: https://developer.apple.com/app-store/onboarding-for-games/
2. BBC GEL. How to design onboarding for games. URL: https://www.bbc.com/gel/features/how-to-design-onboarding-for-games (дата звернення: 2025-05-19).
3. ConversionBet. Importance of Smooth Onboarding Process for Player Retention. URL: https://conversionbet.com/importance-of-smooth-onboarding-process-player-retention/
4. IndieKlem. 7 Ways How to Keep Your Interface Consistent. URL: https://indieklem.com/7-how-to-keep-your-interface-consistent
5. Argentics. How Vital Is Well-Designed UI for a Positive Game User Experience. URL: https://www.argentics.io/how-vital-is-well-designed-ui-for-a-positive-game-user-experience
6. Хом'як Т.В., П’ятоволенко О.О. Аналіз показників гри жанру «пошук предметів» та оптимізація ігрових характеристик для підвищення платежів // Моделювання та прогнозування економічних процесів [Текст]: Матеріали XVII Всеукраїнської науково-практичної конференції з міжнародною участю. – К.: КПІ ім. Ігоря Сікорського, 2023. - с.126-129.
7. Кваліфікаційна робота бакалавра [Електронний ресурс] : методичні рекомендації для здобувачів ступеня бакалавра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / уклад.: Т.А. Желдак, Т.В. Хом’як, А.В. Малієнко ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 32 с.

ДОДАТКИ

Додаток А. Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кількість аркушів	Примітки
1				
2		Документація		
3				
4	САУ.КР.25.06.ПЗ	Пояснювальна записка	58	Формат А4
5				
6		Демонстраційний матеріал	23	Презентація на CD-R
7				
8		Копія роботи	1	Диск CD-R
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

					САУ.КР.25.06.ДА.ПЗ.
Змін.	Аркуш	№ докум.	Підпис	Дата	

Розроб.	Вернигора О. О.			**Матеріали кваліфікаційної роботи**	Літ.	Аркуш	Аркушів
К. розд.	Хом’як Т. В.						
Керівн.	Хом’як Т. В.				НТУ «ДП», 12; 124-21-1		
Н.контр.	Хом’як Т. В.						
Зав. каф.	Желдак. Т. А.						

Додаток Б. Відгук керівника кваліфікаційної роботи

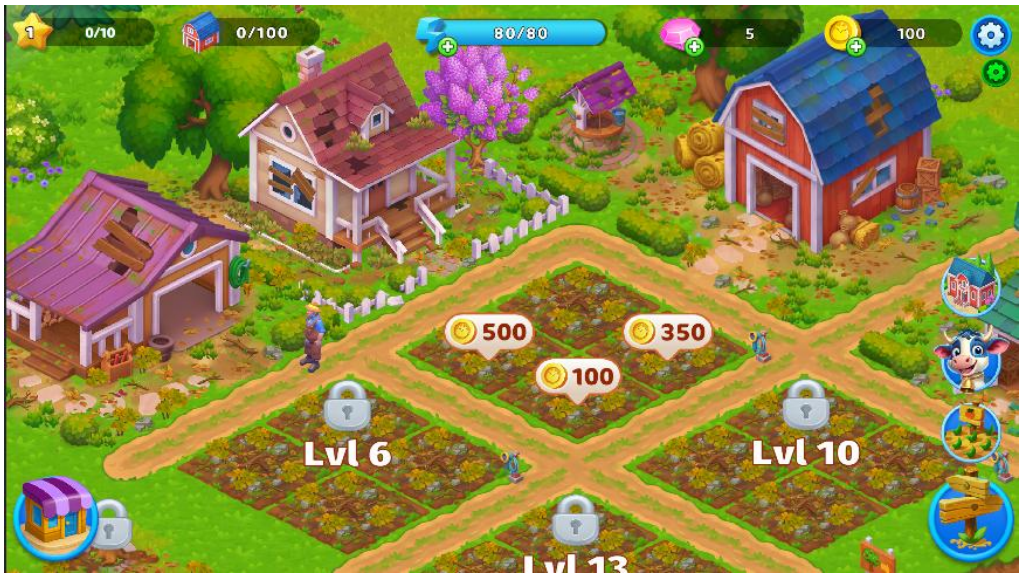
Відгук
на кваліфікаційну роботу бакалавра
здобувача вищої освіти групи *124 – ХХ – Х*
спеціальності *124 Системний аналіз*

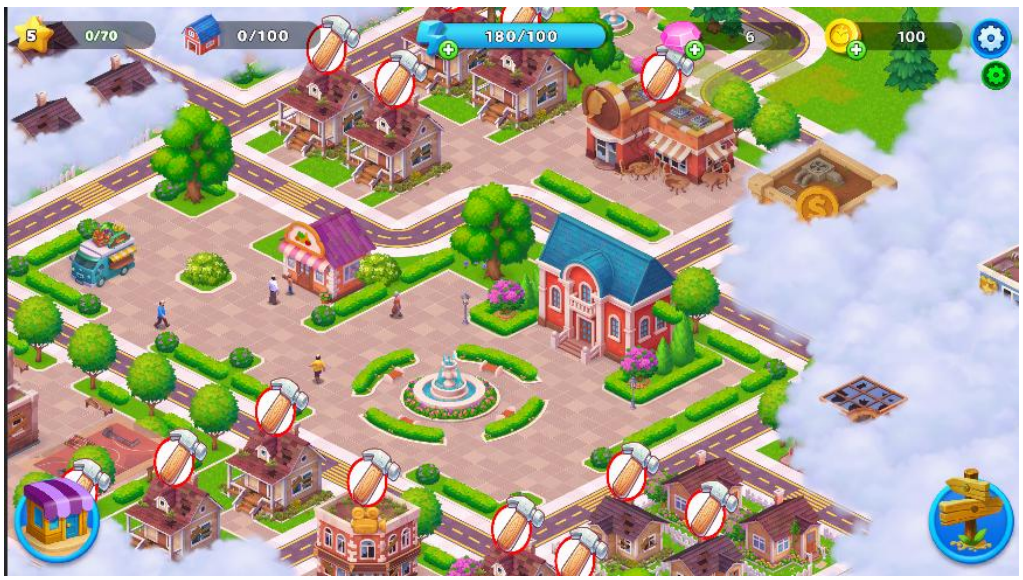
Тема кваліфікаційної роботи: «Аналіз метрик залученості та утримання гравців у грі жанру 'ферма': визначення ключових показників для прийняття рішень»
Обсяг кваліфікаційної роботи 85 стор.
Мета кваліфікаційної роботи: аналіз метрик залученості та утримання гравців у грі жанру «ферма» з визначенням ключових показників, що забезпечують своєчасне й обґрунтоване прийняття рішень для оптимізації геймплею та розробка системи віддаленого редагування конфігів для подальшого впровадження A/B тестувань.
Актуальність теми проведення дослідження сприятиме формуванню методології аналізу метрик у грі жанру «ферма» та підвищенню якості продукту, над яким працює наша команда.
Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 124 Системний аналіз, оскільки передбачає використання методів системного аналізу, обробки даних та розробку програмного забезпечення, яке сприятиме подальшому впровадженню A/B тестування.
Виконані в кваліфікаційній роботі завдання відповідають вимогам ступеня бакалавра. Оригінальність наукових рішень полягає в розробці системи віддаленого редагування конфігів та яка сприяє подальшому аналізу та покарщенню комерційної успішності проекту, а простота використання, дозволяє легко розширювати, та впроваджувати цю систему в інші проекти, які написані мовою C#
Практичне значення результатів кваліфікаційної роботи полягає в тому, що отримані дані та результати використовуються на реальному проєкті, а викладені рекомендації будуть застосовані при подальшому оновлені гри
Висновки підтверджують можливість використання результатів роботи в покращені цільового продукту, забезпечуючи комерційний успіх проекту.
Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі (*в разі невідповідності – вказати*)
У роботі відзначено такі недоліки: мала вибірка даних для аналізу під час відкритого тестування та відсутність детальнішого пояснення програмного коду. Недостатній аналіз наукових публікацій та закордонного досвіду щодо досліджуваного питання.
Кваліфікаційна робота в цілому заслуговує оцінки: «добре»

З урахуванням висловлених зауважень автор заслуговує присвоєння кваліфікації «бакалавр з системного аналізу».

Керівник кваліфікаційної роботи бакалавра,
К. ф.-м.наук доцент _____ / Хом’як Т. В.

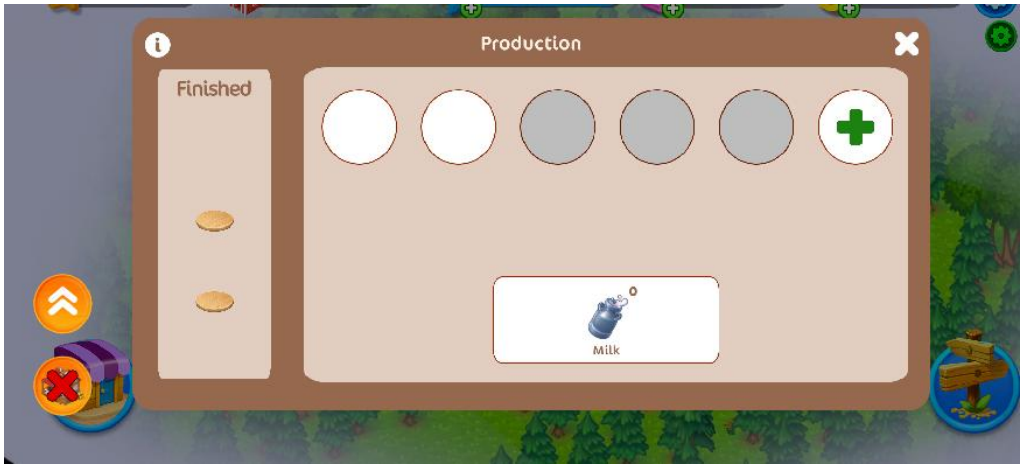
Додаток В. Візульний стиль полів.





Додаток Г. Основні графічні інтерфейси першого поля

Додаток Д. Основні графічні інтерфейси другого поля




Upgrades
1 ST
Storage 2
Recipe 1
Finished 2
2 ST
Storage 3
Recipe 1
Finished 3
Necessary materials:
100/1K
Upgrade


Destroy factory
Do you want to destroy factory?
Accept

Додаток Е. Основні графічні інтерфейси третього поля

Додаток Ж. Програмний код класу BaseItem

```
using System.Collections.Generic;
using System.Linq;
using App.Common.Items.Warehouse;
using App.Fields.BuildingsField.Factories;
using App.Fields.GardenField.Buildings.Warehouse_.Scripts;
using UnityEngine;
using App.Fields.MarketField.OrderSystem.Scripts;
using Assets.App.Common.Items;
using Assets.App.Fields.GardenField.Energy.EnergyDrinks;

[CreateAssetMenu(fileName = "NewItem", menuName = "ItemSystem/ItemBase")]
public class BaseItem : ScriptableObject, IItem
{
    [SerializeField]
    protected ItemType _itemType;
    public ContainerType containerType;
    public int availableLevel;
    public Config[] Configs;
    [HideInInspector]
    public Dictionary<ObjectType, Config> configsDict;
    [SerializeField]
    Sprite _itemIcon;
    public string localizationKey;

    public void Init()
    {
        configsDict = new Dictionary<ObjectType, Config>();
        if (Configs == null) return;
        foreach (var item in Configs)
        {
            configsDict.Add(item.Type, item);
        }
    }

    public virtual T GetConfig<T>() where T : Config
    {
        try
        {
            if (typeof(T) == typeof(SellableConfig))
                return configsDict[ObjectType.Sellable] as T;
            if (typeof(T) == typeof(UpgradableConfig))
                return configsDict[ObjectType.Upgradable] as T;
            if (typeof(T) == typeof(FactoryConfig))
                return configsDict[ObjectType.Factory] as T;
            if (typeof(T) == typeof(PlantConfig))
                return configsDict[ObjectType.Plant] as T;
```

```
        if(typeof(T) == typeof(ToolConfig))
            return configsDict[ObjectType.Tool] as T;
        if (typeof(T) == typeof(WarehouseConfig))
            return configsDict[ObjectType.Warehouse] as T;
        if (typeof(T) == typeof(EnergyDrinkConfig))
            return configsDict[ObjectType.EnergyDrink] as T;
        if (typeof(T) == typeof(GrowthBoosterConfig))
            return configsDict[ObjectType.GrowthBooster] as T;
        if(typeof(T) == typeof(OrderTableAwardConfig))
            return configsDict[ObjectType.OrderTableAward] as T;
        if (typeof(T) == typeof(ProductConfig))
            return configsDict[ObjectType.Product] as T;
        if (typeof(T) == typeof(BoosterConfig))
            return configsDict[ObjectType.Booster] as T;
        if (typeof(T) == typeof(DecoreConfig))
            return configsDict[ObjectType.Decore] as T;
        if (typeof(T) == typeof(MaterialConfig))
            return configsDict[ObjectType.Material] as T;
        if (typeof(T) == typeof(DailyRewardConfig))
            return configsDict[ObjectType.DailyReward] as T;
        if (typeof(T) == typeof(ObstacleConfig))
            return configsDict[ObjectType.ObstacleConfig] as T;
    }
    catch
    {
        Debug.LogError($"{name} not contains config {typeof(T).Name}");
    }
    return null;
}
public virtual int GetAvailableLevel()
{
    return availableLevel;
}

public virtual Sprite GetItemIcon()
{
    return _itemIcon;
}

public virtual ObjectType[] GetObjectTypes()
{
    return configsDict.Keys.ToArray();
}
public virtual ItemType GetItemType()
{
    return _itemType;
}
```

```
    public virtual ContainerType GetContainerType()
    {
        return containerType;
    }
    public virtual bool IsObjectType(ObjectType objectType)
    {
        return configsDict.ContainsKey(objectType);
    }
    public virtual string GetLocalizationKey()
    {
        return localizationKey;
    }
}
```

Додаток И. Програмна реалізація PlotNode

```
[Serializable]
public class PlotNode
{
    public string UID;
    [HideInInspector]
    public UnityEvent OnCompleted;
    [HideInInspector]
    public UnityEvent<PlotNode> OnAllTriggersActive;
    public bool CanStart;

    public int currentActionID;

    [JsonIgnore]
    [SerializeReference]
    public List<TriggerInfo> triggerInfos;

    [JsonIgnore]
    [SerializeField]
    public List<ITrigger> triggers;

    [SerializeReference]
    public List<Action> actions;


    [NonSerialized]
    private bool _completed = false;
    [HideInInspector]
    public bool Completed
    {
        get => _completed;
        private set
        {
            _completed = value;
        }
    }

    public DateTime timeActivationNode;
    public float startDelayInSeconds;
    public bool isRestored;

    public void SetActive(bool active, TriggerFactory triggerFactory)
    {
        OnCompleted = new UnityEvent();
        OnAllTriggersActive = new UnityEvent<PlotNode>();
        triggers = new();
        if (active)
```

```
    {
        CanStart = true;
        if (triggerInfos == null || triggerInfos.Count == 0) return;
        foreach (TriggerInfo info in triggerInfos)
        {
            var trigger = triggerFactory.GetTrigger(info);
            trigger.Init(info);
            trigger.Activate();
            trigger.OnTriggered.AddListener(OnTriggerEnableHandler);
            triggers.Add(trigger);
        }
        OnTriggerEnableHandler();
    }
    else
    {
        CanStart = false;
        Completed = true;
    }
}

public void Complete()
{
    Completed = true;
    OnCompleted?.Invoke();
}

private void OnTriggerEnableHandler()
{
    foreach (var trigger in triggers)
    {
        if (trigger.Enabled == false)
        {
            CanStart = false;
            return;
        }
    }
    CanStart = true;
    if (CanStart)
    {
        OnAllTriggersActive?.Invoke(this);
    }
}

public virtual bool RestoreFromSave(PlotNode node)
{
    timeActivationNode = node.timeActivationNode;
    isRestored = true;
```

```
        return true;
    }

    internal virtual void Reset()
    {
        CanStart = false;
        currentActionID = 0;
        _completed = false;
        isRestored = false;
        foreach (var action in actions)
        {
            action.Reset();
        }
    }

    public override bool Equals(object obj)
    {
        if (obj is PlotNode node)
        {
            return UID == node.UID;
        }
        return false;
    }
}
```

Додаток К. Програмна реалізація ініціалізації StoryAndQuestSystem

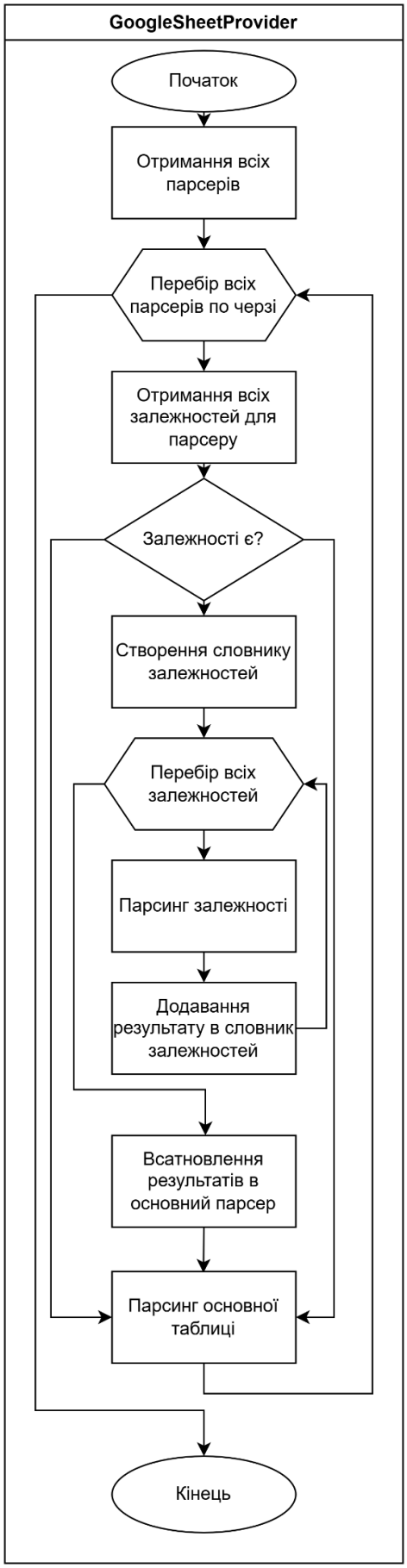
```
public void Init(GameData gameData)
{
    _allNodes = _assetProvider.LoadPlotLine().allNodes;
    gameData.plotData ??= new PlotData();
    _data = gameData.plotData;
    _data.completedNodes ??= new List<string>();
    _data.activeNodes ??= new List<PlotNode>();

    _complitedNodes ??= new List<PlotNode>();
    _uncomplitedNodes ??= new List<PlotNode>();
    _activeNodes ??= new List<PlotNode>();

    for (int i = 0; i < _allNodes.Count; i++)
    {
        var node = _allNodes[i];
        node.CanStart = false;
        var savedActiveNode = _data.activeNodes.Find(x => x.UID.Equals(node.UID));
        if (savedActiveNode != null)
        {
            node.RestoreFromSave(savedActiveNode);
            node.SetActive(true, _triggerFactory);
            node.OnAllTriggersActive.AddListener(OnNodeAllTriggersActiveHandler);
            ActivateNode(node, OnNodePlayCompleteHandler);
        }
        else if (_data.completedNodes.Contains(_allNodes[i].UID))
        {
            _complitedNodes.Add(_allNodes[i]);
            node.SetActive(false, _triggerFactory);
        }
        else
        {
            _uncomplitedNodes.Add(_allNodes[i]);
            node.SetActive(true, _triggerFactory);
            node.OnAllTriggersActive.AddListener(OnNodeAllTriggersActiveHandler);
        }
    }
    GetAvailableNodes();
}
```

Додаток Л. Програмна реалізація GoogleSheetProvider

```
using AYellowpaper.SerializedCollections;
using GoogleImporter.Parser;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Threading.Tasks;
using UnityEngine;

namespace GoogleImporter
{
    [CreateAssetMenu(fileName = "GoogleSheetProvider", menuName = "GoogleSheetImporte/Provider", order = 0)]
    public class GoogleSheetProvider: ScriptableObject
    {
        public TextAsset credentials;
        public string selectedAssemblyDefinition;
        [SerializedDictionary(keyName: "Type", valueName: "Spreadsheets")]
        public SerializedDictionary<string, string> spreadsheets;

        private int totalParser = 0;
        private int currentParser = 0;

        private void OnValidate()
        {
            var parsersType = GetParsersAssembly()
                .GetTypes()
                .Where(t => typeof(IMainGoogleSheetParser).IsAssignableFrom(t) && t.IsClass && !t.IsAbstract)
                .ToList();

            for (var i = 0; i < parsersType.Count; i++)
            {
                if (spreadsheets.ContainsKey(parsersType[i].Name) == false)
                {
                    spreadsheets.Add(parsersType[i].Name, "");
                }
            }
        }

        public async Task LoadAndParseMainParser(Type parserType, Action<float, string> loadCallback)
        {
            Debug.Log("Load and parse " + parserType);
            loadCallback?.Invoke((float)currentParser/totalParser, $"{parserType.Name} {currentParser}/{totalParser}");
            var parser = (IMainGoogleSheetParser)Activator.CreateInstance(parserType);
            var sheetsImporter = new GoogleSheetsImporter(credentials.text, spreadsheets[parserType.Name]);
            var dependencyTypes = parser.GetDependencies();
            if (dependencyTypes == null)
```

```
            {
                await sheetsImporter.DownloadAndParseSheet(parser.GetRange(), parser);
                return;
            }

            Dictionary<Type, ISubGoogleSheetParser> dependencies = new Dictionary<Type, ISubGoogleSheetParser>();
            foreach (var dependencyType in dependencyTypes)
            {
                var subParser = (ISubGoogleSheetParser)Activator.CreateInstance(dependencyType);
                Debug.Log("Parse sub " + subParser.GetType().Name);
                await sheetsImporter.DownloadAndParseSheet(subParser.GetRange(), subParser);
                dependencies.Add(dependencyType, subParser);
            }
            parser.SetDependencies(dependencies);
            await sheetsImporter.DownloadAndParseSheet(parser.GetRange(), parser);
        }

        public Assembly GetParsersAssembly()
        {
            return AppDomain.CurrentDomain.GetAssemblies()
                .FirstOrDefault(assembly => assembly.GetName().Name == selectedAssemblyDefinition);
        }
        public async Task LoadAndParseAllMainParsers(Action<float, string> loadCallback)
        {
            var parsersType = GetParsersAssembly()
                .GetTypes()
                .Where(t => typeof(IMainGoogleSheetParser).IsAssignableFrom(t) && t.IsClass && !t.IsAbstract)
                .ToList();

            totalParser = parsersType.Count;
            currentParser = 0;

            foreach (var parserType in parsersType)
            {
                await LoadAndParseMainParser(parserType, loadCallback);
                currentParser++;
            }
        }
    }
}
```

Додаток М. Схематичний алгоритм парсингу всіх конфігів



Додаток Н. Програмна реалізація GoogleSheetsImporter

```
using Google.Apis.Auth.OAuth2;
using Google.Apis.Services;
using Google.Apis.Sheets.v4;
using System.Collections.Generic;
using System;
using System.Threading.Tasks;
using UnityEngine;
using GoogleImporter.Parser;
using UnityEngine.Networking;
using Newtonsoft.Json.Linq;


#if UNITY_EDITOR
using UnityEditor;
#endif

namespace GoogleImporter
{
   public class GoogleSheetsImporter
   {
      private const string GoogleSheetsApiUrl = "https://sheets.googleapis.com/v4/spreadsheets/{0}/values/{1}";
      private string _apiKey = "API_KEY";
      private readonly string _sheetId;
      private readonly SheetsService _service;

      private List<string> _headers = new();

      public GoogleSheetsImporter(string credentialsJson, string sheetId)
      {
         _sheetId = sheetId;

         GoogleCredential credential;
         credential = GoogleCredential.FromJson(credentialsJson)
            .CreateScoped(SheetsService.Scope.Spreadsheets);

         _service = new SheetsService(new BaseClientService.Initializer()
         {
            HttpClientInitializer = credential
         });
      }

      public async Task DownloadAndParseSheet(string range, IGoogleSheetParser parser)
     {
        // Якщо не задано жодного діапазону (назви листа), просто попереджаємо і виходимо
        if (string.IsNullOrEmpty(range))
        {
```

```
    Debug.LogWarning("Range is null or empty.");
    return;
}

// Перш ніж завантажити самі комірки, даємо парсеру можливість підготуватися:
parser.LoadTargetConfigs();
Debug.Log($"Starting downloading sheet ({range})...");
// Формуємо URL запиту до Google Sheets API:
// _sheetId – це ідентифікатор Google-таблиці (наприклад, "1A2B3C4D5E...")
// _apiKey  – API-ключ, необхідний для доступу до таблиці (у нашому випадку — Service Account)
string url = string.Format(GoogleSheetsApiUrl, _sheetId, range);
url += "?key=" + _apiKey;
// Асинхронно відправляємо HTTP GET-запит UnityWebRequest
using (UnityWebRequest request = UnityWebRequest.Get(url))
{
    await request.SendWebRequest();
    if (request.result == UnityWebRequest.Result.Success)
    {
        try
        {
            // 1. Парсимо отриманий JSON у об’єкт JObject
            JObject response = JObject.Parse(request.downloadHandler.text);
            // 2. Беремо масив рядків (values) – це List<List<object>>
            //    Якщо у таблиці є, наприклад, 30 строк і 5 стовпців,
            //    то tableArray.Count == 30, а tableArray[i].Count == 5
            var tableArray = response["values"].ToObject<List<List<object>>>();
            Debug.Log($"Sheet downloaded successfully: {range}.");

            if (tableArray != null && tableArray.Count > 0)
            {
                // 3. Перший рядок tableArray[0] вважаємо заголовками (headers)
                var firstRow = tableArray[0];
                List<string> headers = new List<string>();
                foreach (var cell in firstRow)
                {
                    headers.Add(cell.ToString());
                }
                // Наприклад, якщо це лист BaseItemConfig, то headers може бути:
                // [ "ConfigName", "Type", "Container Type", "Available Level", "Localization Key" ]
                //
                // Якщо це лист NodeConfig (сюжетних нод), то headers може бути:
                // [ "UID", "startDelayInSeconds", "Triggers", "Actions", "QuestTask", ... ]

                // 4. Дізнаємося, скільки всього рядків (rowsCount)
                int rowsCount = tableArray.Count;

                // 5. Проходимо по всіх рядках, починаючи з i = 1 (тобто другого рядка:
```

```
                //    перший – це заголовки, далі йдуть власне дані)
                for (int i = 1; i < rowsCount; i++)
                {
                    var row = tableArray[i];
                    int rowLength = row.Count;

                    // 6. Для кожної комірки у поточному рядку (j = 0..rowLength-1)
                    for (int j = 0; j < rowLength; j++)
                    {
                        var cell = row[j];          // Значення з Google Sheets
                        var header = headers[j];    // Назва колонки (header), наприклад: "Type" або "Actions"

                        // 7. Передаємо пару (header, cellValue) у універсальний метод парсера
                        //    Сам інтерфейс IGoogleSheetParser має метод Parse(string fieldName, string rawValue),
                        //    який знає, як перетворити рядкове значення rawValue у потрібне поле своєї конфіг-моделі.
                        parser.Parse(header, cell.ToString());
                    }
                    // Після того, як ми викликали parser.Parse(...) для всіх колонок поточного рядка,
                    // об'єкт-парсер (наприклад, BaseItemConfigParser або NodeConfigParser)
                    // вже «на місці» зберіг усі значення полів для цього одного рядка.
                    // Зазвичай кожен виклик parser.Parse(header, rawValue) просто записує
                    // значення rawValue у локальне поле або властивість, згідно з логікою самого парсера.
                    // А після обробки всього рядка (усіх j-клітинок) парсер «закріплює»
                    // отримані дані в колекцію або робить додаткові перевірки валідності.
                }
#if UNITY_EDITOR
                // Якщо ми перебуваємо в редакторі Unity, зберігаємо усі ScriptableObject-об'єкти,
                // які могли створитися або оновитися в процесі парсингу.
                AssetDatabase.SaveAssets();
                Debug.Log("Sheet parse successfully");
#endif
            }
            else
                Debug.LogWarning("No data found in Google Sheets.");
        }
        catch (Exception ex)
        {
            Debug.LogError($"Error parsing the Google Sheets response: {ex.Message}");
        }
    }
    else
        Debug.LogError($"Error retrieving Google Sheets data: {request.error}");
  }
 }
}
```

Додаток П. Програмна реалізація BaseItemConfigParser

```
using Assets.App.Common.Items;
using System;
using System.Collections.Generic;
using System.Linq;
#if UNITY_EDITOR
using UnityEditor;
#endif
using UnityEngine;

namespace GoogleImporter.Parser.Configs
{
    public class BaseItemConfigParser: IMainGoogleSheetParser
    {
        private List<BaseItem> _items;
        private BaseItem _currentItem;
        bool useConfigName = false;

        public string GetRange()
        {
            return "BaseItemConfigs!A1:D";
        }

        public List<Type> GetDependencies() => null;

        public void SetDependencies(Dictionary<Type, ISubGoogleSheetParser> dependecies) { }

        public void Parse(string header, string token)
        {
            switch (header)
            {
                case "ConfigName":
                    if (string.IsNullOrEmpty(token))
                    {
                        useConfigName = false;
                        return;
                    }

                    _currentItem = _items.FirstOrDefault(item =>
                    {
                        var name = item.ToString().Split(" ", StringSplitOptions.None);
                        var finallString = name[0];
                        for (int i = 1; i < name.Length - 1; i++)
                        {
                            finallString += $" {name[i]}";
                        }
                        return finallString == token;
```

```
                });
                useConfigName = true;
                break;

            case "Type":

                if (useConfigName) return;
                if (Enum.TryParse(token, out ItemType itemType))
                {
                    _currentItem = _items.Find(x => x.GetItemType() == itemType);
                }
                else
                {
                    throw new System.Exception("Invalid item type");
                }
                break;
            case "Container Type":
                ContainerType containerType = (ContainerType)Enum.Parse(typeof(ContainerType), token);
                _currentItem.containerType = containerType;
                break;
            case "Available Level":
                _currentItem.availableLevel = int.Parse(token)-1;
                break;
            case "Localization Key":
                _currentItem.localizationKey = token;
                break;
            default:
                return;
        }
#if UNITY_EDITOR
        EditorUtility.SetDirty(_currentItem);
#endif
    }

    public void LoadTargetConfigs()
    {
        ItemsAssetProvider assetProvider = new ItemsAssetProvider();
        _items = assetProvider.LoadAllItems();
    }
  }
}
```

Додаток Р. Програмна реалізація PlotParser

```
using System;
using System.Collections.Generic;
#if UNITY_EDITOR
using UnityEditor;
#endif
using UnityEngine;

namespace GoogleImporter.Parser.Plot
{
    public partial class PlotParser : IMainGoogleSheetParser
    {
        private string ALL_PLOT_NODES_PATH = "StoryAndQuests/Nodes/AllNodes";
        private AllPlotNodes _allPlotNodes;
        private PlotNode _currentNode;
        public DialogTurnsParser turnsParser;

        partial void InitializeParserPlotActions();
        partial void InitializeParserPlotTriggers();
        partial void InitializeParserPlotQuest();

        public string GetRange()
        {
            return "NodeFinaly!A1:H";
        }

        public List<Type> GetDependencies()
        {
            return new List<Type>() { typeof(DialogTurnsParser) };
        }

        public void SetDependencies(Dictionary<Type, ISubGoogleSheetParser> dependecies)
        {
            turnsParser = dependecies[typeof(DialogTurnsParser)] as DialogTurnsParser;
        }

        public void LoadTargetConfigs()
        {
            _allPlotNodes = Resources.Load<AllPlotNodes>(ALL_PLOT_NODES_PATH);
            _allPlotNodes.allNodes = new();
            InitializeParserPlotActions();
            InitializeParserPlotTriggers();
            InitializeParserPlotQuest();
        }

        public void Parse(string header, string token)
        {
```

```
            switch (header)
            {
                case "UID":
                    if (string.IsNullOrEmpty(token)) return;
                    _currentNode = new PlotNode();
                    _currentNode.UID = token;
                    _currentNode.triggerInfos = new();
                    _currentNode.actions = new();
                    _allPlotNodes.allNodes.Add(_currentNode);
                    _currentNode.startDelayInSeconds = 0;
                    break;
                case "startDelayInSeconds":
                    if (string.IsNullOrEmpty(token) == false)
                        _currentNode.startDelayInSeconds = float.Parse(token);
                    break;
                case "Triggers":
                    if (string.IsNullOrEmpty(token)) return;
                    ParseTriggers(token);
                    break;
                case "Actions":
                    if (string.IsNullOrEmpty(token)) return;
                    ParseActions(token);
                    break;
                case "QuestTask":
                    if (string.IsNullOrEmpty(token)) return;
                    ConvertPlotNodeToQuestNode();
                    ParseQuestTasks(token);
                    break;
                case "QuestDescriptionKey":
                    if (string.IsNullOrEmpty(token)) return;
                    if (_currentNode is Quest)
                        ParseQuestDescription(token);
                    break;
                case "QuestAward":
                    if (string.IsNullOrEmpty(token)) return;
                    if (_currentNode is Quest)
                        ParseQuestAward(token);
                    break;
                case "ActorQuestIcon":
                    if (string.IsNullOrEmpty(token)) return;
                    if (_currentNode is Quest)
                        ParseActorTaskIcon(token);
                    break;
                default:
                    return;
            }
#if UNITY_EDITOR
```

```
            EditorUtility.SetDirty(_allPlotNodes);
#endif
        }

        private void ConvertPlotNodeToQuestNode()
        {
            if (_currentNode is PlotNode plotNode)
            {
                var questNode = new Quest
                {
                    UID = plotNode.UID,
                    triggerInfos = plotNode.triggerInfos,
                    actions = plotNode.actions,
                    info = new QuestInfo()
                    {
                        tasks = new(),
                        description = "",
                        award = new()
                    }
                };
                _currentNode = questNode;

                int index = _allPlotNodes.allNodes.IndexOf(plotNode);
                if (index >= 0)
                {
                    _allPlotNodes.allNodes[index] = questNode;
                }
            }
            else
            {
                Debug.LogError("currentNode not is PlotNode");
            }
        }
    }
}
```