Міністерство освіти і науки України

Український державний університет науки і технологій

Навчально-науковий інститут

«Український державний хіміко-технологічний університет»

(назва навчально-наукового інституту)

Факультет комп'ютерних наук та інженерії

(повна назва факультету)

Кафедра спеціалізованих комп'ютерних систем

(повна назва кафедри)

Пояснювальна записка

до дипломної роботи

бакалавр

(освітній рівень)

на тему «Аналіз ефективності ігрових механік за допомогою математичного моделювання»

Виконав: студент 4 курсу, групи 4-КІ

спеціальності

123 Комп'ютерна інженерія

(код і назва спеціальності)

Студент КОЛІСНІКОВА О. О. (підпис)

(прізвище та ініціали)

Керівник ДУБОВИК Т. М. (підпис)

(прізвище та ініціали)

Рецензент ХОРОШИЛОВ С. В. (підпис)

(прізвище та ініціали)

Дніпро - 2026 рік

Український державний університет науки і технологій

(повне найменування вищого навчального закладу)

Навчально-науковий інститут

«Український державний хіміко-технологічний університет»

(назва навчально-наукового інституту)

Факультет, відділення комп'ютерних наук та інженерії

Кафедра спеціалізованих комп'ютерних систем

Освітній рівень бакалавр

Спеціальність 123 Комп'ютерна інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ

В. о. зав. Кафедри

канд. техн. наук, доц. Ясінчук І. Л.

«12» 06 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Колеснікова Олександра Олександрівна

(прізвище, ім'я, по батькові)

1. Тема роботи «Аналіз ефективності ігрових механік за допомогою математичного моделювання»

керівник роботи Дубовик Тетяна Миколаївна, старший викладач

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 02.03.2026 р. № 439-ст

2. Строк подання студентом роботи 16.06.2026 р.

3. Вихідні дані до роботи Аналіз системи гри Genshin Impact, дослідження механізмів pity та 50/50, створення програмної системи з використанням Spring Boot, MongoDB та Docker.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Загальний розділ. Спеціальний розділ. Охорона праці та безпека в надзвичайних ситуаціях. Організаційно-економічний розділ. Висновок. Список використаних літературних джерел. Додатки.

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

Презентація MS PowerPoint

6. Консультанти розділів роботи

Розділ	Призвіще, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Охорона праці та безпека в надзвичайних ситуаціях	Дубовик Тетяна Миколаївна	19.02.2026	10.05.26
Організаційно-економічний розділ	Дубовик Тетяна Миколаївна	20.02.2026	10.05.26
Консультант	Романчук Олександр Олександрович	20.02.2026	10.05.2026

7. Дата видачі завдання 14 лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Бібліотечний огляд за темою	Лютий	виконано
2	Робота над загальним розділом	Березень	виконано
3	Робота над спеціальним розділом	Березень	виконано
4	Робота над розділом ОП та БНС	Квітень	виконано
5	Робота над ОЕ розділом	Квітень	виконано
6	Створення презентації	Травень	виконано
7	Предзахист	Травень	виконано

Студент _____ КОЛЄСНІКОВА О. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ ДУБОВИК Т. М.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить: сторінок, рисунків, таблиць, джерел.

Об’єкт дослідження – гача-система у грі Genshin Impact та програмна система аналізу статистики круток.

Мета роботи – дослідження механізмів pity, soft pity та 50/50 у грі Genshin Impact, побудова математичної моделі гача-системи та створення програмної системи аналізу статистики з використанням Spring Boot, MongoDB та Docker.

У загальному розділі наведено теоретичні відомості щодо принципів роботи гача-систем у сучасних відеоіграх, розглянуто механізми pity, soft pity та 50/50, а також особливості гача-системи гри Genshin Impact.

У практичному розділі проведено математичний та статистичний аналіз механізмів випадіння персонажів, виконано дослідження особистої статистики круток та розглянуто принципи реалізації програмної системи аналізу даних.

У розділі програмної реалізації описано використання Spring Boot для створення серверної частини застосунку, MongoDB для зберігання статистичних даних та Docker для контейнеризації й розгортання системи.

Результати, отримані під час виконання дипломної роботи, можуть бути використані для подальшого дослідження гача-механік, аналізу статистики випадкових винагород та створення систем симуляції круток у відеоіграх.

Актуальність теми роботи обумовлена популярністю гача-ігор та необхідністю дослідження механізмів випадкових винагород, які використовуються в сучасних F2P-проєктах.

Ключові слова: GENSHIN IMPACT, ГАЧА-СИСТЕМА, PITY, SOFT PITY, 50/50, SPRING BOOT, MONGODB, DOCKER, REST API, JSON, СТАТИСТИЧНИЙ АНАЛІЗ.

ЗМІСТ

ВСТУП

У сучасному світі відеоігри стали невід'ємною частиною повсякденного життя великої кількості людей. Ігрова індустрія активно розвивається, а сучасні проєкти вже давно перестали бути лише способом проведення вільного часу. Сьогодні відеоігри поєднують у собі елементи програмування, математичного моделювання, економічних систем та аналізу поведінки користувачів. Особливої популярності набули F2P (Free-to-Play) ігри, у яких користувач може безкоштовно отримати доступ до основного функціоналу гри, а система монетизації реалізується через внутрішньоігрові покупки та механізми випадкових винагород.

Одним із найпоширеніших елементів сучасних F2P-ігор є гача-система. Принцип її роботи базується на випадковому отриманні персонажів, предметів або інших внутрішньоігрових нагород за допомогою спеціальних круток або випадкових випадінь. Подібні механізми активно використовуються в багатьох сучасних іграх та поєднують у собі елементи ймовірності, статистики та психології користувача.

Особливістю гача-систем є наявність механізмів pity, soft pity та гарантій, які впливають на ймовірність отримання рідкісних нагород. З одного боку, такі системи створюють елемент випадковості та підтримують інтерес гравців, а з іншого – формують складні математичні моделі, які можуть бути досліджені за допомогою статистичного аналізу та симуляцій.

Одним із найвідоміших представників сучасних гача-ігор є Genshin Impact, розроблена компанією Hoyoverse. Дана гра стала популярною завдяки поєднанню відкритого світу, сюжетної складової та складної системи отримання персонажів через гача-механіку. Саме Genshin Impact є зручним прикладом для проведення дослідження, оскільки в грі реалізовано декілька механізмів гарантії, soft pity та систему 50/50, що дозволяє виконувати математичний та статистичний аналіз роботи банерів.

Актуальність теми дипломної роботи полягає у зростаючій популярності гача-ігор та необхідності дослідження механізмів випадкових винагород, які використовуються у сучасних F2P-проєктах. Незважаючи на значну популярність подібних систем, принципи їхньої роботи часто залишаються недостатньо зрозумілими для більшості користувачів. Аналіз гача-механіки дозволяє оцінити вплив випадковості, гарантій та статистичних закономірностей на процес отримання персонажів.

У межах даної дипломної роботи розглядаються основні принципи роботи гача-систем, особливості банерів у грі Genshin Impact, механізми pity, soft pity та 50/50, а також виконується математичне моделювання процесу отримання персонажів. Окрему увагу приділено статистичному аналізу особистої історії круток автора роботи, що дозволяє порівняти теоретичні моделі з реальними результатами

Крім математичного аналізу, у роботі розглядається програмна реалізація системи аналізу статистики круток. Для створення серверної частини застосунку використовується Spring Boot, для зберігання статистичних даних – MongoDB, а для контейнеризації та розгортання системи – Docker. Використання сучасних технологій дозволяє створити гнучку систему для зберігання, обробки та аналізу статистичних даних.

Об’єктом дослідження є гача-система гри Genshin Impact та процес отримання персонажів за допомогою внутрішньоігрових механізмів випадкових винагород.

Предметом дослідження є механізми pity, soft pity, 50/50, статистичні закономірності роботи банерів та програмні засоби аналізу статистики круток.

Метою дипломної роботи є дослідження гача-системи гри Genshin Impact, побудова математичної моделі роботи банерів, проведення статистичного аналізу результатів круток та розробка програмної системи для аналізу гача-механік із використанням Spring Boot, MongoDB та Docker.

Для досягнення поставленої мети необхідно виконати такі завдання:

- дослідити принципи роботи гача-систем у сучасних відеоіграх;
- проаналізувати механізми pity, soft pity та 50/50 у грі Genshin Impact;
- побудувати математичну модель отримання 5★ персонажів;
- провести статистичний аналіз особистої історії круток;
- дослідити можливості Spring Boot, MongoDB та Docker у межах програмної системи;
- розглянути принципи реалізації серверної частини застосунку та зберігання статистичних даних.

Практична цінність роботи полягає у можливості використання отриманих результатів для подальшого аналізу гача-механік, створення систем симуляції круток та дослідження механізмів випадкових винагород у сучасних відеоіграх.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Гача-система у грі Genshin Impact

Гача-система (від японського терміну «gachapon») – це механізм отримання випадкових віртуальних предметів, персонажів та/або інших ресурсів за допомогою внутрішньоігрової валюти. Принцип її функціонування базується на ідеї японських автоматів з капсулами-сюрпризами, де покупець заздалегідь не знає, який саме предмет йому дістанеться.

У цифровому середовищі гача-механіка є алгоритмічно реалізована система випадкового вибору з наперед визначеним набором можливих результатів і встановленими ймовірностями випадіння. Основою її роботи є генератор випадкових чисел (Random Number Generator, RNG), який визначає результат кожної спроби («крутки»).

Структура гача-системи включає:

- об’єкти різної рідкості (наприклад рідкісні, епічні, легендарні або срібні, золоті, алмазні).
- зафіксовані базові ймовірності отримання кожної категорії.
- систему гарантій, яка підвищує шанс або забезпечує гарантовану винагороду після певної кількості невдалих спроб.
- обмежені за часом можливості отримання лімітних нагород.

На відміну від класичних систем випадкових винагород, гача-механіка часто містить додаткові алгоритмічні умови, що змінюють розподіл імовірностей залежно від історії попередніх спроб гравця. Таким чином, вона не є простою незалежною випадковою подією, а перетворюється на стохастичний процес з пам’яттю стану.

У сучасних F2P іграх гача-система грає дуже важливу економічну функцію. Вона є ключовим інструментом монетизації, оскільки мотивую

гравців здійснювати повторні спроби для отримання бажаної винагороди. Таке поєднання випадковості, обмеженості в часі й гарантованих механізмів формує складну модель поведінкової взаємодії між алгоритмом та користувачем.

З точки зору математичного аналізу гача-механіку можна розглядати як імовірнісну систему, що включає дискретні події, умовні переходи між станами, а також специфічні обмеження, які модифікують традиційний біноміальний розподіл. Така структура забезпечує зручність у формалізації та подальшому моделюванні за допомогою інструментів програмної інженерії.

Одним із ключових елементів гача-систем є використання генератора випадкових чисел (Random Number Generator він ж RNG). У комп'ютерних програмах RNG являє собою алгоритм, який генерує послідовність чисел, що імітує випадковість. Саме на основі цих чисел визначається результат кожної окремої спроби отримання винагороди.

У більшості відеоігор використовуються псевдовипадкові генератори чисел (PRNG), які генерують послідовність значень, спираючись на заданий початковий параметр (seed). Незважаючи на алгоритмічну природу, такі генератори забезпечують достатній рівень статистичної випадковості для ігрових систем.

Кожному можливому результату в гача-системі відповідає певна ймовірність випадіння, яка зазвичай називається drop rate. Ці значення визначають частоту появи предметів або персонажів різної рідкості.

Під час кожної спроби система виконує наступні базові дії:

1. Створюється випадкове число в заданому діапазоні.
2. Згенероване значення співставляється із заданими порогами ймовірностей.
3. Встановлюється результат – певний предмет або персонаж.

Таким чином, кожна спроба може бути представлена як випадкова подія з певною ймовірністю успіху. У найпростішому випадку таку систему можна

описати за допомогою біноміального або геометричного розподілу, де кожна спроба є незалежною від попередньої.

У більшості сучасних гача-ігор механізм отримання випадкових нагород ускладняються додатковими правилами. До таких правил належать системи гарантії отримання рідкісного предмета після певної кількості спроб, підвищення шансів при досягнені визначеного порогу та спеціальні події з модифікаціями ймовірності. Такі елементи змінюють традиційну модель незалежних випадкових подій та створюють більш складну й динамічну систему.

Через всі ці умови гача-механіка може бути описана у вигляді математичної моделі. Це дозволяє застосовувати методи статистичного аналізу та імітаційного моделювання для дослідження реальної поведінки системи.

У сучасній індустрії відеоігор значного поширення набула модель розповсюдження free-to-play (F2P), яка передбачає безкоштовний доступ до гри для користувачів. На відміну від класичної моделі pay-to-play (P2P), де гравець повинен придбати гру перед початком використання, F2P-проєкти дозволяють розпочати гру без фінансових витрат, а прибуток розробників формується за рахунок внутрішньоігрових покупок.

Основною ідеєю F2P-моделі є залучення максимальної кількості гравців за рахунок відсутності початкового бар'єру входу. У подальшому частина користувачів здійснює добровільні покупки, які можуть надавати доступ до додаткового контенту, косметичних предметів, ресурсів або нових персонажів.

Одним із найбільш поширених інструментів монетизації в F2P-іграх є системи випадкових винагород, зокрема гача-механіка. Вона дозволяє інтегрувати у гру елемент випадковості, що стимулює повторні спроби отримання бажаного предмета або персонажа. У багатьох випадках такі спроби можуть здійснюватися як за безкоштовну внутрішньоігрову валюту, отриману під час проходження гри, так і за валюту, придбану за реальні кошти.

Ключовим аспектом F2P-економіки є підтримання рівноваги між доступністю контенту для безкоштовних гравців і стимулюванням до здійснення покупок. Розробники прагнуть розробити механіку, яка дозволяє гравцям отримувати нагороди без фінансових витрат, але при цьому дає змогу значно пришвидшити досягнення бажаних результатів шляхом вкладення коштів.

У цьому випадку гача-системи слугують інструментом, який інтегрує ігрову механіку з економічною моделлю. Вони сприяють утриманню довготривалого інтересу гравців, забезпечують постійне впровадження нового контенту і створюють надійне джерело доходу для розробників.

Таким чином, гача-механіка виступає не просто частиною ігрового процесу, а й ключовим елементом економічної системи сучасних free-to-play проєктів.

Genshin Impact – це гра, розроблена китайською компанією HoYoverse, вона була офіційно випущена у вересні 2020 року та доступна для персональних комп'ютерів, мобільних телефонів та консолей. Причини популярності цієї гри є поєднання наступних факторів:

- Великий відкритий світ.
- Сюжетна складова гри.
- Система розвитку персонажів.
- Регулярне оновлення контенту.

У грі Genshin Impact система отримання персонажів і зброї реалізується через банери, які є окремими елементами гача-системи зі своїми правилами та специфікою. Кожен банер пропонує визначений набір доступних нагород, встановлені шанси на їх випадіння, а також передбачає механізми гарантії отримання рідкісних предметів.

Головною особливістю банерів у Genshin Impact є система pity, яка гарантує отримання предмета певної рідкості після конкретної кількості спроб.

Окрім цього, деякі банери включають механізм 50/50, що визначає ймовірність випадання івентового персонажа або зброї.

У грі банери розділяються на кілька основних типів: івентовий банер персонажа, постійний банер і банер зброї. Кожен із цих типів має свої унікальні правила та особливості отримання нагород.

Дані щодо активних банерів, доступних персонажів і умов отримання винагород відображаються прямо в ігровому інтерфейсі. Це дає гравцям змогу оперативно ознайомитися з актуальними подіями та зважити свої можливості перед витрачанням внутрішньоігрової валюти.

1.2 Структура та типи банерів у Genshin Impact

Банер івентового персонажа є основним й самим популярним банером у гравців. Завдяки ньому, є можливість отримати обмежених у часі персонажів, які недоступні у постійному банері. Після завершення певного періоду (зазвичай 3 тижні) склад такого банера змінюється.

Основна особливість цього банеру – система 50/50, де під час отримання бажаного п’ятизіркового персонажа є два можливі варіанти.

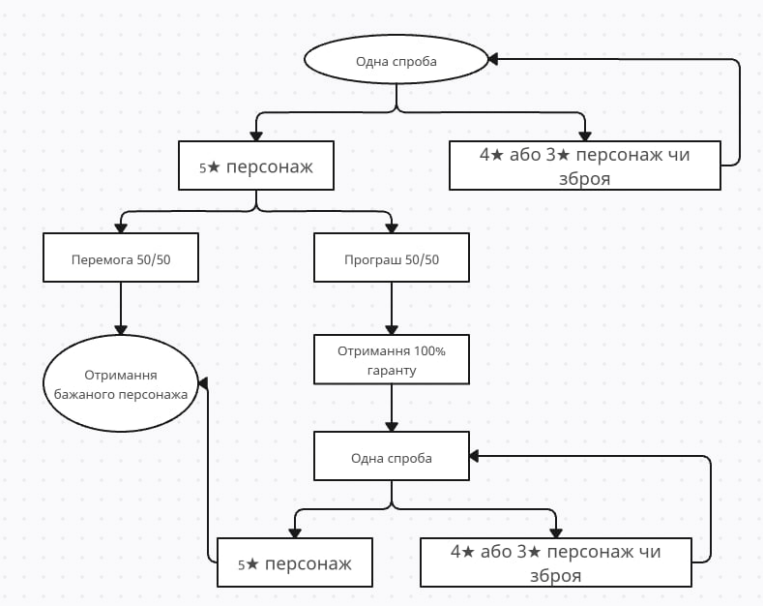

Рис. 1.1 - Схема роботи івентового банера.

Схема демонструє принцип роботи івентового банера персонажа у грі Genshin Impact. Після кожної спроби система перевіряє результат крутки. Якщо гравець не отримує п’ятизіркового персонажа, випадінням стає тризірковий або чотиризірковий предмет чи персонаж, після чого виконується наступна спроба. У разі отримання п’ятизіркового персонажа система визначає результат механізму 50/50. При перемозі 50/50 гравець одразу отримує бажаного івентового персонажа. Якщо ж 50/50 програно, активується механізм гаранту, за яким наступний п’ятизірковий персонаж буде гарантовано івентовим.

Крім механізму 50/50, в івентовому банері використовується система pity, яка обмежує максимальну кількість спроб до отримання персонажа високої рідкості. Базова ймовірність отримання п’ятизіркового персонажа є відносно низькою, проте після певної кількості невдалих спроб шанс поступово починає збільшуватися.

У гравців умовно сформувалися такі поняття:

- Early pity – отримання персонажа на ранніх крутках з маленьким шансом (1-50);
- Mid pity – отримання персонажа на ранніх крутках, але з більшим шансом (51-75);
- Soft pity – отримання персонажа на пізніх крутках з дуже великим шансом (76-80);
- Late pity – отримання персонажа на пізніх крутках, але з меншим шансом (81-89);
- Hard pity - максимальна кількість круток, після якої гарантовано випадає п’ятизірковий персонаж (90).

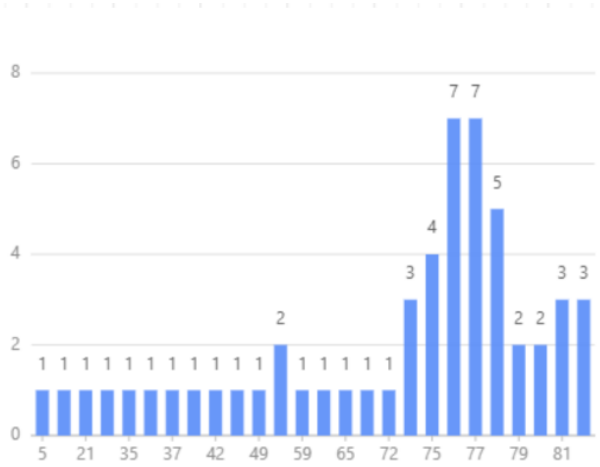

Рис. 1.2 – Розподіл випадіння п’ятизіркових персонажів за кількістю круток на основі особистої статистики.

Діаграма демонструє розподіл отримання п’ятизіркових персонажів залежно від кількості виконаних круток в івентовому банері. По осі X відображається кількість круток до отримання персонажа, а по осі Y – кількість випадків отримання п’ятизіркових персонажа на відповідній крутці.

На основі отриманих даних можна зазначити, що найбільша кількість п’ятизіркових персонажів випадає у діапазоні приблизно від 74 до 78 круток. Це відповідає механізму soft pity у грі Genshin Impact, який передбачає поступове збільшення ймовірності отримання п’ятизіркового персонажа після близько 74 крутки.

На діаграмі також можна побачити випадки раннього отримання персонажів, коли п’яти персонаж з'являвся значно раніше порогу soft pity. Це пов'язано з випадковістю роботи RNG-системи.

Отримані результати свідчать про те, що більшість п’ятизіркових персонажів у розглянутій вибірці були отримані в зоні підвищеної ймовірності випаду. Це відповідає офіційно заявленому принципу роботи pity-системи у грі Genshin Impact.

Особливістю також є те, що історія pity зберігається між змінами івентових банерів. Наприклад, якщо гравець зробив 50 круток в одному банері,

але не отримав п’ятизіркового персонажа, після зміни банера ці крутки не скидаються і продовжують враховуватись.

Постійний банер, або стандартний банер, є одним із базових типів банерів у грі Genshin Impact. На відміну від івентових банерів, його вміст практично не змінюється, а доступ до нього залишається постійним протягом усього часу існування гри.

Окрім п’ятизіркових персонажів, гравець має можливість отримати п’ятизіркову зброю. У список п’ятизіркових персонажів входять, так звані, стандартні персонажі, котрих також можна отримати під час програшу 50/50 в івентовому банері.

Головною характеристикою постійного банера є те, що він не має механізму 50/50. Іншими словами, у ньому відсутній спеціальний івентовий персонаж із підвищеною ймовірністю випадіння. Усі доступні п’ятизіркові персонажі та зброя представлені в постійному наборі й випадають у випадковому порядку.

При цьому система soft pity також продовжує діяти, поступово збільшуючи шанс випадіння п’ятизіркового предмета після великої кількості невдалих спроб.

Банер зброї в грі Genshin Impact є унікальним типом банера, завдяки якому гравці можуть отримати рідкісну зброю. На відміну від івентового банера персонажів, головними нагородами тут виступає п’ятизіркова та чотиризіркова зброя, яка може суттєво підвищити бойову міць персонажів і їхню ефективність у різних битвах.

Особливість банера зі зброєю полягає в тому, що він пропонує два івентових п’ятизіркові види зброї з підвищеною ймовірністю випадіння. Проте, навіть при отриманні п’ятизіркової зброї, гравець не гарантовано здобуде бажаний предмет, оскільки система випадково обирає одну з доступних нагород. Окрім цього, в банері присутня стандартна зброя, яку

можна отримати у постійному банері. З цього випливає, що шанс отримати бажану зброю дорівнює не 50/50, а 25/75.

У грі впроваджено механіку під назвою Шлях втілення (Epitomized Path), яка дозволяє підвищити ймовірність отримання конкретної п’ятизіркової зброї. Перед тим як розпочати використання круток, гравець має змогу вибрати бажану зброю цього рівня. Якщо під час випадання п’ятизіркових предметів натомість з’являється інший вид зброї, накопичуються спеціальні очки долі. Як тільки досягається визначена кількість цих очок, наступна п’ятизіркова зброя гарантовано відповідатиме попередньо обраному предмету.

Крім того, в банері зброї також діє система soft pity, яка поступово збільшує шанси після певної кількості невдалих спроб.

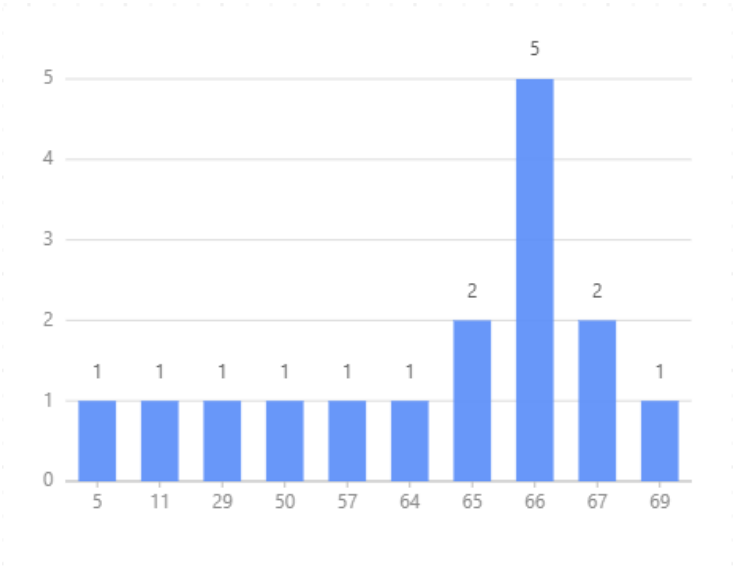

Рис. 1.3 - Розподіл випадіння п’ятизіркової зброї за кількістю круток на основі особистої статистики.

Діаграма демонструє розподіл отримання п’ятизіркової зброї залежно від кількості виконаних круток в банері зброї. По осі X відображається кількість круток до отримання зброї, а по осі Y – кількість випадків отримання п’ятизіркової зброї на відповідній крутці.

На основі отриманих даних можна зазначити, що найбільша кількість п’ятизіркової зброї випадає у діапазоні приблизно від 65 до 67 круток. Це

відповідає механізму soft pity у грі Genshin Impact, який передбачає поступове збільшення ймовірності отримання п’ятизіркової зброї після близько 65 крутки.

На діаграмі також відображені поодинокі випадки раннього отримання п’ятизіркової зброї, зокрема на 5-й, 11-й та 29-й спробах. Подібні результати пояснюються випадковістю гача-системи, що підтверджує можливість отримання рідкісних предметів значно раніше від гарантованого результату.

Незважаючи на наявність додаткових механізмів гарантії, банер зброї часто вважається більш ризикованим для гравців порівняно з івентовим банером персонажа. Це пов’язано з тим, що для отримання конкретної зброї може знадобитися декілька спрацьовувань гаранту, що збільшує загальну кількість необхідних круток.

Таким чином, банер зброї є найбільш складним з точки зору внутрішньої логіки роботи серед усіх типів банерів у Genshin Impact. Поєднання систем pity, підвищених шансів та механізму Шлях втілення робить його цікавим прикладом реалізації багаторівневої гача-системи.

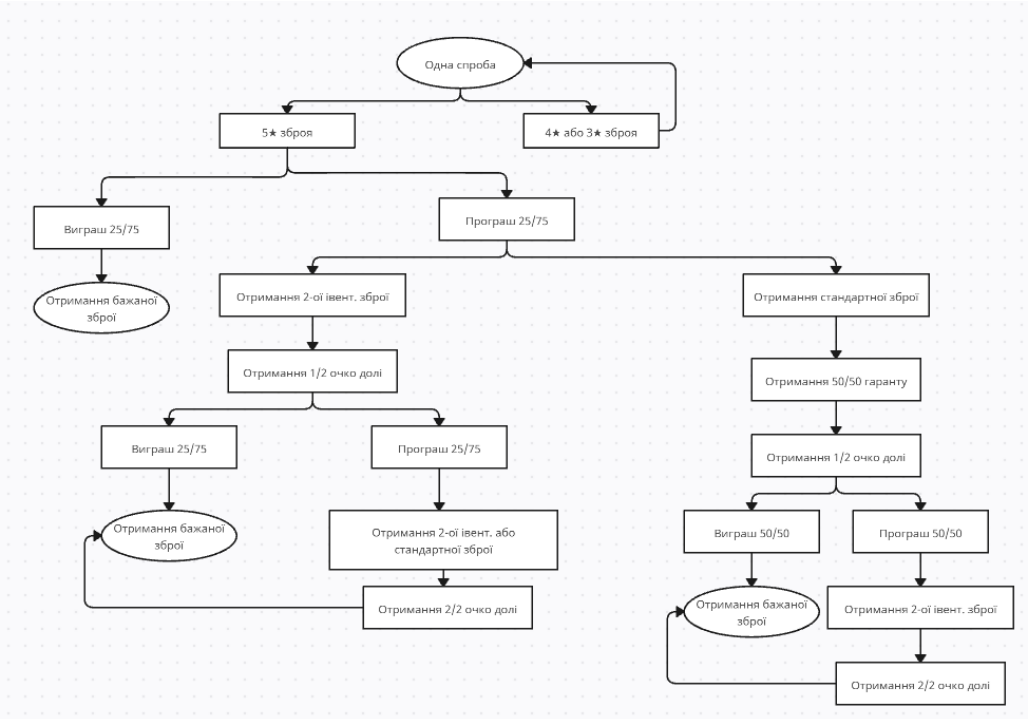

Рис. 1.4 - Схема роботи банера зброї.

Схема демонструє принцип роботи банера зброї. Після кожної спроби система визначає рідкість отриманого предмета. Якщо п’ятизіркова зброя не випадає, гравець отримує предмет нижчої рідкості та продовжує обертання до моменту активації системи гарантії.

Після отримання п’ятизіркової зброї активується механізм 25/75. У разі успішного результату гравець отримує одну бажану івентову зброю, у разі невдачі – стандартну або небажану п’ятизіркову зброю. Якщо випадає небажаний результат, активуються додаткові гарантійні механізми, які збільшують шанс отримати івентову зброю під час наступних спроб.

Схема також ілюструє функціонування системи Шлях втілення, де після досягнення необхідної кількості очок наступна зброя легендарного рівня обов'язково відповідатиме обраному шляху.

Банер із зброєю має значно складнішу структуру порівняно з івентовим банером персонажа, оскільки в ньому одночасно діють кілька механізмів гарантії: система pity, механізм 25/75 та Шлях втілення. Завдяки цьому поступово знижується роль випадковості, що зрештою дає змогу гравцеві гарантовано отримати бажану зброю після визначеної кількості спроб.

1.3 Механізми гарантії в Genshin Impact

Однією з характерних рис гача-системи Genshin Impact є прозорість у поданні інформації про ймовірності випадіння предметів і персонажів. Розробники надають офіційну інформацію про шанси отримання нагород різного рівня рідкості, яка доступна безпосередньо в ігровому клієнті у розділі деталей банера. Це дозволяє гравцям ознайомитися з базовими ймовірностями ще до виконання круток.

Для івентового банера персонажа встановлені такі базові шанси отримання нагород:

Рідкість нагороди	Базова ймовірність
П’ятизірковий персонаж	0,6%
Чотиризірковий персонаж чи зброя	5,1%
Тризіркова зброя	94,3%

З наведених даних видно, що ймовірність отримання п’ятизіркового персонажа під час однієї крутки є досить низькою і складає лише 0,6 %. Саме тому в грі передбачено систему pity, яка поступово компенсує цей низький шанс і гарантує отримання персонажа високої рідкості після визначеної кількості спроб.

Для банера зброї базові шанси дещо відрізняються:

Рідкість нагороди	Базова ймовірність
П’ятизіркова зброя	0,7%
Чотиризірковий персонаж чи зброя	6%
Тризіркова зброя	93,3%

Хоч різниця у шансах незначна, у банері зброї базова ймовірність отримати п’ятизірковий предмет трохи вища. До того ж, для банера зброї максимальний гарант становить 80 круток, тоді як у банері персонажа він дорівнює 90 круткам.

Варто підкреслити, що наведені значення відображають саме базові шанси випадіння. Реальна ймовірність отримання п’ятизіркової нагороди змінюється в міру накопичення pity і суттєво зростає, коли досягається зона soft pity. У зв’язку з цим середня фактична кількість круток для отримання п’ятизіркової нагороди зазвичай є меншою за встановлене максимальне значення гаранту.

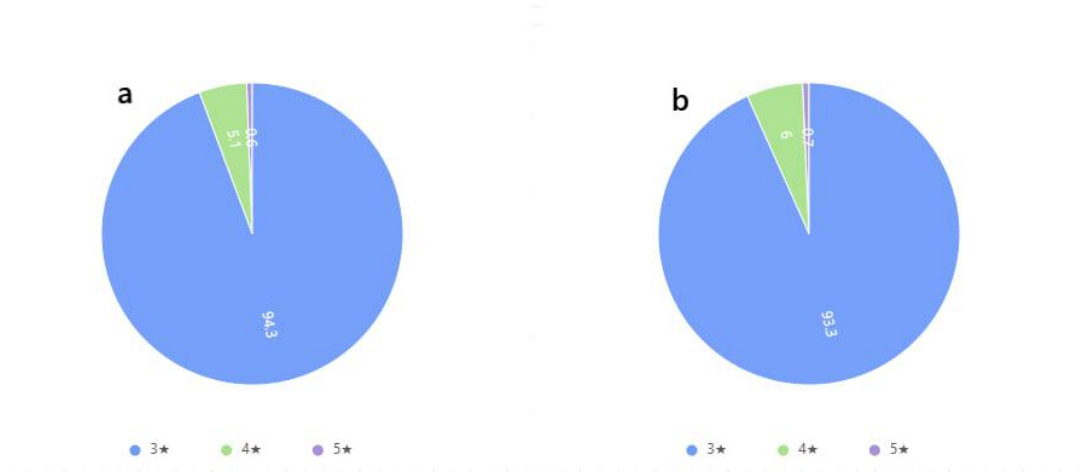

Рис. 1.5 – Діаграми базової ймовірності для a. Івентового банеру персонажа, b. Банеру зброї.

Однією з ключових особливостей гача-системи Genshin Impact є використання механізмів гарантії, які дозволяють обмежити вплив випадковості на результати круток. На відміну від класичних систем випадкових винагород, де кожна спроба є незалежною та не враховує попередні результати, у Genshin Impact реалізовано декілька додаткових механізмів, спрямованих на підвищення ймовірності отримання бажаної нагороди після певної кількості невдалих спроб.

Використання таких механізмів дозволяє зробити систему більш передбачуваною для гравців та зменшити ризик надмірно великої кількості круток без отримання цінних предметів або персонажів. Завдяки цьому гра підтримує баланс між випадковістю, яка є основою гача-механіки, та гарантованим прогресом користувача.

Механізми гарантії відіграють ключову роль не лише у формуванні ігрового досвіду, але й у математичному аналізі роботи гача-системи. Вони визначають ймовірності отримання персонажів і зброї, а також впливають на середню кількість спроб, необхідних для досягнення бажаного результату. У наступних підпунктах детально розглядаються принципи функціонування pity-системи, механізму 50/50 та гарантії, застосовані в банері зброї.

Однією із ключових особливостей гача-системи в Genshin Impact є механізм pity, який зменшує вплив випадковості при отриманні рідкісних персонажів і зброї. Його основна мета – забезпечити гарантоване отримання предмета високої рідкості після визначеної кількості невдалих спроб.

У класичних випадкових системах кожна подія не залежить від попередніх результатів, тобто ймовірність отримання бажаного результату залишається постійною, незалежно від кількості невдалих спроб. У Genshin Impact застосовується інший принцип: гра відслідковує кількість попередніх круток без отримання п’ятизіркового предмета та поступово підвищує шанси на його випадіння.

Для івентового банера персонажа максимальна кількість круток до гарантованого отримання п’ятизіркового персонажа становить 90. Якщо протягом цих спроб гравець не отримав персонажа найвищої рідкості, на дев'яностій крутці система гарантовано видає п’ятизіркову нагороду. Такий механізм отримав неофіційну назву hard pity.

Проте на практиці більшість гравців отримують п’ятизіркового персонажа значно раніше за 90-ту крутку. Це пов'язано з існуванням іншого механізму, який у спільноті отримав назву soft pity. Після приблизно 74–75 крутки ймовірність випадіння п’ятизіркового персонажа починає поступово зростати з кожною наступною спробою.

Хоча офіційної публікації щодо точної формули, яку використовують розробники багаторічний аналіз статистики мільйонів круток дозволив гравцям встановити закономірність роботи цього механізму. Саме тому більшість п’ятизіркових персонажів випадають у діапазоні від 75 до 80 круток.

Впровадження механізму soft pity підвищує передбачуваність системи для користувачів. Наприклад, якщо після 70 спроб персонаж найвищого рангу не був отриманий, ймовірність його випадіння під час наступних спроб суттєво

зростає порівняно з початком. Це дозволяє гравцям прогнозувати обсяг ресурсів, необхідних для досягнення цільового результату.

Ще однією важливою особливістю pity-системи є збереження прогресу між івентовими банерами. Якщо гравець виконав певну кількість круток в одному банері, але не отримав п’ятизіркового персонажа, після зміни банера накопичений pity не скидається. Наприклад, якщо до завершення попереднього банера було виконано 60 круток без отримання п’ятизіркового персонажа, то в новому банері відлік продовжиться саме з цього значення.

З точки зору математичного аналізу pity-система є одним із найважливіших елементів гача-механіки. Саме вона суттєво впливає на розподіл ймовірностей та середню кількість круток, необхідних для отримання персонажа.

Одним із найбільш відомих елементів гача-системи Genshin Impact є механізм, який гравці називають 50/50. Він використовується в івентовому банері персонажа та визначає, чи отримає гравець саме персонажа, представленого в поточному банері, після випадіння п’ятизіркової нагороди.

На відміну від pity-системи, яка відповідає за кількість круток до отримання п’ятизіркового персонажа, механізм 50/50 визначає результат уже після того, як п’ятизірковий персонаж був отриманий. Саме тому ці дві системи працюють разом і доповнюють одна одну.

Під час першого отримання п’ятизіркового персонажа в івентовому банері система виконує перевірку 50/50. У разі успіху гравець отримує персонажа, який представлений у поточному банері. У разі невдачі випадає один із стандартних п’ятизіркових персонажів, доступних у грі.

Рис. 1.6 – Схема роботи системи 50/50.

Особливістю цієї системи є наявність гарантії після програшу 50/50. Якщо гравець отримав стандартного персонажа замість івентового, наступний п’ятизірковий персонаж у цьому типі банера гарантовано буде івентовим. Завдяки цьому навіть у випадку невдалого результату система поступово наближає гравця до отримання бажаної нагороди.

Рис. 1.7 – Приклад роботи системи 50/50, де персонаж Кецін є програшом 50/50 (стандартним), персонаж Нефер є гарантованим після програшу, а Коломбіна – виграшем після гарантованого.

На практиці це означає, що для отримання конкретного персонажа може знадобитися від одного до двох спрацьовувань pity-системи. У найкращому випадку гравець виграє 50/50 та отримує персонажа одразу після випадіння першого п’ятизіркового. У найгіршому випадку спочатку випадає стандартний персонаж, після чого доводиться накопичувати ще один pity для отримання гарантованого персонажа банера.

Саме через це в спільноті гравців часто використовується поняття гарант. Якщо попередній п’ятизірковий персонаж був стандартним, то вважається, що акаунт перебуває на гаранті, а наступний п’ятизірковий персонаж обов'язково буде івентовим.

З точки зору теорії ймовірностей механізм 50/50 є додатковим рівнем випадковості, який впливає на кількість необхідних ресурсів для отримання бажаного персонажа. Саме тому під час планування круток гравці зазвичай враховують не лише кількість круток до pity, але й можливість програшу 50/50.

Для математичного моделювання цей механізм має особливе значення, оскільки він безпосередньо впливає на очікувану кількість круток до отримання івентового персонажа.

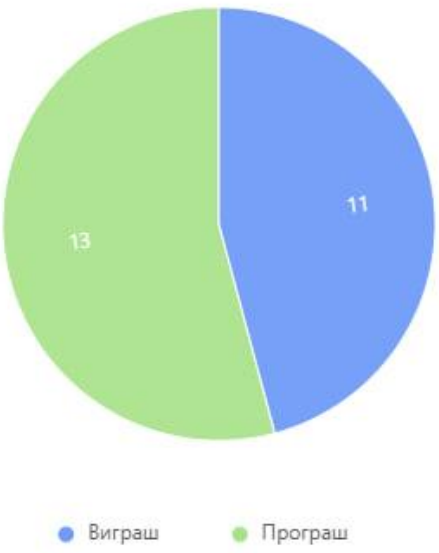

Рис. 1.8 - Співвідношення виграшів та програшів механізму 50/50 на основі особистої статистики.

Діаграма відображає співвідношення виграшів та програшів механізму 50/50 в івентовому банері персонажа на основі особистої історії круток. За весь період дослідження було зафіксовано 24 спрацювання механізму 50/50, з яких 11 завершилися перемогою та 13 – поразкою.

У відсотковому співвідношенні це становить приблизно 45,8 % виграшів та 54,2 % програшів. Отримані результати є близькими до теоретичного значення 50 %, яке закладене в механізм роботи івентового банера.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Технологічне забезпечення програмної системи

Для реалізації програмного комплексу аналізу механік випадковості було обрано технологічний стек, що відповідає сучасним стандартам розробки вебзастосунків та систем обробки даних. Архітектура системи базується на трьох основних компонентах: фреймворку Spring Boot, документно-орієнтованій СУБД MongoDB та платформі контейнеризації Docker. Комплексне використання цих технологій забезпечує високу продуктивність обробки статистичних вибірок, гнучкість збереження даних та уніфікацію процесу розгортання програмного забезпечення.

У межах розробленої системи користувач може отримувати та аналізувати інформацію про результати круток, статистику отримання персонажів, роботу механізмів pity і 50/50, а також інші показники, необхідні для дослідження гача-системи. Для реалізації такої функціональності необхідно забезпечити взаємодію між серверною частиною застосунку, базою даних та середовищем виконання.

Платформою для реалізації бізнес-логіки серверної частини обрано фреймворк Spring Boot. Вибір зумовлений його модульною архітектурою, підтримкою концепції інверсії керування (IoC) та вбудованими інструментами для проєктування RESTful API. Фреймворк містить готові рішення для безшовної інтеграції з базами даних (Spring Data) та автоматичної конфігурації компонентів. Це дозволило мінімізувати часові витрати на налаштування інфраструктури застосунку та зосередити розробку безпосередньо на алгоритмах обробки математичної й статистичної логіки досліджуваних механік.

Для управління даними та їх довготривалого зберігання інтегровано MongoDB – документно-орієнтовану нереляційну систему керування базами

даних типу NoSQL. Цей підхід вважається оптимальним для обробки логів ігрових спроб, оскільки структура результатів генерації може варіюватися залежно від типу банера. Використання документів у форматах BSON/JSON дозволяє позбутися обмежень жорстко визначених схем, характерних для реляційних баз даних. Окрім того, MongoDB забезпечує високу продуктивність при виконанні операцій запису та агрегації, навіть у випадках роботи з великими масивами стохастичних даних.

Для автоматизації розгортання та забезпечення відтворюваності середовища функціонування системи використано платформу Docker. Контейнеризація дозволяє інкапсулювати застосунок, СУБД та їхні системні залежності в ізольовані контейнери. Це нівелює проблему конфігураційної несумісності («dependency hell») під час міграції системи між різними обчислювальними вузлами та гарантує ідентичність умов функціонування програмного забезпечення незалежно від архітектури хостової операційної системи.

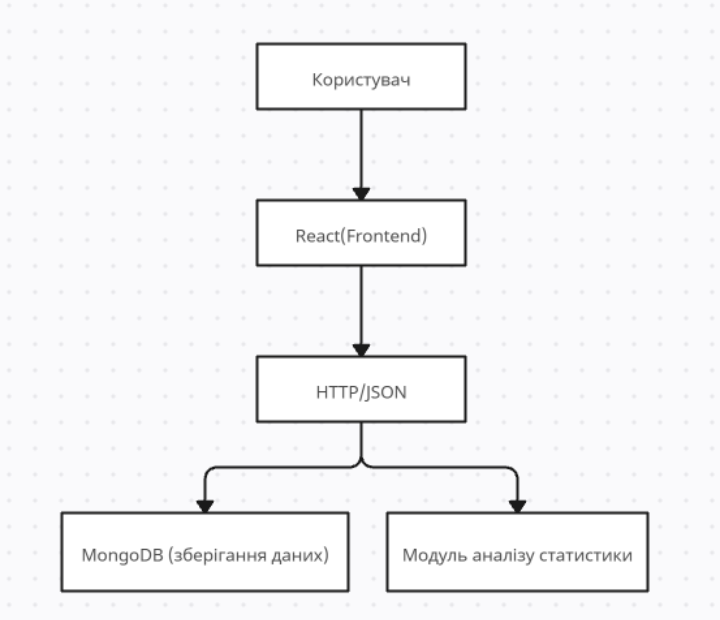

Рис. 2.1 – Схема загальної архітектури системи.

Узагальнена схема роботи розробленого програмного комплексу представлена у форматі класичної клієнт-серверної взаємодії. Користувач надсилає HTTP-запит до серверної частини застосунку через REST API.

Серверний компонент виконує перевірку і обробку отриманих даних, здійснює виконання необхідних математичних алгоритмів аналізу та взаємодіє з системою управління базами даних MongoDB для збереження або отримання історичних логів генерації. Після цього створена аналітична відповідь серіалізується у структурований формат JSON та передається клієнтові. Для забезпечення ізоляції всіх компонентів системи вони поміщені в окремі Docker-контейнери, що спрощує управління залежностями, покращує декларативність і полегшує процеси розгортання та технічного обслуговування.

Розроблена архітектура та підібраний набір технологій створюють основу для побудови слабко пов'язаної, гнучкої та масштабованої системи, призначеної для аналізу статистичних даних механік випадкового заохочення. Завдяки чіткому розподілу відповідальності між компонентами досягається оптимальний баланс між обчислювальною ефективністю при роботі з великими обсягами стохастичної інформації, надійністю зберігання даних і зручністю розгортання програмного забезпечення.

2.2 Обґрунтування вибору Spring Boot

Для створення серверної частини програмної системи було обрано фреймворк Spring Boot, завдяки його численним можливостям у розробці вебзастосунків, підтримці REST API та зручній інтеграції з базами даних.

Spring Boot вважається одним із найпопулярніших фреймворків для розробки Java-застосунків, оскільки значно спрощує процес створення серверної частини. На відміну від класичних підходів до створення Java-проєктів, він забезпечує швидке налаштування програми, дозволяючи зосередитись на розробці основної логіки системи.

У поточному проєкті Spring Boot використовується для обробки статистичних даних, організації роботи з базою даних MongoDB та реалізації REST API, через яке клієнтська частина програми комунікує із сервером.

Вибір на користь Spring Boot також зумовлений його підтримкою роботи з даними у форматі JSON. Оскільки статистичні дані круток і аналітичні результати зберігаються в структурованому вигляді у форматі JSON, це значно спрощує обмін інформацією та її обробку між різними компонентами системи.

Додатковою перевагою є те, що Spring Boot підтримує масштабованість і дозволяє легко додавати нові модулі для обчислення статистики, аналізу даних чи імітації роботи гача-механік. Таким чином, цей фреймворк є оптимальним вибором для реалізації системи аналізу гача-механік.

Рис. 2.2 – Логотип Spring Boot.

У наступних підпунктах детально проаналізовано архітектурні особливості фреймворку Spring Boot, принципи проєктування RESTful API, а також практичну реалізацію серверної бізнес-логики. Окрему увагу приділено перевагам використання обраної платформи для обробки, агрегації та аналізу масивів статистичних даних.

Spring Boot є високорівневим Java-фреймворком, розробленим для оптимізації та спрощення життєвого циклу створення серверних застосунків

підприємницького рівня. Ключова концептуальна особливість платформи полягає в автоматизації рутинних інфраструктурних конфігурацій, які в класичних проєктах на базі Spring Framework вимагали значних трудовитрат на ручне налаштування через XML-файли або Java-класи конфігурації. Такий підхід реалізує парадигму Convention over Configuration, що дозволяє розробнику декомпозувати інфраструктурні задачі та зосередити обчислювальні ресурси і логіку проєкту безпосередньо на реалізації бізнес-вимог системи.

Фундаментальною перевагою Spring Boot є інтегрований механізм автоматичної конфігурації (Auto-Configuration). Під час ініціалізації та запуску застосунку фреймворк здійснює динамічний сканаж контексту (classpath), ідентифікує наявні залежності (starter-бібліотеки) та на основі умовних анотацій (conditional annotations) автоматично створює і конфігурує необхідні екземпляри класів (Spring Beans). Застосування Auto-Configuration дозволяє радикально мінімізувати обсяг шаблонового коду (boilerplate code) та суттєво підвищити швидкість розгортання прототипів і готових модулів програмного забезпечення.

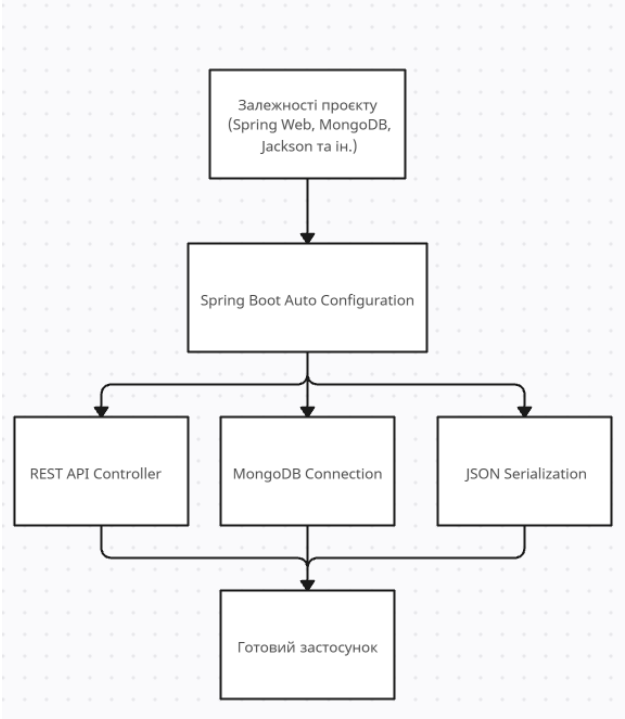

Рис. 2.3 – Схема роботи Auto Configuration у Spring Boot.

Важливою перевагою Spring Boot є його висока адаптивність та безшовна інтеграція з компонентами стороннього програмного забезпечення (Third-Party Software). У межах архітектури цього проєкту фреймворк виступає інтеграційним ядром, що забезпечує взаємодію з нереляційною базою даних MongoDB на рівні шару даних, а також оптимізований для розгортання в ізольованому середовищі Docker. Завдяки концепції «стартерів» (starter dependencies), зокрема модулю spring-boot-starter-data-mongodb, розробник отримує готові абстракції для управління життєвим циклом підключень та виконання запитів, що мінімізує інфраструктурний оверхед і дозволяє гнучко масштабувати систему обробки даних.

Окремим функціональним аспектом Spring Boot є розвинена архітектурна підтримка розробки інтерфейсів RESTful API та нативна обробка об'єктів у форматі JSON. Фреймворк містить інтегровані засоби вебмодуля (Spring Web), які автоматично інкапсулюють процеси маршалінгу та демаршалінгу (серіалізації Java-об'єктів у JSON-документи і десеріалізації вхідних HTTP-запитів у DTO-моделі). Ця автоматизація, що базується на бібліотеці Jackson, повністю нівелює необхідність ручного парсингу пакетів даних, забезпечує високу швидкість обміну інформацією між клієнтською та серверною частинами застосунку, а також гарантує цілісність передачі великих масивів статистичних логів.

Окрім суто технічних переваг, Spring Boot вирізняється потужною глобальною екосистемою, детальною офіційною документацією та активною підтримкою з боку інженерної спільноти. Це створює сприятливі умови для швидкого освоєння інструментарію завдяки низькому порогу входження, а також забезпечує регулярне оновлення безпекових патчів і модулів. У результаті фреймворк став галузевим стандартом, який успішно використовується як у некомерційних академічних дослідженнях, так і під час розробки високонавантажених комерційних систем рівня Enterprise.

На основі проведеного аналізу архітектурних особливостей фреймворку Spring Boot можна виділити кілька ключових переваг, які роблять його ефективним для реалізації цього дослідження:

- Зниження інфраструктурного навантаження завдяки автоматизованій конфігурації через механізм Auto-Configuration.
- Вбудована підтримка RESTful API, що включає засоби для швидкої серіалізації та десеріалізації даних у форматі JSON.
- Абстрактний доступ до даних за допомогою інтеграції з MongoDB через Spring Data, забезпечуючи безшовну роботу із базою даних.
- Можливість декларативного підходу та висока гнучкість при розгортанні серверної частини в контейнеризованих середовищах.

Однією з основних функцій Spring Boot у межах даного проєкту є реалізація REST API для взаємодії між клієнтською та серверною частинами системи. REST API забезпечує передачу даних між frontend-застосунком та сервером за допомогою HTTP-запитів.

REST (Representational State Transfer) фактично став галузевим стандартом серед архітектурних стилів, які застосовуються для розробки та створення розподілених вебсервісів. Основою REST API є парадигма клієнт-серверної взаємодії без збереження стану (Stateless). У цій моделі клієнтський додаток надсилає стандартизовані HTTP-запити до заданих вебресурсів, а сервер обробляє їх і повертає представлення стану цих ресурсів у структурованому форматі, який не залежить від платформи, як правило, у вигляді JSON-документів.

У системі аналізу гача-механік REST API застосовується для виконання таких завдань:

- збору статистичних даних про крутки;
- збереження отриманих результатів у базі даних;
- обміну інформацією щодо персонажів та рівнів pity;

- отримання результатів аналізу зібраної статистики.

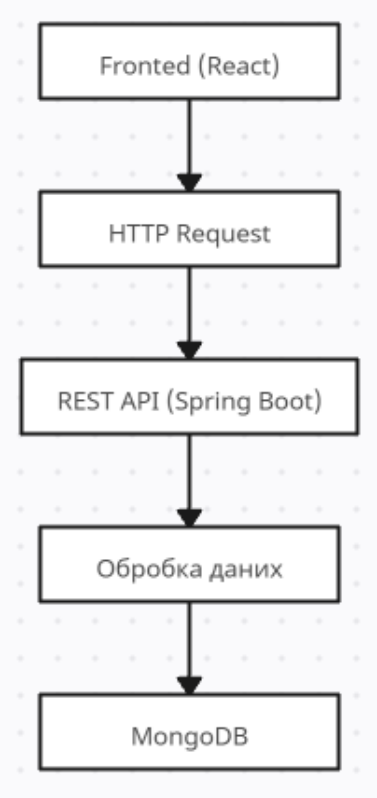

Рис. 2.4 – Схема роботи REST API.

Для обробки HTTP-запитів у Spring Boot використовуються спеціальні контролери (Controllers). Вони відповідають за прийом запитів від клієнта, виклик необхідної логіки застосунку та формування відповіді.

Основними типами HTTP-запитів, які можуть використовуватися в системі, є:

Метод	Призначення
GET	Отримання даних
POST	Надсилання нових даних
PUT	Оновлення інформації
DELETE	Видалення даних

Наприклад, клієнтська частина може надсилати GET-запит для отримання статистики круток певного користувача, а серверна частина — повертати результат у форматі JSON.

Приклад JSON-відповіді REST API:

```
{
"totalPulls": 3200,
"averagePity": 72.25,
"winRate50_50": 45.8
}
```

Однією з важливих переваг REST API є його здатність забезпечувати незалежність між клієнтською і серверною частинами системи. Це дозволяє розробляти frontend і backend окремо, при цьому забезпечуючи їх ефективну взаємодію завдяки стандартизованим HTTP-запитам.

Додатково, використання REST API сприяє легкому масштабуванню системи. Воно дає можливість інтегрувати нові сервіси або модулі для аналізу статистики без істотного втручання в архітектуру застосунку.

Таким чином, REST API виступає ключовою складовою програмної системи, адже через нього здійснюється обмін статистичними даними між компонентами застосунку та організовується взаємодія користувача із серверною частиною.

Серверна логіка становить одну з ключових складових програмної системи, адже саме вона відповідає за обробку запитів користувачів, аналіз статистичних даних і взаємодію з базою даних. У рамках цього проєкту серверна частина була реалізована за допомогою Spring Boot, який функціонує як центральний модуль для обробки інформації.

Головна мета серверної логіки полягає в отриманні даних із клієнтської частини застосунку, їх подальшій обробці та формуванні відповідей для користувачів. Зокрема, сервер здатний опрацьовувати інформацію про результати круток, виконувати обчислення середнього значення pity, аналізувати результати алгоритму 50/50 або генерувати загальну статистику щодо отриманих персонажів залежно від параметрів.

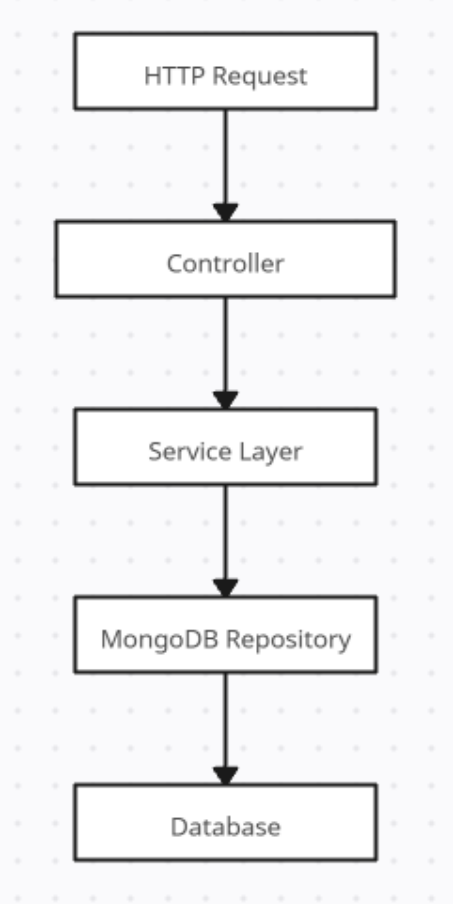

Рис. 2.5 – Схема роботи серверної логіки.

У межах розробки серверного компонента на базі Spring Boot було застосовано класичний патерн багатошарової архітектури (Multilayer Architecture), що базується на принципі розподілу відповідальності (Separation of Concerns). Функціональний контур системи декомпозовано на три основні логічні рівні:

- Шар контролерів (Web/Controller Layer): Представлений компонентами з анотацією @RestController. Вони виконують роль точок входу (End-points) для RESTful API, здійснюють перехоплення та валідацію вхідних HTTP-запитів, десеріалізують тіло запиту в об'єкти передачі даних (DTO) та делегують подальше виконання бізнес-процесів сервісному шару. Після обробки контролери формують та повертають клієнту уніфіковані HTTP-відповіді.

- Шар бізнес-логіки (Service Layer): Реалізований у вигляді сервісних класів (анотованих @Service). Цей рівень є ядром застосунку, де зосереджено всі алгоритми математичного моделювання, статистичного аналізу стохастичних даних гача-механік, а також логіку обчислення ймовірностей (pity та 50/50). Сервіси ізольовані від прямої взаємодії з протоколом HTTP, що забезпечує їхню незалежність від типу інтерфейсу.
- Шар доступу до даних (Data Access / Repository Layer): Представлений інтерфейсами-репозиторіями, які наслідують інтерфейси Spring Data (наприклад, MongoRepository). Вони інкапсулюють у собі логіку взаємодії з базою даних MongoDB, забезпечуючи виконання операцій CRUD (Create, Read, Update, Delete) та складних агрегаційних запитів без необхідності написання низькорівневого коду взаємодії з СУБД.

Триланкова архітектура сприяє зменшенню рівня зв'язності між компонентами системи (Low Coupling), що, у свою чергу, забезпечує високу модульність програмного коду. Розподіл на незалежні шари істотно покращує можливість тестування застосунку, оскільки спрощується проведення модульного тестування із використанням mock-об'єктів. Крім того, така структура полегшує процес рефакторингу і технічної підтримки, а також створює передумови для потенційного масштабування системи у випадку розширення її аналітичних функціональних можливостей.

У межах системи аналізу гача-механік серверна логіка може виконувати такі операції:

- збереження історії круток;
- обчислення середнього pity;
- визначення відсотка виграшів 50/50;
- аналіз статистики за певний період;

- формування загальної інформації про результати круток.

Однією з ключових архітектурних переваг фреймворку Spring Boot постає його інтуїтивна підтримка автоматизованої роботи з даними у форматі JSON. Ця можливість реалізується через інтеграцію модуля Spring Web із бібліотекою Jackson та її класом ObjectMapper. Серверний компонент забезпечує процес автоматичного зв'язування даних (Data Binding), у межах якого вхідні JSON-документи десеріалізуються в відповідні Java-об’єкти (POJO/DTO) одразу на етапі обробки запиту контролером. Завдяки цьому, аналітичні операції виконуються на рівні бізнес-логіки, після чого згенеровані результати серіалізуються в JSON-формат для передачі їх у відповідь. Така декларативна модель роботи виключає необхідність створення низькорівневих парсерів і суттєво знижує вірогідність помилок типізації у процесі розробки.

На додаток до автоматизованої обробки даних, серверна логіка виступає як центральне ядро управління системою. У рамках реалізації архітектурного патерну «тонкого клієнта» (Thin Client) всі ресурсоємні обчислення, стохастичне моделювання та роботи з масштабними масивами історичних даних переносяться на сервер, тобто Backend-компонент. Клієнтська частина програми, таким чином, зосереджується виключно на функціях представлення (Presentation Layer), відповідаючи за рендеринг графічного інтерфейсу та візуалізацію отриманих результатів аналітики, що суттєво знижує її обчислювальне навантаження.

У результаті застосування фреймворку Spring Boot для розробки серверної логіки створюється гнучка, структурована і високолінійна до навантажень система аналізу статистичних даних ігрових генерацій. Чітке розподілення функціональних зон між контрольним (Controllers), сервісним (Services) і персистентним (Repositories) рівнями сприяє спрощенню процесу модульного тестування, зменшенню технічного боргу, а також формує міцні підвалини для реалізації майбутніх процесів масштабування та модернізації програмного комплексу.

Однією з важливих переваг Spring Boot є підтримка масштабованості програмних систем. Масштабованість означає можливість розширення функціональності застосунку та збільшення обсягів оброблюваних даних без необхідності повного перепроєктування системи.

У межах системи аналізу гача-механік підтримка масштабованості є особливо важливою, оскільки зі збільшенням кількості користувачів та накопиченням статистики зростає обсяг даних, які необхідно обробляти та зберігати. Крім цього, у майбутньому система може бути доповнена новими модулями аналізу або підтримкою інших ігор із гача-механіками.

Spring Boot дозволяє будувати застосунки за модульним принципом. Завдяки цьому окремі компоненти системи можуть розроблятися та змінюватися незалежно один від одного. Наприклад, модуль обробки статистики, модуль роботи з базою даних або REST API можуть бути розширені без значного впливу на інші частини застосунку.

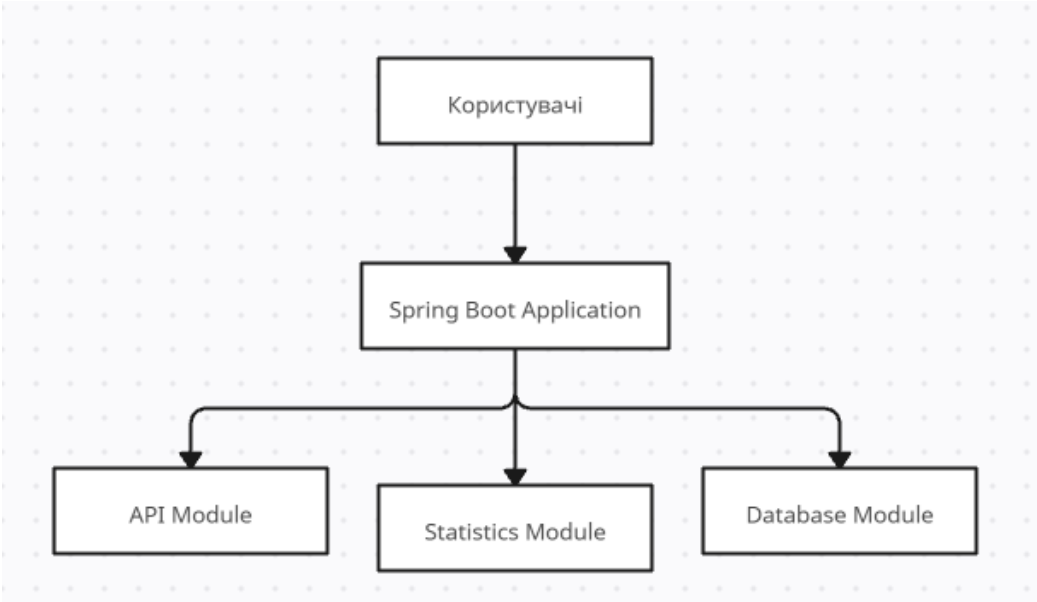

Рис. 2.6 – Схема масштабування системи.

Важливу роль у підтримці масштабованості також відіграє використання REST API. Завдяки розділенню frontend та backend частин системи клієнтський застосунок може змінюватися або оновлюватися незалежно від серверної

частини. Це спрощує модернізацію системи та дозволяє поступово розширювати її функціональність.

Ще однією перевагою Spring Boot є підтримка інтеграції з різними базами даних та зовнішніми сервісами. У разі необхідності MongoDB може бути замінена або доповнена іншими системами зберігання даних без повного переписування серверної логіки застосунку.

Крім цього, Spring Boot добре поєднується з Docker-контейнеризацією. У разі збільшення навантаження система може бути розгорнута одночасно на декількох контейнерах або серверах, що дозволяє підвищити продуктивність та стабільність роботи застосунку.

Окрему увагу необхідно приділити аспекту масштабованості статистичного аналізу. Із зростанням обсягів зібраних круток система набуває спроможності виконувати більш складні обчислення, аналізувати суттєво більші вибірки даних та генерувати більш точні й релевантні висновки стосовно функціонування гача-механіки.

У цьому контексті застосування Spring Boot забезпечує можливість розробки гнучкої та масштабованої серверної інфраструктури. Така система здатна ефективно обробляти значні обсяги статистичної інформації, що, своєю чергою, створює передумови для розширення функціональних можливостей програмного забезпечення в майбутньому.

Однією з ключових переваг Spring Boot є можливість легкої інтеграції з базами даних. У рамках цього проєкту серверна частина застосунку використовує MongoDB для збереження даних про статистику круток, результати аналізу та іншу інформацію, що стосується функціонування гача-системи.

Spring Boot надає широкий набір інструментів для роботи з базами даних, завдяки чому процес підключення та обробки даних стає значно простішим. Для взаємодії з MongoDB застосовується модуль Spring Data

MongoDB. Цей модуль забезпечує автоматизацію створення запитів до бази даних і адаптований для роботи з документно-орієнтованою структурою.

У рамках роботи застосунку статистика круток може зберігатися у вигляді окремих документів, кожен з яких містить інформацію про персонажа, кількість круток, результат 50/50, тип банера та інші параметри. Завдяки JSON-подібній структурі MongoDB ідеально підходить для роботи з такими динамічними даними, структура яких може змінюватися чи доповнюватися.

Використання Spring Boot дозволяє автоматизувати перетворення Java-об’єктів у документи MongoDB й у зворотному напрямку. Це значно спрощує взаємодію з базою даних, звільняючи розробника від необхідності вручну створювати складні SQL-запити або виконувати додаткову обробку даних.

Важливою перевагою Spring Boot у контексті роботи з даними є інтеграція з модулем Spring Data, який реалізує високорівневу абстракцію над шаром доступу до даних (Data Access Layer) за допомогою патерну Repository (Репозиторій). Замість ручного проектування низькорівневих компонентів доступу (DAO) та написання громіздкого коду для формування запитів, фреймворк надає можливість декларативного опису інструментарію через стандартні Java-інтерфейси (зокрема, MongoRepository).

Завдяки вбудованому механізму об'єктно-документного відображення (ODM – Object-Document Mapping), Spring Boot автоматично генерує проксі-класи для реалізації цих інтерфейсів під час запуску контексту застосунку. Це повністю звільняє розробника від ручного написання складних запитів агрегації чи вибірки, мінімізує обсяг шаблонного коду та забезпечує стандартизовану, безпечну від ін'єкцій взаємодію з колекціями СУБД MongoDB.

Завдяки інтеграції Spring Boot та MongoDB система може:

- зберігати історію круток;
- отримувати статистику за певний період;

- виконувати пошук інформації;
- формувати результати аналізу;
- працювати з великими обсягами статистичних даних.

Використання інструментарію Spring Boot надає можливість створення централізованої архітектури доступу до даних, що суттєво підвищує ремонтопридатність кодової бази та спрощує процес еволюційного розвитку програмного забезпечення.

Інтеграція Spring Boot із СУБД MongoDB відкриває можливість побудови потужного механізму персистенції для зберігання та аналізу статистичних даних. Завдяки синхронізації об'єктної моделі застосунку (Java-класи), інтерфейсів доступу (Repositories) та безсхемної документно-орієнтованої структури бази даних досягається висока архітектурна гнучкість. Це дає змогу безперешкодно адаптувати модель даних до зміни функціональних алгоритмів і механік ігрових подій, уникнувши складних і ризикованих міграцій схем. Такий підхід значно полегшує реалізацію математичних моделей для аналізу систем випадкового заохочення.

Одним із ключових завдань розробленого програмного комплексу є виконання розширеного статистичного аналізу та стохастичної симуляції алгоритмів роботи механік випадкового заохочення (gacha). У зв’язку з цим, під час архітектурного проєктування та вибору технологічного стеку особлива увага приділялася спроможності системи ефективно обробляти великі масиви даних (Big Data) і виконувати інтенсивні обчислення на стороні сервера.

Фреймворк Spring Boot виявився оптимальним вибором для таких завдань завдяки високій продуктивності віртуальної машини Java (JVM), підтримці багатопоточності та багатому інструментарію Java Collections Framework у поєднанні з чисельними математичними бібліотеками. Виконання аналітичних обчислень на сервері (Backend) дає змогу запускати ресурсоємні алгоритми агрегації та симуляції в оптимізованому середовищі. Цей підхід дозволяє уникнути перевантаження апаратних ресурсів клієнтських пристроїв,

забезпечуючи стабільну роботу користувацького інтерфейсу та незалежність виконання обчислень від продуктивності кінцевих терміналів.

У межах даного проєкту серверна частина може виконувати:

- обчислення середнього pity;
- аналіз результатів механізму 50/50;
- підрахунок статистики випадінь;
- формування розподілу круток;
- обробку великих вибірок статистичних даних;
- симуляцію отримання персонажів.

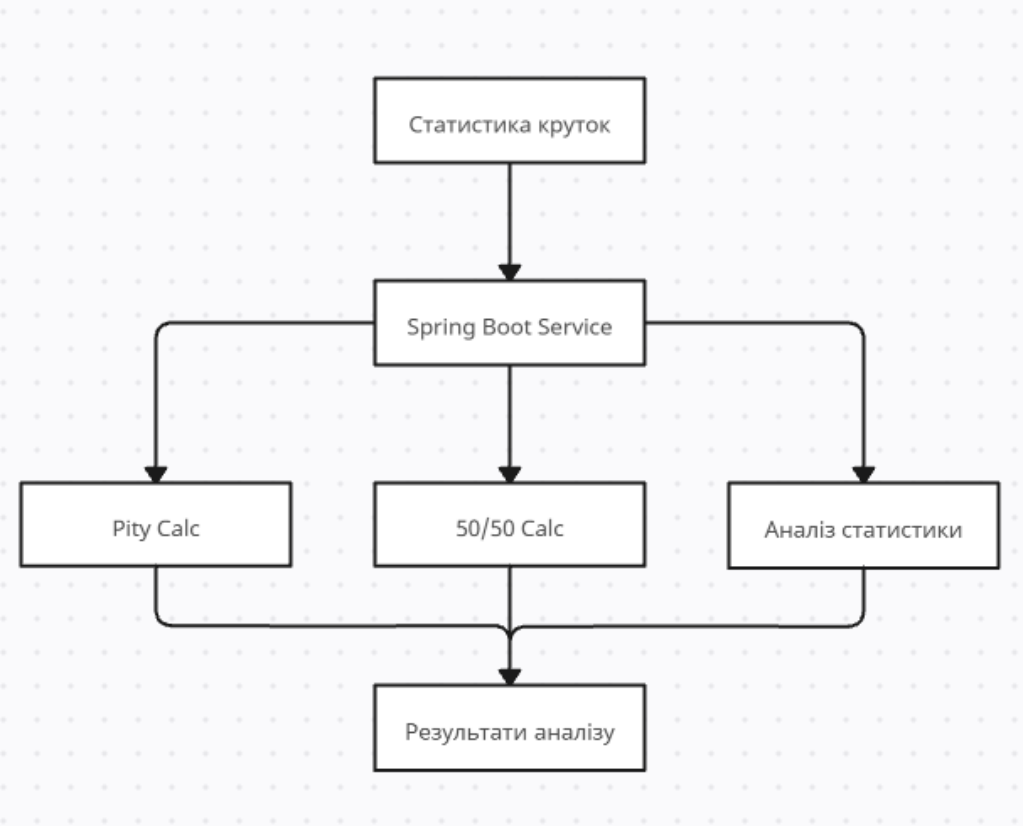

Рис. 4.7 – Схема процесу статистичного аналізу.

Важливою перевагою Spring Boot є підтримка багатошарової архітектури застосунку. Завдяки цьому модулі статистичного аналізу можуть бути реалізовані окремо від REST API та роботи з базою даних. Такий підхід спрощує підтримку системи та дозволяє поступово додавати нові алгоритми аналізу або симуляції.

Оскільки основною мовою програмування Spring Boot є Java, система отримує доступ до великої кількості бібліотек для математичних обчислень, роботи з колекціями даних та обробки статистики. Це дозволяє реалізовувати складніші алгоритми аналізу без необхідності використання окремих зовнішніх сервісів.

Spring Boot також забезпечує зручну роботу з великими обсягами даних. У разі накопичення значної кількості статистики сервер може виконувати обробку інформації поетапно, формувати результати аналізу та повертати готові дані клієнтській частині у форматі JSON.

Окрему роль у дослідженні відіграють симуляції роботи гача-системи. Серверна частина може багаторазово моделювати процес виконання круток із врахуванням pity, soft pity та механізму 50/50. Це дозволяє оцінити середню кількість круток, ймовірність отримання персонажа та інші характеристики системи без необхідності використання реальних ресурсів гри.

Таким чином, використання Spring Boot забезпечує зручну платформу для реалізації статистичного аналізу та симуляцій гача-механік. Поєднання серверної логіки, роботи з базою даних та можливостей обробки статистики дозволяє створити ефективну систему дослідження механізмів випадкових винагород у відеоіграх.

2.3 Обґрунтування вибору MongoDB

Для зберігання статистичних даних у межах програмної системи було обрано MongoDB – документно-орієнтовану NoSQL базу даних. Дане рішення пов’язане з особливостями структури інформації, яка використовується під час аналізу гача-механік.

Статистика круток, результати механізму 50/50, інформація про pity та інші параметри можуть мати різну структуру та змінюватися залежно від типу банера або способу аналізу даних. Саме тому використання класичних

реляційних баз даних не завжди є найбільш зручним рішенням для подібних задач.

MongoDB дозволяє зберігати інформацію у вигляді JSON-подібних документів, що добре підходить для роботи зі статистичними даними та результатами симуляцій. Завдяки цьому система отримує гнучку структуру зберігання інформації та можливість швидко змінювати або доповнювати формат даних без складного перепроєктування бази даних.

Ще однією важливою перевагою MongoDB є висока швидкість роботи з великими обсягами даних. Оскільки система аналізу гача-механік може накопичувати значну кількість статистики круток та результатів симуляцій, ефективність роботи бази даних є важливим фактором для забезпечення стабільної роботи застосунку.

Рис. 2.8 – Логотип MongoDB.

У цьому проєкті MongoDB виконує роль зберігання історії круток, статистики персонажів, результатів аналізу та інших даних, які забезпечують дослідження функціонування гача-системи. Завдяки інтеграції MongoDB зі

Spring Boot можна автоматизувати роботу з документами та спростити зв’язок серверної частини застосунку з базою даних.

У подальших розділах буде докладно розглянуто принципи функціонування NoSQL баз даних, специфіку документно-орієнтованої моделі MongoDB, а також переваги її застосування для обробки та зберігання статистичних даних.

MongoDB належить до класу нереляційних систем керування базами даних (NoSQL). На відміну від класичних реляційних СУБД (RDBMS), які базуються на строгій реляційній алгебрі та потребують попереднього проектування жорстко фіксованих табличних схем із суворими зв'язками, NoSQL-рішення реалізують концепцію динамічних схем даних (schema-less). Це дозволяє оперувати гнучкими, поліморфними та слабкоструктурованими масивами інформації, що є критично важливим під час обробки динамічних потоків даних.

Домінантним архітектурним принципом документно-орієнтованого підходу, покладеного в основу MongoDB, є агрегація даних у логічні сутності – колекції, які є аналогами таблиць у реляційних базах, проте не накладають обмежень на структуру своїх елементів. На фізичному та логічному рівнях інформація організовується у вигляді BSON-документів (Binary JSON) – бінарного розширення формату JSON. Кожен такий документ є незалежною структурою типу «ключ-значення», що може містити довільну кількість полів, масивів та глибоко вкладених об'єктів. Це забезпечує високу локальність даних та дозволяє уникнути ресурсомістких операцій об'єднання таблиць (JOIN), характерних для реляційного підходу.

У класичних реляційних моделях даних архітектура сутностей підпорядковується правилам суворої типізації та нормалізації: структура таблиць має бути декларована до моменту персистенції, а кожен кортеж (рядок таблиці) зобов'язаний містити ідентичний набір атрибутів (стовбців), що призводить до надлишкового зберігання значень NULL у випадку

опціональних полів. На противагу цьому, документно-орієнтована архітектура MongoDB підтримує концепцію структурного поліморфізму: кожен окремий документ у межах однієї колекції може володіти унікальним набором атрибутів, динамічно адаптуючись під характер інформаційного об'єкта.

У контексті дослідження, спрямованого на аналіз і моделювання механік випадкових заохочень (gacha-систем), гнучкість стає ключовою архітектурною перевагою. Оскільки емпіричні дані, що генеруються для різних типів ігрових подій (банерів), суттєво відрізняються своїми математичними і алгоритмічними характеристиками, застосування єдиної реляційної моделі бази даних призвело б до значного ускладнення її архітектури. Використання MongoDB дозволяє працювати з диференційованими документами в рамках спільних аналітичних колекцій.

Зокрема:

- Документ події персонажів включає в себе логіку розподілу «50/50» та маркер гарантії для конкретного персонажа.
- Документ події зброї містить унікальні метрики ймовірнісного розподілу, такі як система вибору бажаної зброї через «Epitomized Path» або інші параметри, на кшталт «hard pity».
- Документ з результатами статистичної симуляції замість збереження окремих логів містить масиви агрегованих розподілів ймовірностей, які були отримані шляхом серверних обчислень.

Такий підхід усуває необхідність розділяти дані на велику кількість взаємопов’язаних таблиць, що істотно підвищує ефективність виконання аналітичних запитів до бази даних.

Приклад документа MongoDB:

```
{
"character": "Furina",
"pity": 76,
```

```
"result50_50": "win",
"banner": "event"
}
```


Суттєвою технічною перевагою NoSQL СУБД MongoDB є висока оптимізація для роботи з великими обсягами даних, що забезпечує значну пропускну здатність при виконанні операцій читання і запису. Швидка обробка досягається завдяки ефективним індексаційним механізмам, таким як B-tree індекси, використанню потужного рушія зберігання WiredTiger із функцією оперативного кешування, а також відсутності потреби у перевірці цілісності складних реляційних зв'язків, як-от зовнішні ключі, під час кожної транзакції. Все це робить MongoDB ідеальною системою для аналітичних платформ, які працюють із потужними потоками стохастичних даних.

Ще однією важливою особливістю MongoDB є високий рівень масштабованості. На відміну від реляційних баз даних, які зазвичай масштабуються вертикально, тобто через збільшення ресурсів одного сервера, архітектура MongoDB створена для горизонтального масштабування. Вбудована підтримка реплікації (Replica Sets) і шардингу (Sharding) дозволяє безперервно розширювати систему, додаючи нові обчислювальні вузли. Це критично важливо в умовах зростання кількості користувачів або обсягів даних, адже розширення можливе без зупинки роботи сервісу та без значних змін у структурі бази даних.

З точки зору системної інтеграції, MongoDB демонструє сильну синергію з форматом JSON. Оскільки серверна логіка на базі Spring Boot та RESTful API використовують JSON як стандартний формат даних для обміну між компонентами, документно-орієнтована природа MongoDB повністю уникає проблеми невідповідності об'єктно-реляційної моделі. Це означає, що дані не потребують додаткової трансформації, декомпозиції чи нормалізації. Пакети даних із REST-ендпоінтів можуть безпосередньо перетворюватись у

BSON-документи та записуватись у базу без значного обчислювального навантаження.

Отже, впровадження MongoDB у якості нереляційної СУБД дозволяє створити гнучкий, надійний та високопродуктивний компонент зберігання для аналітичних систем. Її здатність ефективно працювати з великомасштабними гетерогенними даними, підтримувати динамічну зміну схем без руйнівних модифікацій та забезпечувати горизонтальне масштабування робить MongoDB найкращим вибором для реалізації програмного забезпечення для аналізу гача-механік.

Однією з головних особливостей MongoDB є документно-орієнтована структура зберігання даних. На відміну від реляційних баз даних, де інформація організовується у вигляді таблиць і рядків, MongoDB використовує документи, об’єднані у спеціальні колекції.

Документи в MongoDB мають JSON-подібний формат та можуть містити різні набори полів. Завдяки цьому структура даних є значно гнучкішою порівняно з класичними SQL базами даних.

У межах системи аналізу гача-механік кожен документ може містити інформацію про окреме випадіння персонажа або результати певної серії круток. Наприклад, документ може зберігати ім’я персонажа, кількість круток до отримання, результат механізму 50/50 та тип банера.

Приклад документа статистики круток:

```
{
"character": "Raiden Shogun",
"pity": 78,
"result50_50": "null",
"guaranteed": true,
"bannerType": "event"
}
```

Однією з переваг документно-орієнтованої структури є можливість легко змінювати формат даних. Якщо в майбутньому виникне необхідність додати нові параметри, наприклад дату крутки або версію гри, це можна зробити без повної зміни структури бази даних.

У реляційних базах даних подібні зміни часто потребують створення нових таблиць або модифікації вже існуючих. У MongoDB нові поля можуть бути додані без складного перепроєктування системи.

Ще однією важливою перевагою є зручна робота зі вкладеними структурами даних. Наприклад, у межах одного документа можуть одночасно зберігатися результати декількох круток або статистика окремого банера.

Документно-орієнтований підхід демонструє високу ефективність у комбінації зі Spring Boot та REST API. Оскільки взаємодія між фронтендом і бекендом відбувається за допомогою JSON-документів, використання MongoDB значно скорочує необхідність у додаткових перетвореннях даних між різними компонентами системи.

Отже, документно-орієнтована архітектура бази даних MongoDB надає гнучкі можливості для збереження статистичної інформації, спрощує обробку даних зі змінною структурою та забезпечує ефективну реалізацію систем аналізу, таких як механізми гача.

Однією з ключових причин вибору MongoDB для розробки системи аналізу гача-механік стала її здатність ефективно обробляти великі обсяги статистичних даних. У ході аналізу круток система поступово накопичує значні масиви інформації: результати випадінь, значення pity, статистику механізму 50/50, дані різних банерів, а також підсумки симуляцій.

Для таких завдань надзвичайно важливо забезпечити швидке збереження даних, стабільну роботу бази та оперативний доступ до необхідної аналітики. Завдяки документно-орієнтованому підходу та оптимізації роботи з

великими колекціями даних MongoDB прекрасно підходить для реалізації цих вимог.

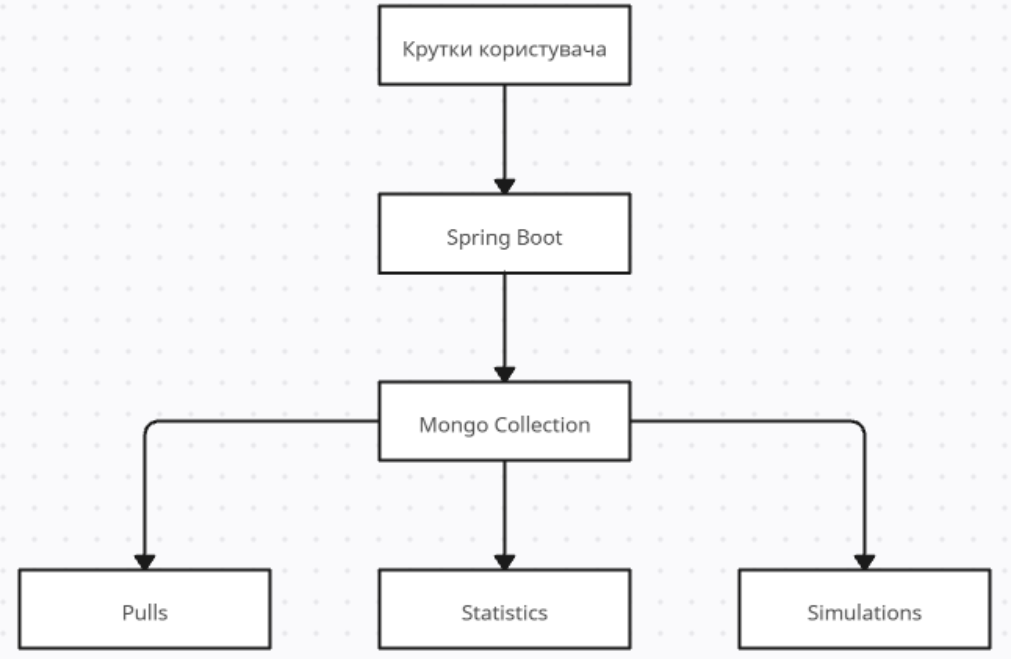

Рис. 4.9 – Схема накопичення статистичних даних у MongoDB.

У межах архітектури розробленого програмного комплексу кожна окрема спроба персистується в базі даних як автономний, атомарний документ. У результаті безперервного транзакційного логування дій користувача СУБД динамічно акумулює структурований масив релевантної статистичної інформації.

Накопичений масив даних формує репрезентативну емпіричну вибірку, яка згодом виступає обчислювальним базисом для проведення статистичного аналізу. Використовуючи вбудований аналітичний інструментарій MongoDB (Aggregation Framework), система здійснює конвеєрну обробку, фільтрацію та групування цих документів на серверній стороні застосунку. Це дозволяє в реальному часі розраховувати параметри математичного розподілу випадкових подій, оцінювати коректність функціонування механізмів pity та верифікувати теоретичні ймовірнісні моделі гача-механіки на основі реальних користувацьких даних.

Важливою перевагою MongoDB є можливість швидкого пошуку та фільтрації даних. Наприклад, система може окремо отримувати:

- статистику певного персонажа;
- результати конкретного банера;
- дані за визначений період часу;
- інформацію про виграші та програші 50/50;
- результати симуляцій.

Горизонтальна масштабованість бази даних MongoDB є ключовим архітектурним компонентом при роботі з великими масивами статистичних даних (Big Data). У випадках, коли відбувається лінійне або експоненціальне зростання кількості користувачів, а також при значному збільшенні обсягів накопичуваних даних, система може розширювати свою пропускну здатність шляхом розподілу даних між вузлами кластера (шардинг). Такий підхід дозволяє масштабувати інфраструктуру без необхідності змінювати архітектуру програми чи втручатися в її код.

Ще одним важливим аспектом є висока продуктивність системи при роботі в умовах значного навантаження на запис (Write-Heavy workload). Оскільки логи взаємодій користувачів (наприклад, ігрові спроби) можуть надходити у вигляді великих асинхронних потоків даних, архітектура MongoDB оптимізована для їхнього безперервного і швидкого збереження. Це стало можливим завдяки ефективній організації пам'яті та зведенню до мінімуму накладних витрат на транзакції, що запобігає втраті продуктивності та перевантаженню обчислювальних ресурсів навіть під час пікових навантажень.

Крім функцій зберігання емпіричних даних, MongoDB також використовується як аналітична база для задач імітаційного моделювання та проведення симуляцій. У рамках досліджень база даних застосовується для організації і збереження результатів роботи стохастичних симуляторів, що

функціонують за принципами обчислювальних алгоритмів, як-от метод Монте-Карло. Завдяки своїй масштабованості система може оперативно обробляти вибірки, які включають мільйони ітерацій, зберігати ці дані у вигляді спеціалізованих колекцій та використовувати їх для поглибленого аналізу, створення прогнозних моделей та перевірки відповідності отриманих емпіричних результатів з теоретично очікуваними розподілами ймовірностей.

Завдяки поєднанню високої швидкості роботи, гнучкої структури документів та підтримки масштабування MongoDB є зручним рішенням для систем, які працюють із великими масивами статистичних даних.

Таким чином, використання MongoDB дозволяє ефективно організувати процес зберігання, пошуку та аналізу великої кількості статистичної інформації, необхідної для дослідження роботи гача-механік.

Однією з ключових особливостей MongoDB є використання JSON-подібного формату для зберігання інформації. У MongoDB дані організовуються у вигляді документів BSON (Binary JSON), які є розширеною версією стандартного JSON-формату.

Використання JSON-документів є особливо зручним у межах системи аналізу гача-механік, оскільки статистичні дані природно представляються у вигляді структурованих об’єктів. Наприклад, інформація про персонажа може містити ім’я, кількість круток, результат механізму 50/50 та тип банера.

Приклад JSON-документа статистики:

```
{
"character": "Nahida",
"pity": 74,
"result50_50": "win",
"bannerType": "event",
"year": 2024
}
```

Однією з переваг JSON-документів є їхня гнучкість. На відміну від реляційних баз даних, MongoDB не вимагає жорстко визначеної структури таблиць. Завдяки цьому різні документи можуть містити різний набір полів.

Наприклад, один документ може зберігати лише базову інформацію про крутку, тоді як інший додатково міститиме результати симуляції або статистичний аналіз.

JSON-формат також добре підходить для взаємодії між frontend та backend частинами системи. Оскільки REST API у Spring Boot використовує JSON для передачі даних, інформація може передаватися між компонентами системи практично без додаткових перетворень.

Ще однією перевагою є зручність читання та редагування JSON-документів. Структура даних є зрозумілою навіть без складних SQL-запитів, що спрощує процес налагодження та тестування застосунку.

У межах системи JSON-документи можуть використовуватися для:

- збереження статистики круток;
- передачі результатів аналізу;
- роботи з REST API;
- зберігання результатів симуляцій;
- формування відповідей серверної частини застосунку.

Приклад JSON-відповіді REST API:

```
{
"totalPulls": 3200,
"averagePity": 72.25,
"wins50_50": 11,
"loses50_50": 13
}
```

Завдяки використанню JSON-документів система отримує гнучку структуру зберігання інформації, зручний обмін даними між компонентами та можливість легко масштабувати формат статистичних даних у майбутньому.

Таким чином, робота з JSON-документами є однією з важливих переваг MongoDB та суттєво спрощує реалізацію системи аналізу гача-механік.

2.4 Обґрунтування вибору Docker

Для забезпечення стабільної роботи та зручного розгортання програмної системи в межах проєкту використовується технологія Docker. Завдяки контейнеризації вдається ізолювати окремі компоненти застосунку і запускати їх у спеціальному середовищі, яке вже містить усі необхідні залежності та налаштування.

У контексті аналізу гача-механік Docker застосовується для роботи серверної частини системи, бази даних MongoDB і додаткових компонентів. Такий підхід допомагає уникнути труднощів, пов’язаних із відмінностями в операційних системах, версіях бібліотек чи параметрах конфігурацій.

Ключовою перевагою Docker є можливість швидкого та комфортного розгортання застосунку. Усі потрібні компоненти можна запустити за допомогою контейнерів, що виключає необхідність ускладненого ручного налаштування. Це суттєво полегшує тестування, перенесення та введення проєкту в експлуатацію на різних пристроях.

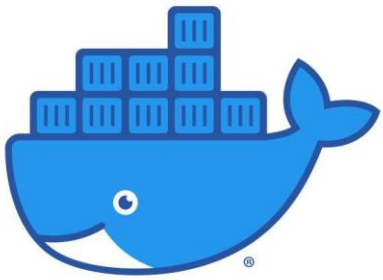

Рис. 2.10 – Логотип Docker.

Крім цього, Docker забезпечує ізоляцію компонентів системи. Наприклад, серверна частина Spring Boot та база даних MongoDB можуть працювати в окремих контейнерах, не впливаючи безпосередньо один на одного. Такий підхід підвищує стабільність роботи застосунку та спрощує підтримку проєкту.

У наступних підпунктах буде детально розглянуто поняття контейнеризації, принцип ізоляції середовища виконання, автоматизацію запуску системи та переваги використання Docker для розгортання програмного забезпечення.

Контейнеризація є сучасним підходом до запуску програмного забезпечення, який дозволяє ізолювати застосунок разом з усіма необхідними компонентами в окремому середовищі виконання – контейнері. У межах даного проєкту контейнеризація реалізується за допомогою технології Docker.

Основна ідея контейнеризації полягає в тому, що застосунок запускається не безпосередньо в операційній системі користувача, а всередині спеціального контейнера, який містить необхідні бібліотеки, налаштування та залежності. Завдяки цьому програма працює однаково незалежно від комп’ютера або операційної системи.

На відміну від класичних віртуальних машин, контейнери не потребують окремої повної операційної системи для кожного середовища. Усі контейнери використовують спільне ядро операційної системи, що дозволяє значно зменшити використання ресурсів комп’ютера та прискорити запуск застосунків.

У межах системи аналізу гача-механік контейнеризація дозволяє ізолювати серверну частину Spring Boot, базу даних MongoDB та інші компоненти системи. Кожен контейнер виконує власну функцію та може запускатися незалежно від інших частин застосунку.

Однією з головних переваг контейнеризації є стабільність середовища виконання. Усі необхідні залежності, бібліотеки та налаштування вже знаходяться всередині контейнера, тому система працює однаково на різних комп’ютерах.

Без Docker:

- Проблеми з версіями бібліотек;
- Різні налаштування системи;
- Помилки запуску.

З Docker:

- Однакове середовище;
- Стабільний запуск;
- Просте розгортання.

Контейнеризація значно полегшує тестування та перенесення проєкту. Наприклад, запуск застосунку на іншому комп’ютері можна здійснити без необхідності вручної перевстановлення всіх залежностей.

Однією з вагомих переваг Docker є здатність одночасно працювати з кількома контейнерами. Це дає змогу розділити систему на окремі модулі, сприяючи більш ефективній та гнучкій організації програмного забезпечення.

Отже, контейнеризація відіграє ключову роль у сучасній розробці програмних систем. Завдяки використанню Docker у цьому проєкті гарантовані стабільність роботи застосунку, ізольованість його компонентів і зручність розгортання системи аналізу гача-механік.

Однією з головних переваг Docker є ізоляція середовища виконання. Завдяки цьому підходу можна запускати окремі компоненти системи незалежно один від одного, розміщуючи їх у власних контейнерах з індивідуальними налаштуваннями та залежностями.

У межах проєкту, який передбачає аналіз гача-механік, система складається з кількох компонентів: серверної частини на базі Spring Boot, бази даних MongoDB та клієнтського застосунку. Кожен із цих елементів здатний функціонувати в окремому Docker-контейнері.

Концепція ізоляції середовища виконання (Runtime Isolation) полягає в тому, що кожен контейнер оперує в межах власного, відокремленого користувацького простору (User Space). Він володіє індивідуальною файловою системою, ексклюзивним набором системних бібліотек, конфігураційних файлів та ізольованим деревом процесів (PID Namespace). Завдяки такій архітектурній автономії, будь-які деструктивні зміни, збої або модифікації всередині одного контейнера не мають безпосереднього транзитивного впливу на інші функціональні компоненти розподіленої системи.

У контексті розгорнутої інфраструктури аналізу гача-механік це забезпечує повну технологічну незалежність шарів:

- Модернізація Backend-вузла: Оновлення версії виконання Java (наприклад, перехід на новішу версію LTS JDK), рефакторинг серверної логіки Spring Boot або оновлення внутрішніх залежностей виконуються виключно всередині контексту відповідного вебконтейнера. Це не вимагає переналаштування чи перезапуску суміжного контейнера СУБД MongoDB.

- Автономія шару персистенції: Зміна конфігурації СКБД, обслуговування індексів або оновлення версії образу MongoDB відбуваються ізольовано, що гарантує безперебійну доступність інтерфейсів клієнтської частини (Presentation Layer) таBackend-сервісів.

Фундаментальною перевагою такої ізоляції є забезпечення консистентності та високої відмовостійкості системи. У класичних монолітних архітектурах, що розгортаються безпосередньо на хостовій операційній системі, виникнення конфліктів між версіями спільних динамічних бібліотек або системних утиліт часто призводить до «конфігураційного дрейфу» та помилок ініціалізації сервісів (Dependency Hell).

Технологія Docker повністю нівелює цей ризик за рахунок принципу незмінної інфраструктури (Immutable Infrastructure): абсолютно всі залежності, бінарні файли та специфічні налаштування середовища ініціалізуються та фіксуються на етапі збирання Docker-образу. Це гарантує ідентичність поведінки застосунку незалежно від конфігурації хост-машини, на якій виконується розгортання кластера.

Без ізоляції:

- Конфлікти бібліотек;
- Помилки версій;
- Проблеми запуску.

З Docker:

- Ізольоване середовище;
- Стабільна робота;
- Незалежність компонентів.

Контейнеризація компонентів суттєво підвищує рівень тестованості системи (Testability). Завдяки ізоляції, кожен функціональний вузол (будь то аналітичний модуль на Spring Boot чи сховище даних MongoDB) можна автономно ініціалізувати, зупиняти, перезапускати або піддавати процедурі гарячого оновлення без деструктивного впливу на суміжні сервіси. У процесі

життєвого циклу розробки та налагодження (Debugging) це дозволяє ефективно впроваджувати практики інтеграційного тестування (наприклад, з використанням бібліотеки Testcontainers), імітуючи різні стани мережевої активності та інфраструктурних збоїв у повністю контрольованому середовищі.

З погляду інформаційної безпеки та системної стійкості, архітектура контейнеризації виступає надійним механізмом локалізації відмов (Fault Isolation Domain). Оскільки процеси всередині контейнера обмежені його віртуальним контуром, потенційні критичні збої, переповнення буфера або виняткові ситуації (Runtime Exceptions) в окремому модулі обробки статистики не спроможні спровокувати каскадне падіння всієї системи. Програмний комплекс зберігає часткову працездатність, що мінімізує показник середнього часу на відновлення сервісу (MTTR – Mean Time To Repair).

Таким чином, ізоляція середовища виконання є фундаментальним архітектурним елементом застосування технології Docker у межах цього проєкту. Вона гарантує детерміновану стабільність функціонування розподіленої системи, забезпечує високу технологічну автономність і слабку зв'язність інфраструктурних компонентів, а також оптимізує процеси верифікації, підтримки та супроводу програмного забезпечення.

Ключовою перевагою використання технології Docker у цьому проєкті є повна автоматизація процесів ініціалізації та розгортання програмної системи. Завдяки контейнеризації інфраструктура переводиться у декларативну площину, що дозволяє впроваджувати концепцію Infrastructure as Code (IaC). У результаті запуск усіх пов'язаних сервісів виконується за допомогою єдиного набору команд, виключаючи потребу в ручному налаштуванні та створенні цільового середовища.

У традиційних імперативних підходах до розгортання ПЗ системні адміністратори й розробники змушені виконувати велику кількість рутинних операцій. Це включає встановлення Java середовища виконання (JRE/JDK),

розгортання MongoDB, компіляцію та підключення зовнішніх бібліотек, а також ручну конфігурацію мережевих параметрів: проброс портів, мапування системних шляхів, ініціалізацію змінних оточення тощо. Такий спосіб роботи не лише обтяжує процес, а й піддає його ризикам через людські помилки, що можуть спричинити некоректну конфігурацію або розрив у послідовності середовищ розробки, тестування та продакшену.

Docker суттєво полегшує цей аспект шляхом ізоляції всіх необхідних налаштувань і залежностей у межах контейнерів. Системні компоненти, бінарні файли, мережеві правила та скрипти ініціалізації описуються в Docker-образах, а їх взаємодія визначається декларативно через файли специфікацій (наприклад, за допомогою Docker Compose). Це дозволяє розгортати багатокомпонентну систему за допомогою однієї команди, гарантуючи повну повторюваність і однаковість робочої інфраструктури на будь-якій обчислювальній платформі.

Для запуску системи достатньо виконати одну команду, після чого Docker автоматично створює та запускає всі необхідні контейнери. Завдяки цьому процес розгортання застосунку значно спрощується та займає менше часу.

Автоматизація запуску є особливо важливою під час тестування та демонстрації проєкту. Оскільки всі компоненти системи запускаються автоматично, ризик помилок через неправильне налаштування середовища суттєво зменшується.

- Без Docker:
- Ручне встановлення залежностей;
- Налаштування середовища;
- Запуск кожного компонента окремо.

З Docker:

- Автоматичний запуск;
- Готове середовище;

- Швидке розгортання системи.

Ще однією перевагою є можливість швидкого повторного запуску системи. У разі помилки або необхідності оновлення контейнер може бути автоматично перезапущений без складних дій з боку користувача.

Крім цього, Docker дозволяє легко переносити проєкт між різними комп’ютерами. Усі конфігурації запуску залишаються однаковими, що забезпечує стабільну роботу системи незалежно від операційної системи або особливостей середовища.

У межах системи аналізу гача-механік автоматизація запуску дозволяє швидко розгортати серверну частину Spring Boot, базу даних MongoDB та інші компоненти застосунку без додаткового ручного налаштування.

Таким чином, використання Docker значно спрощує процес запуску та розгортання програмної системи, забезпечує стабільність середовища виконання та зменшує кількість помилок, пов’язаних із конфігурацією застосунку.

3 РОЗРАХУНКОВИЙ РОЗДІЛ

3.1 Імовірнісний опис гача-системи

Для проведення математичного аналізу необхідно представити гача-систему гри Genshin Impact у вигляді формалізованої ймовірнісної моделі. З позиції теорії ймовірностей кожну спробу можна розглядати як окрему випадкову подію, результат якої визначається фіксованим розподілом ймовірностей випадіння елементів різного рівня рідкості.

У базовому варіанті кожну спробу можна моделювати як випробування з двома можливими наслідками: успіхом або невдачею. У межах цього дослідження під «успіхом» розуміється отримання легендарного персонажа, а під «невдачею» - будь-який інший альтернативний результат.

Нехай випадкова подія *A* полягає в отриманні п’ятизіркового персонажа в результаті однієї спроби. Тоді базова ймовірність цієї події для івентового банера персонажа становить 0,6% або:

$$\boldsymbol{P(A) = 0,006}$$

Відповідно, ймовірність протилежної події (неотримання п’ятизіркового персонажа за одну спробу) визначається як:

$$\boldsymbol{P(\overline{A}) = 1 - P(A) = 0,994}$$

Отже, кожну окрему крутку можна представити як випробування Бернуллі, де можливі лише два результати: успіх або невдача.

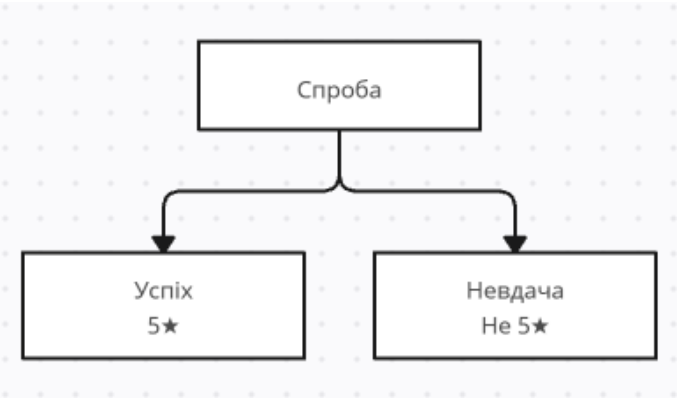

Рис. 3.1 – Схема одного випробування Бернуллі.

Якщо б у грі враховувалася лише базова ймовірність випадіння, то процес отримання персонажа можна було б описати за допомогою геометричного розподілу. Однак на практиці механіка Genshin Impact є значно складнішою через впровадження системи pity та гарантій.

Коли кількість невдалих спроб досягає певного порогу, ймовірність отримання п’ятизіркового персонажа поступово зростає, що порушує принцип незалежності подій. Тобто результат поточної спроби безпосередньо залежить від кількості попередніх невдач.

З погляду математики, така гача-система моделюється як стохастичний процес з внутрішніми станами, де станом є поточне значення pity. З кожною невдалою спробою значення pity збільшується, а після отримання п’ятизіркового персонажа воно скидається до вихідного рівня.

Крім pity-системи, на процес отримання персонажа впливає механізм 50/50. У цьому випадку після випадіння п’ятизіркового персонажа система додатково визначає, чи буде отримано івентового персонажа, чи одного зі стандартних персонажів. Таким чином, модель містить декілька рівнів випадкових подій, які необхідно враховувати під час математичного аналізу.

Для подальшого дослідження будемо розглядати гача-систему як послідовність випадкових подій із змінною ймовірністю успіху, що залежить від поточного значення pity та стану гарантії. Такий підхід дозволяє побудувати математичну модель, максимально наближену до реального механізму роботи гри.

3.2 Математична модель отримання п’ятизіркового персонажа

Для розробки математичної моделі розглянемо процес отримання п’ятизіркового персонажа в івентовому банері гри Genshin Impact як

сукупність незалежних спроб, або так званих "круток". Кожна крутка характеризується визначеною ймовірністю успішного результату, тобто випадінням п’тизіркового персонажа.

Якщо припустити, що pity-система не враховується, ймовірність неотримання п’ятизіркового персонажа за *n* послідовних круток описується наступним виразом:

$$\boldsymbol{P_0(n) = (1 - p)^n},$$

де:

- *p* – ймовірність отримання п’ятизіркового персонажа;
- *n* – кількість виконаних круток;
- $P_0(n)$ – ймовірність не отримати п’ятизіркового персонажа за *n* спроб.

В такому випадку ймовірність того, що принаймні один п’ятизірковий персонаж випаде протягом *n* круток, визначається формулою:

$$\boldsymbol{P_1(n) = 1 - (1 - p)^n}$$

Ця формула демонструє, як зі збільшенням кількості спроб поступово зростає шанс отримання п’ятизіркового персонажа.

Розглянемо конкретні приклади. Для 10 круток:

$$\boldsymbol{P_1(10) = 1 - (1 - 0,006)^{10} = 1 - 0,994^{10} \approx 0,058}$$

Згідно з простим підходом (без pity-системи), ймовірність отримати хоча б одного п’ятизіркового персонажа за 10 круток становить приблизно 5,8 %.

Для 70 круток:

$$\boldsymbol{P_1(70) = 1 - 0,994^{70} \approx 0,344}$$

Відповідно, без врахування pity-системи, ймовірність отримання щонайменше одного п’ятизіркового персонажа за 70 круток дорівнює приблизно 34,4 %.

Проте реальна механіка гри Genshin Impact є значно складнішою і відрізняється від цієї простої моделі із фіксованою ймовірністю. У межах pity-системи ймовірність випадіння п’ятизіркового персонажа починає плавно зростати після досягнення певного порогу, відомого як soft pity. В результаті модель із постійною ймовірністю не враховує цієї особливості, тому для точнішої оцінки необхідно розробити формулу, яка враховує змінну ймовірність події.

У такому випадку ймовірність не отримати п’ятизіркового персонажа після *n* круток можна подати у такій формі:

$$\boldsymbol{P_0(n) = \prod_{i=1}^{n}(1 - p_i)}$$

А ймовірність отримати хоча б одного п’ятизіркового персонажа після *n* круток вираховується наступним чином:

$$\boldsymbol{P_1(n) = 1 - \prod_{i=1}^{n}(1 - p_i)}$$

Де p_i – ймовірність отримання п’ятизіркового персонажа на і-й крутці.

Ця формула краще описує реальну гача-систему, оскільки дозволяє враховувати зміну ймовірності після накопичення pity.

Для івентового банера персонажа можна умовно поділити крутки на три діапазони:

Діапазон круток	Характеристика
1-73	Базова ймовірність випадіння
74-89	Зона soft pity
90	Гарантоване отримання п’ятизіркового персонажа

У спрощеній математичній моделі приймається, що до 73-ї спроби ймовірність залишається сталою:

$$\boldsymbol{p_i = 0,006}$$

З 74-ї спроби значення ймовірності (p_i) починає поступово зростати, а на 90-й спробі досягає максимуму, гарантуючи отримання 5★ персонаж:

$$\boldsymbol{p_{90} = 1}$$

Таким чином, процес отримання п’ятизіркового персонажа складається з двох основних етапів: базової випадкової складової та механізму гарантії. Базова складова описує стандартні спроби з низькою початковою ймовірністю успіху, тоді як pity-система збільшує цю ймовірність після певної кількості невдалих спроб.

Цю модель можна використовувати для оцінки середньої кількості спроб, необхідних для отримання п’ятизіркового персонажа, а також для аналізу теоретичних результатів у порівнянні з реальною статистикою гри.

3.3 Аналіз механізму 50/50

Після отримання п’ятизіркового персонажа в івентовому банері система Genshin Impact виконує додаткову перевірку, відому серед гравців як механізм 50/50. Його призначення полягає у визначенні того, чи буде отримано івентового персонажа поточного банера, чи одного зі стандартних персонажів.

Позначимо подію отримання івентового персонажа як B. У такому разі ймовірність успіху становить 50%, або 0.5.

$$\boldsymbol{P(B) = 0,5}$$

Відповідно, ймовірність програшу в ситуації 50/50 також дорівнює 50%, або 0.5.

$$\boldsymbol{P(\overline{B}) = 1 - P(B) = 0,5}$$

Таким чином, у момент першого появлення персонажа з легендарною рідкістю обидва можливих результати є рівновірогідними.

Якщо гравець виграє в механізмі 50/50, то бажаний персонаж з'являється одразу після активації pity-системи. У разі програшу механізм гарантії активується й під час наступного випадіння п'ятизіркового персонажа цей персонаж обов'язково буде івентовим.

Тому для отримання конкретного персонажа можливі два основні сценарії:

1. Виграш 50/50 та отримання персонажа після першого п'ятизіркового.
2. Програш 50/50 та отримання персонажа після наступного п'ятизіркового.

Ймовірність отримати персонажа за перший цикл pity:

$$\boldsymbol{P_1 = 0,5}$$

Ймовірність того, що для отримання персонажа знадобиться два цикли pity:

$$\boldsymbol{P_2 = 0,5}$$

Таким чином, у половині випадків гравець отримує персонажа одразу, а в іншій половині необхідно накопичити ще один гарант.

Для оцінки середньої кількості циклів pity застосовується математичне сподівання:

$$\boldsymbol{M = 1 * 0,5 + 2 * 0,5 = 1,5}$$

Отримане значення показує, що в середньому для здобуття конкретного івентового персонажа потрібно 1,5 цикли pity.

Якщо врахувати, що максимальний гарант одного циклу становить 90 круток, то математичне сподівання кількості круток можна оцінити як:

$$\boldsymbol{N = 90 * 1,5 = 135}$$

За спрощеною моделлю, середня кількість круток для отримання певного івентового персонажа складає приблизно 135. Однак на практиці це

число зазвичай менше завдяки механізму soft pity, який підвищує ймовірність випадіння персонажа п’ятизіркового рівня до досягнення максимального гаранту.

У межах цього дослідження також буде проведено аналіз власної статистики круток, щоб порівняти теоретичні значення з реальними результатами. Це дозволить визначити, наскільки фактичний розподіл виграних і програних спроб відповідає заявленій імовірності у 50 %.

3.4 Оцінка очікуваної кількості круток

Одним із важливих показників ефективності гача-системи є кількість круток, необхідна для отримання бажаного івентового персонажа. Саме цей показник дозволяє оцінити обсяг ресурсів, який повинен накопичити гравець.

У межах івентового банера можливі різні сценарії отримання персонажа. У найбільш сприятливому випадку гравець отримує п’ятизіркового персонажа на ранніх крутках та одночасно виграє механізм 50/50. Тоді бажаний персонаж може бути отриманий практично одразу.

У найгіршому випадку гравець доходить до максимального гаранту, програє 50/50 та змушений накопичувати ще один гарантований цикл. Оскільки максимальний гарант для івентового банера становить 90 круток, загальна кількість необхідних круток може досягати 180.

Якщо врахувати, що половина гравців виграє 50/50, а інша половина змушена використовувати гарант, то середня кількість круток для отримання конкретного івентового персонажа зазвичай перевищує середню кількість круток до будь-якого п’ятизіркового персонажа приблизно у півтора раза.

Таким чином, для отримання бажаного персонажа гравцю необхідно орієнтуватися не лише на систему гаранту, але й на можливий результат

механізму 50/50. Це робить оцінку кількості необхідних круток важливим елементом аналізу ефективності гача-механіки.

Важливим показником ефективності гача-системи є кількість внутрішньоігрової валюти, необхідна для отримання бажаного персонажа.

Нагадування. Для виконання однієї крутки необхідно витратити 160 примогемів. Відповідно, вартість десяти круток становить 1600 примогемів.

Мінімальні витрати можуть становити лише декілька сотень примогемів, тоді як максимальні витрати дорівнюють:

$$\mathbf{180 * 160 = 28800}$$

Аналіз витрат внутрішньоігрової валюти дає змогу оцінити практичну ефективність гача-системи та визначити кількість ресурсів, необхідних для отримання бажаних персонажів. Отримані дані також можуть слугувати основою для зіставлення теоретичних прогнозів із фактичною статистикою здійснених круток.

3.5 Аналіз особистої статистики круток

Для перевірки отриманих теоретичних результатів було проведено аналіз особистої статистики круток в грі Genshin Impact. Використання реальних даних дозволило не лише наочно підтвердити практичну цінність побудованої математичної моделі, а й оцінити її відповідність фактичним результатам.

У межах дослідження було проаналізовано історію прокруток івентового банера персонажів за період із березня 2021 року до квітня 2026 року. До вибірки увійшли всі отримані п’ятизіркові персонажі, разом із інформацією про кількість круток до їх здобуття, результати механізму 50/50 та наявність гарантії.

Загалом було розглянуто 36 випадків отримання п’ятизіркових персонажів. Зібрана статистика охоплює понад п’ять років активної гри,

забезпечуючи значний обсяг даних для подальшого аналізу. Це дозволяє дослідити особливості функціонування механізмів pity та 50/50 не лише з теоретичної, а й з практичної точки зору.

У цьому розділі проведено аналіз розподілу кількості круток до отримання п’ятизіркових персонажів, вивчено роботу механізму 50/50, оцінено вплив pity-системи та виявлено загальні закономірності, що випливають із аналізу статистичних даних.

Результати дослідження будуть зіставлені з теоретичними показниками, розглянутими в попередніх підрозділах, що дозволить оцінити ефективність та передбачуваність гача-системи Genshin Impact.

Одним із основних показників ефективності гача-системи є кількість круток, необхідних для отримання п’ятизіркового персонажа. Саме цей показник дозволяє оцінити вплив механізму pity на процес отримання рідкісних персонажів та визначити, наскільки фактичні результати відповідають теоретичним очікуванням.

№	Дата	Банер персонажа	Персонаж, який випав	Крутка	Гарант	50/50
1	30.03.2021	Venti	Mona	77	Soft pity	Lose
2	28.04.2021	Zhongli	Zhongli	78	Soft pity	Guaranteed
3	18.05.2021	Zhongli	Zhongli	79	Soft pity	Win
4	30.06.2021	Kazuha	Kazuha	77	Soft pity	Win
5	18.07.2021	Kazuha	Kazuha	65	Mid pity	Win
6	11.08.2021	Yoimiya	Yoimiya	77	Soft pity	Win
7	14.09.2021	Raiden Shogun	Jinn	76	Soft pity	Lose
8	24.11.2021	Albedo	Albedo	81	Late pity	Guaranteed
9	14.12.2021	Itto	Itto	42	Early pity	Win
10	27.01.2022	Ganyu	Ganyu	77	Soft pity	Win
11	30.03.2022	Venti	Diluc	81	Late pity	Lose
12	30.03.2022	Venti	Venti	74	Mid pity	Guaranteed
13	31.05.2022	Yelan	Keqing	79	Soft pity	Lose
14	31.08.2022	Tignari	Tignari	80	Soft pity	Guaranteed
15	07.12.2022	Itto	Qiqi	56	Mid pity	Lose
16	07.12.2022	Itto	Itto	75	Mid pity	Guaranteed
17	07.12.2022	Itto	Itto	76	Soft pity	Win
18	28.12.2022	Raiden Shogun	Keqing	78	Soft pity	Lose
19	07.02.2023	Yelan	Yelan	76	Soft pity	Guaranteed
20	12.04.2023	Nilou	Dehya	83	Late pity	Lose
21	27.05.2023	Yoimiya	Yoimiya	78	Soft pity	Guaranteed
22	16.08.2023	Lyney	Jinn	74	Mid pity	Lose
23	26.09.2023	Tartaglia	Tartaglia	74	Mid pity	Guaranteed
24	13.10.2023	Hutao	Hutao	62	Mid pity	Win
25	20.12.2023	Navia	Diluc	77	Soft pity	Lose
26	01.01.2024	Navia	Navia	77	Soft pity	Guaranteed
27	13.03.2024	Chiori	Chiori	76	Soft pity	Win
28	25.07.2024	Nilou	Qiqi	49	Early pity	Lose
29	09.10.2024	Xilonen	Xilonen	21	Early pity	Guaranteed
30	01.01.2025	Mavuika	Mavuika	76	Soft pity	Win
31	02.01.2025	Citlali	Qiqi	39	Early pity	Lose
32	09.01.2025	Citlali	Citlali	75	Mid pity	Guaranteed
33	27.09.2025	Nahida	Keqing	75	Mid pity	Lose
34	10.11.2025	Nefer	Nefer	76	Soft pity	Guaranteed
35	16.01.2026	Columbina	Colombina	78	Soft pity	Win
36	16.04.2026	Zibai	Mizuki	81	Late pity	Lose

Рис. 3.2 – Особиста історія випадіння персонажів.

На основі особистої статистики було проаналізовано 36 випадків отримання п’ятизіркових персонажів. Для кожного випадку враховувалася кількість круток, виконаних з моменту попереднього отримання п’ятизіркового персонажа до наступного випадіння.

Показник	Значення
Кількість проаналізованих легендарок	36
Середня кількість круток	71,53
Мінімальна кількість круток	21
Максимальна кількість круток	83

Для знаходження середньої кількості круток для отримання п’ятизіркового персонажа використовуємо формулу середнього арифметичного числа:

$$\boldsymbol{S = \frac{\text{сума круток}}{\text{сума кількості круток}} = \frac{2575}{36} \approx 71,53}$$

За результатами аналізу було встановлено, що середня кількість круток до отримання п’ятизіркового персонажа становить 71,53 крутки. Найменше зафіксоване значення дорівнює 21 крутці, а найбільше – 83 круткам. Таким чином, жодного разу не було досягнуто максимального гаранту в 90 круток.

Отримані результати свідчать про значний вплив механізму soft pity на роботу банера. Більшість випадінь п’ятизіркові персонажів припадає на діапазон від 74 до 79 круток, де ймовірність отримання персонажа починає суттєво зростати. Саме в цьому інтервалі спостерігається найбільша концентрація результатів.

Для більш детального аналізу всі випадіння були розподілені за категоріями pity.

Категорія	Кількість випадінь
Early pity	4
Mid pity	9
Soft pity	19
Late pity	4

З наведених даних видно, що найбільшу частку становлять випадіння в зоні soft pity – 19 із 36 отриманих персонажів. Це відповідає приблизно 52,8 % усіх спостережень. Водночас лише чотири персонажі були отримані на ранніх крутках, що підтверджує відносно низьку базову ймовірність випадіння п’ятизіркових персонажів.

Окремий інтерес викликають випадки раннього отримання персонажів. У дослідженій вибірці найменше число становило 21 крутку під час отримання

персонажа Шилонен. Такі ситуації трапляються доволі рідко та пояснюються випадковістю, що властива гача-системі.

З іншого боку, максимальний показник досяг 83 круток. Попри те, що це значення наближається до порогу hard pity, навіть у цьому випадку персонажа вдалося отримати до активації максимальної гарантії.

Узагалі, аналіз персональної статистики підтверджує, що більшість п’ятизіркових персонажів випадають у зоні soft pity. Отримані результати узгоджуються як із теоретичною моделлю, так і з даними, які публікує спільнота гравців Genshin Impact, підкреслюючи правильність раніше висунутих припущень щодо механізмів роботи гача-системи.

Проведемо аналіз особистої статистики виграшів та програшів 50/50.

Результат	Кількість	Частка, %
Виграш	11	45,8
Програш	13	54,2

У ході дослідження було зафіксовано 24 спрацювання механізму 50/50. Із них 11 випадків завершилися отриманням івентового персонажа, тоді як у 13 випадках було отримано стандартного персонажа, після чого активувався гарант.

Отримані результати демонструють незначне переважання програшів над виграшами. Частка виграшів склала 45,8%, а частка програшів – 54,2%. Незважаючи на різницю в декілька відсотків, дані значення залишаються близькими до теоретичного співвідношення 50% на 50%.

Відхилення від ідеального розподілу пояснюється випадковим характером роботи системи та обмеженим обсягом вибірки. Навіть за великої кількості круток фактичні результати рідко збігаються з теоретичними значеннями повністю.

Варто зазначити, що кожен програш 50/50 у подальшому призводив до активації гаранту, завдяки чому наступний п’ятизірковий персонаж

гарантовано був івентовим. Саме тому механізм 50/50 не збільшує нескінченно кількість необхідних круток, а лише впливає на швидкість отримання бажаного персонажа.

Таким чином, результати аналізу підтверджують відповідність особистої статистики заявленому принципу роботи механізму 50/50. Отримані значення можуть вважатися достатньо близькими до теоретичних та свідчать про відсутність суттєвих відхилень у роботі системи.

Для аналізу роботи даного механізму було досліджено всі випадки отримання п’ятизіркових персонажів із поділом на три категорії:

- виграш 50/50;
- програш 50/50;
- отримання персонажа по гаранту після попереднього програшу.

Тип отримання	Кількість
Виграш	11
Програш	13
Гарантований персонаж	12

З отриманих даних видно, що найбільшу частку становлять програші 50/50 – 13 випадків із 36. Водночас було зафіксовано 12 гарантованих отримань івентових персонажів після попереднього програшу.

Фактично майже кожен третій отриманий п’ятизірковий персонаж був результатом роботи механізму гарантії. Це свідчить про значний вплив даної системи на кінцеві результати круток.

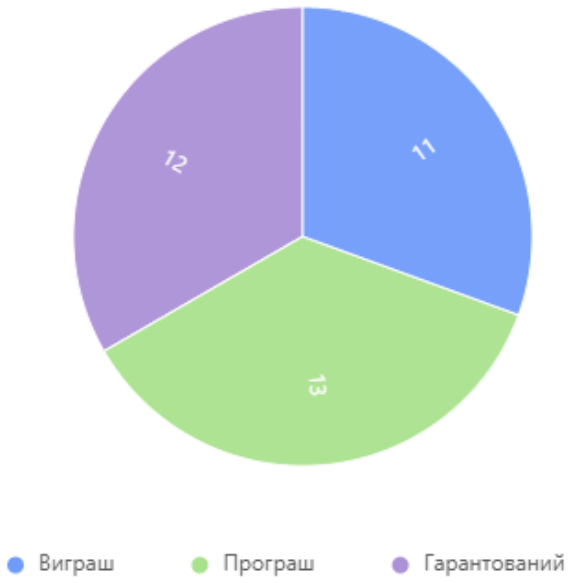

Рис. 3.3 – Діаграма розподілу результатів.

Особливу увагу привертає те, що кількість гарантованих персонажів майже дорівнює кількості програшів 50/50. Такий результат є очікуваним, оскільки після кожного програшу система повинна забезпечити отримання івентового персонажа під час наступного випадіння п’ятизіркового персонажа.

Без існування гаранту гравець міг би програвати 50/50 необмежену кількість разів поспіль, що значно збільшувало б необхідну кількість круток для отримання бажаного персонажа. Наявність гарантії робить систему більш передбачуваною та дозволяє оцінити максимальні витрати ресурсів.

Таким чином, аналіз особистої статистики підтверджує ефективність механізму гарантії. Незважаючи на наявність випадкової складової у вигляді 50/50, система забезпечує отримання івентового персонажа після невдалого результату, що суттєво знижує ризики для гравця та робить процес накопичення ресурсів більш прогнозованим.

У рамках цього дослідження було здійснено аналіз персональної статистики витрат круток в грі Genshin Impact за період із 2021 до 2026 року. Розглянуто 36 випадків отримання персонажів легендарного рівня, а також детально проаналізовано роботу механізмів pity, 50/50 та гарантії.

За результатами аналізу встановлено, що середня кількість круток, необхідна для отримання п’ятизіркового персонажа, становить 71,53. Це значення наближається до зони soft pity, що підтверджує вагомий вплив даного механізму на частоту випадінь. Жодного разу не було досягнуто максимального гаранту в 90 круток – більшість персонажів випадали до активації цього механізму.

Дослідження механізму 50/50 показало, що ймовірність виграшу становила 45,8%, а програшу – 54,2%. Незважаючи на невелике відхилення від теоретичного співвідношення 50% на 50%, отримані дані можна вважати близькими до очікуваних. Це свідчить про те, що особиста статистика узгоджується із заявленими розробниками принципами роботи системи.

Окремо була оцінена роль механізму гарантії. Результати дослідження засвідчили, що після кожного програшу в 50/50 система гарантії забезпечувала отримання івентового персонажа при наступному випадінні легендарного персонажа. Завдяки цьому рівень випадковості значно зменшувався, а отримання бажаного персонажа ставало більш передбачуваним.

Аналіз також продемонстрував, що більшість випадінь відбуваються в діапазоні soft pity. Це відповідає побудованій математичній моделі і збігається з даними, які публікує геймерське співтовариство. Таким чином, фактичні результати підтверджують припущення щодо механізмів роботи гача-системи.

Загалом проведене дослідження доводить, що механізми pity, 50/50 та гарантії забезпечують збалансоване співвідношення між випадковістю та прогнозованістю отримання персонажів. Незважаючи на елемент шансів, гравець може досить точно розрахувати необхідну кількість круток та ресурсів для отримання персонажа.

Результати аналізу демонструють коректність побудованої математичної моделі й потенціал її використання для подальшого дослідження гача-механік

не тільки у грі Genshin Impact, але й у інших сучасних відеоіграх із подібними системами розподілу винагород.

3.6 Опис веб-застосунку Genshin Analytics

Система виконує аналітику історії круток (Wish Pulls) у Genshin Impact. Дані зберігаються в MongoDB та аналізуються сервісом AnalyticsService. Користувач може фільтрувати результати за playerId, bannerName та heroName.

У системі використовується AnalyticsController з базовим URL /api/analytics.

Endpoint	**Аналітика**	**Що повертає**
/probability/any-5-star	Ймовірність випадіння будь-якого 5★	ProbabilityResponse
/probability/specific-hero	Ймовірність випадіння конкретного героя	ProbabilityResponse
/charts/five-star-hero-pie	Розподіл 5★ героїв	HeroPieChartResponse
/charts/five-star-by-pity	Ймовірність 5★ залежно від pity	PityAnalyticsResponse
/charts/featured-5-pity-cdf	CDF для featured 5★	FeaturedPityDistributionResponse
/math/expected-pulls-5-star	Математичне сподівання до 5★	MetricResponse
/math/variance-pulls-5-star	Дисперсія до 5★	MetricResponse
/math/expected-pulls-featured-5-limit	Середня кількість круток до featured 5★	MetricResponse
/math/variance-pulls-featured-5-limit	Дисперсія featured 5★	MetricResponse

1. Ймовірність будь-якого 5★ персонажа обчислюється як відношення кількості випадінь 5★ до загальної кількості круток.

2. Ймовірність конкретного героя визначається для вибраного heroName.

3. Діаграма Hero Pie/Bar агрегує всі випадіння 5★ персонажів та показує їх відносну частоту.

4. Pity Analytics аналізує, на якому pity найчастіше випадають 5★ персонажі.

5. Featured 5★ CDF відображає накопичувальну функцію розподілу для featured персонажів.

6. Expected Pulls використовує формулу $E[X]=1/p$.

7. Variance використовує формулу $Var(X)=(1-p)/p^2$.

Frontend реалізований на React. Головна сторінка містить:

- форму фільтрів (playerId, bannerName, heroName);
- картки KPI;
- графік розподілу 5★ героїв;
- графік pity-аналітики;
- графік CDF для featured 5★.

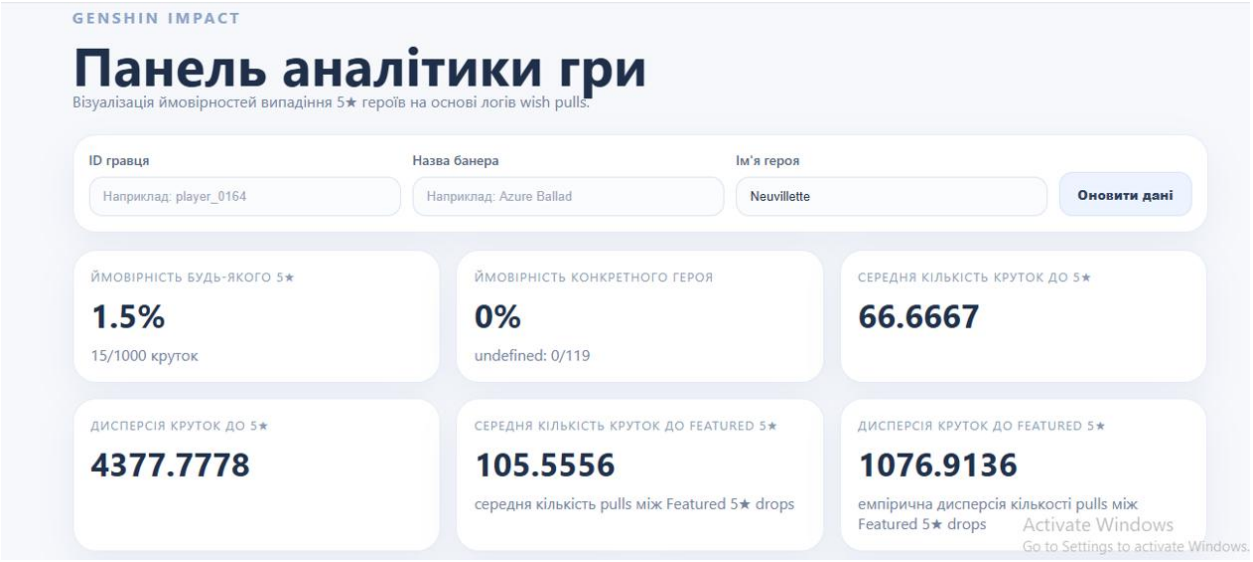

Рис. 3.4 – Інтерфейс панелі аналітики гача-системи

У блоці KPI відображаються:

- ймовірність будь-якого 5★;
- ймовірність конкретного героя;
- середня кількість круток до 5★;

- дисперсія;
- середня кількість круток до featured 5★;
- дисперсія featured 5★.

Рис. 3.5 – Візуалізація статистичних даних у системі аналітики

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Аналіз умов праці під час розробки програмного забезпечення

Під час розробки програмного забезпечення основним робочим інструментом є персональний комп’ютер. Робота програміста або розробника програмних систем пов’язана з тривалим перебуванням за монітором, виконанням великої кількості операцій із введення та обробки інформації, аналізом даних та постійним зоровим і розумовим навантаженням. Саме тому під час організації робочого процесу важливо враховувати умови праці та фактори, які можуть впливати на стан здоров’я працівника.

У межах даного проєкту розробка системи аналізу гача-механік виконується з використанням персонального комп’ютера та сучасних програмних засобів, зокрема середовищ розробки, серверних технологій та баз даних. Основна частина роботи пов’язана з написанням коду, аналізом статистичних даних, роботою з документацією та тестуванням програмної системи.

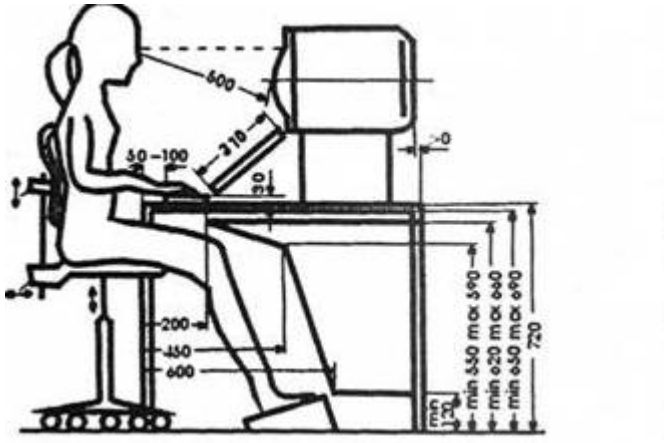

Рис. 4.1 – Рекомендована організація робочого місця.

На рисунку 4.1 представлено рекомендовану організацію робочого місця під час роботи за персональним комп’ютером. Правильне розташування

монітора, клавіатури та робочого крісла дозволяє зменшити навантаження на органи зору та опорно-руховий апарат. Відстань до монітора повинна становити приблизно 50–70 см, а положення рук і ніг має забезпечувати комфортну роботу протягом тривалого часу.

Під час розробки програмного забезпечення також значне навантаження припадає на нервову систему. Робота з кодом, пошук помилок, аналіз статистики та тривала концентрація уваги можуть викликати психологічну втому та зниження працездатності. Саме тому важливу роль відіграє правильна організація режиму праці та відпочинку.

Для забезпечення комфортних умов праці необхідно дотримуватися вимог щодо організації робочого місця. Робоче приміщення повинно мати достатній рівень освітлення, вентиляцію та оптимальні параметри мікроклімату. Температура повітря в приміщенні повинна залишатися комфортною для тривалої роботи, а рівень шуму не повинен перевищувати допустимі норми.

Параметр	**Рекомендоване значення**
Температура повітря	20-24°C
Вологість повітря	40-60%
Освітленість робочого місця	300-500 лк

Окрему увагу слід приділити ергономіці робочого місця. Монітор повинен розташовуватися на відстані приблизно 50–70 см від очей користувача, а верхня межа екрана — на рівні очей або трохи нижче. Крісло має забезпечувати підтримку спини та дозволяти підтримувати правильне положення тіла під час роботи.

Для зниження негативного впливу роботи за комп’ютером рекомендується регулярно робити короткі перерви, виконувати вправи для очей та змінювати положення тіла протягом робочого дня. Подібні заходи дозволяють зменшити втому та підвищити ефективність роботи.

Таким чином, під час розробки програмного забезпечення важливо враховувати умови праці та дотримуватися вимог охорони праці. Правильна

організація робочого місця, контроль мікроклімату та дотримання режиму праці й відпочинку дозволяють знизити вплив шкідливих факторів та забезпечити комфортні умови роботи розробника програмного забезпечення.

4.2 Шкідливі та небезпечні фактори під час роботи за комп’ютером

Під час роботи за персональним комп’ютером працівник може зазнавати впливу різних шкідливих та небезпечних факторів. Особливо актуальним це є під час розробки програмного забезпечення, коли значна частина робочого часу проходить перед монітором у сидячому положенні та потребує тривалої концентрації уваги.

До основних шкідливих факторів під час роботи за комп’ютером належать:

- навантаження на органи зору;
- статичне навантаження на опорно-руховий апарат;
- нервово-емоційне напруження;
- недостатня фізична активність;
- вплив електромагнітного випромінювання;
- підвищена втомлюваність.

Одним із найбільш поширених факторів є навантаження на зір. Під час тривалої роботи з монітором очі постійно фокусуються на екрані, що може викликати втому, сухість очей, головний біль та зниження концентрації уваги. Особливо негативно впливають неправильне освітлення, надмірна яскравість екрана та тривала робота без перерв.

Ще одним важливим фактором є статичне навантаження на м’язи та хребет. Під час роботи за комп’ютером людина тривалий час перебуває у сидячому положенні, що може призводити до болю у спині, шиї та руках.

Неправильна поза або невідповідне робоче місце значно збільшують ризик виникнення проблем із опорно-руховим апаратом.

Окрему увагу слід приділити нервово-емоційному навантаженню. Робота програміста часто пов’язана з високою концентрацією уваги, пошуком помилок у коді, аналізом даних та необхідністю тривалий час виконувати складні логічні операції. Це може викликати психологічну втому, стрес та зниження працездатності.

Під час роботи за комп’ютером також присутній вплив електромагнітного випромінювання, яке створюється електронним обладнанням. Сучасні монітори мають значно нижчий рівень випромінювання порівняно зі старими пристроями, однак тривала робота з електронною технікою все одно потребує дотримання правил безпечної організації робочого місця.

Недостатня рухова активність є ще одним негативним фактором. Тривале сидіння без фізичної активності може спричиняти погіршення кровообігу, втому та загальне зниження фізичного стану організму. Саме тому під час роботи за комп’ютером рекомендується регулярно робити перерви та виконувати легкі фізичні вправи.

Фактор	**Можливі наслідки**
Навантаження на зір	Втома очей, головний біль
Статичне навантаження	Біль у спині та шиї
Нервово-емоційне напруження	Стрес, перевтома
Недостатня рухова активність	Погіршення кровообігу
Електромагнітне випромінювання	Загальна втома

До небезпечних факторів також належать ризики, пов’язані з використанням електричного обладнання. Пошкоджені кабелі, несправні блоки живлення або перевантаження електромережі можуть призвести до короткого замикання чи ураження електричним струмом.

Для зменшення впливу шкідливих та небезпечних факторів необхідно дотримуватися правил охорони праці, правильно організовувати робоче місце, підтримувати оптимальні умови мікроклімату та регулярно робити перерви під час роботи.

Таким чином, під час роботи за комп’ютером на працівника можуть впливати різні шкідливі та небезпечні фактори, які здатні негативно впливати на здоров’я та працездатність. Дотримання правил охорони праці та правильна організація робочого процесу дозволяють суттєво знизити їх негативний вплив.

4.3 Вимоги до організації робочого місця

Правильна організація робочого місця є важливим елементом забезпечення безпечних та комфортних умов праці під час роботи за персональним комп’ютером. Особливо це актуально для розробників програмного забезпечення, оскільки значна частина робочого часу проходить у сидячому положенні перед монітором.

Основною метою правильної організації робочого місця є зниження навантаження на органи зору, опорно-руховий апарат та нервову систему працівника. Для цього необхідно враховувати вимоги до розташування обладнання, параметрів освітлення, мікроклімату приміщення та ергономіки робочого місця.

Робочий стіл повинен забезпечувати достатню площу для розміщення монітора, клавіатури, миші та іншого необхідного обладнання. Поверхня столу має бути стійкою та розташовуватися на зручній висоті для користувача.

Бажано, щоб робочий стіл мав матове покриття для зменшення відблисків від джерел світла.

Крісло повинно бути регульованим за висотою та забезпечувати підтримку спини. Під час роботи ноги користувача мають повністю стояти на підлозі або спеціальній підставці, а руки — розташовуватися у зручному положенні без надмірного навантаження на кисті та плечі.

Монітор необхідно розташовувати на відстані приблизно 50–70 см від очей користувача. Верхня межа екрана повинна знаходитися на рівні очей або трохи нижче. Подібне розташування дозволяє зменшити навантаження на шию та органи зору.

Клавіатура та миша повинні знаходитися на такій висоті, щоб руки користувача були зігнуті приблизно під кутом 90°. Це дозволяє знизити навантаження на суглоби та м’язи рук під час тривалої роботи.

Важливу роль відіграє освітлення робочого місця. Приміщення повинно мати достатній рівень природного або штучного освітлення. Джерела світла не повинні створювати відблиски на поверхні монітора, оскільки це може призводити до швидкої втоми очей.

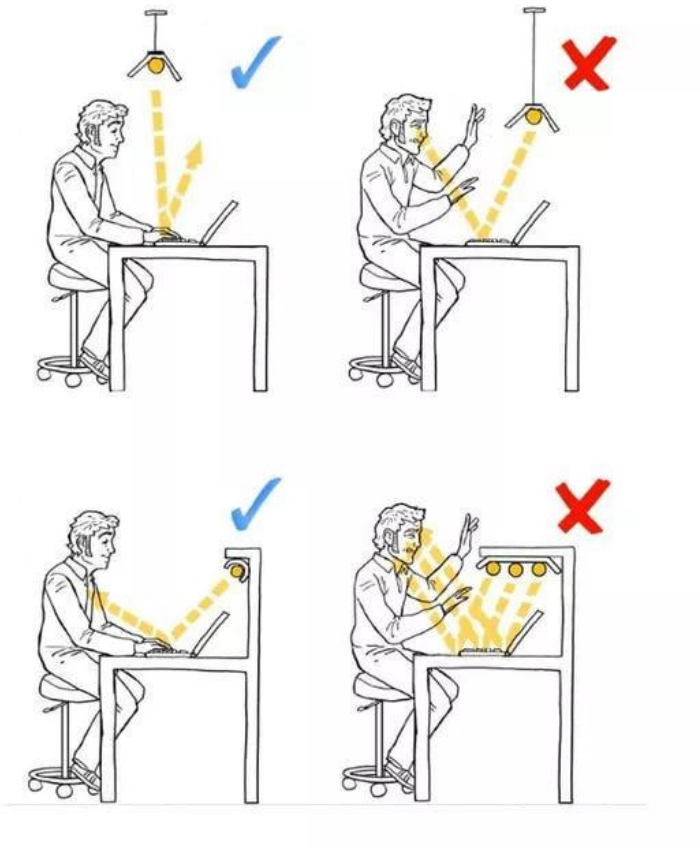

Рис. 4.2 – Правильне та неправильне освітлення робочого місця.

На рисунку 4.2 представлено приклади правильного та неправильного розташування джерел освітлення під час роботи за комп’ютером. Правильне освітлення дозволяє зменшити навантаження на органи зору та уникнути появи відблисків на екрані монітора. Джерела світла не повинні бути спрямовані безпосередньо в очі користувача або створювати яскраві відбиття на поверхні екрана.

Під час роботи за комп’ютером рекомендується регулярно робити короткі перерви для відпочинку очей та зміни положення тіла. Це дозволяє зменшити втому та підвищити ефективність роботи.

Окрему увагу необхідно приділяти безпечному розташуванню електричного обладнання. Кабелі живлення повинні бути ізольованими та не створювати ризику пошкодження або випадкового зачеплення.

Таким чином, правильна організація робочого місця є важливою умовою безпечної та ефективної роботи за комп’ютером. Дотримання ергономічних вимог дозволяє зменшити вплив шкідливих факторів та створити комфортні умови праці під час розробки програмного забезпечення.

4.4 Пожежна безпека в приміщенні

Під час роботи з комп’ютерною технікою важливу роль відіграє дотримання правил пожежної безпеки. У приміщеннях, де використовується велика кількість електронного обладнання, існує ризик виникнення пожеж через несправність електромережі, перевантаження обладнання або коротке замикання.

У межах розробки програмного забезпечення основними джерелами потенційної пожежної небезпеки є персональні комп’ютери, блоки живлення, мережеве обладнання, електричні кабелі та інші пристрої, підключені до електромережі.

Основними причинами виникнення пожеж у приміщеннях із комп’ютерною технікою можуть бути:

- коротке замикання електропроводки;
- перевантаження електромережі;
- використання несправного обладнання;
- пошкодження ізоляції кабелів;
- перегрів електронних компонентів;
- недотримання правил експлуатації електроприладів.

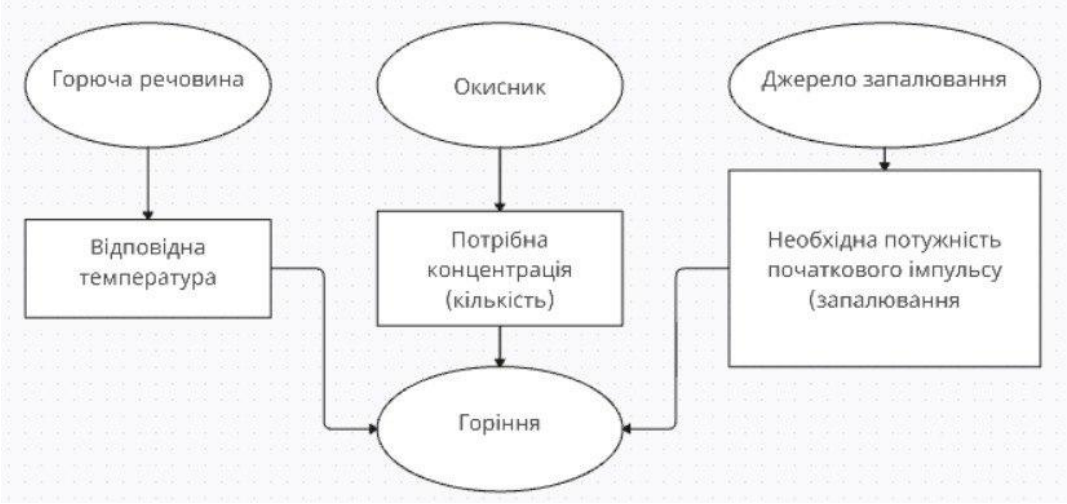

Рис. 4.3 – Схема основних причин виникнення пожежі

Особливу небезпеку становить використання несправних електричних кабелів або перевантажених подовжувачів. Підключення великої кількості пристроїв до одного джерела живлення може призводити до перегріву проводки та виникнення короткого замикання.

Для забезпечення пожежної безпеки необхідно регулярно перевіряти технічний стан електрообладнання та не допускати використання пошкоджених кабелів або несправних пристроїв. Усі електроприлади повинні використовуватися відповідно до вимог безпеки та технічної документації.

У приміщенні повинні бути передбачені засоби пожежогасіння, зокрема вогнегасники. Для приміщень із комп’ютерною технікою найчастіше використовуються порошкові або вуглекислотні вогнегасники, які дозволяють гасити електрообладнання без значного пошкодження техніки.

Також важливо забезпечити вільний доступ до шляхів евакуації та аварійних виходів. Працівники повинні бути ознайомлені з правилами поведінки у разі виникнення пожежі та знати місце розташування засобів пожежогасіння.

Рис. 4.4 – Приклад схеми евакуації.

У разі виникнення пожежі необхідно:

- негайно відключити електроживлення обладнання;
- повідомити відповідні служби;
- використати первинні засоби пожежогасіння;
- організовано залишити приміщення через евакуаційні виходи.

Для зменшення ризику виникнення пожежі рекомендується:

- не залишати увімкнене обладнання без нагляду;
- регулярно очищати техніку від пилу;
- не перевантажувати електромережу;
- використовувати справні мережеві фільтри та джерела живлення;
- дотримуватися правил експлуатації електроприладів.

Тип вогнегасника	**Призначення**
Порошковий	Гасіння електрообладнання
Вуглекислий	Гасіння комп’ютерної техніки
Водяний	Гасіння твердих матеріалів

Таким чином, дотримання правил пожежної безпеки є важливою умовою безпечної роботи в приміщеннях із комп’ютерною технікою. Регулярний контроль стану електрообладнання, правильна організація електромережі та наявність засобів пожежогасіння дозволяють суттєво знизити ризик виникнення пожежі під час роботи з програмними системами.

5 ОРГАНІЗАЦІЙНО-ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Організація процесу розробки програмного забезпечення

Організація процесу розробки програмного забезпечення є важливим етапом створення будь-якої програмної системи. Від правильного планування етапів роботи залежить ефективність реалізації проєкту, зручність підтримки системи та якість кінцевого результату.

У межах даної дипломної роботи розробляється програмна система аналізу гача-механік гри Genshin Impact із використанням Spring Boot, MongoDB та Docker. Основною метою системи є зберігання, обробка та аналіз статистичних даних, пов’язаних із механізмами pity, soft pity та 50/50.

Процес розробки програмного забезпечення можна поділити на декілька основних етапів:

- аналіз предметної області;
- проєктування системи;
- вибір технологій;
- реалізація серверної частини;
- організація бази даних;
- тестування системи;
- аналіз отриманих результатів.

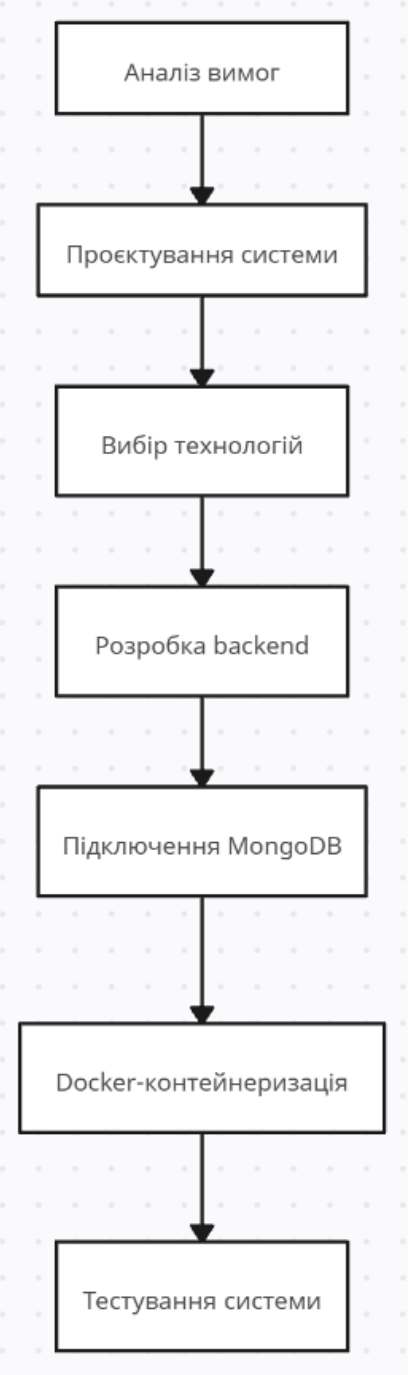

Рис. 5.1 – Схема етапів розробки програмного забезпечення.

На першому етапі було проведено аналіз предметної області та дослідження принципів роботи гача-систем у сучасних відеоіграх. Особливу увагу приділено механізмам pity, soft pity та 50/50 у грі Genshin Impact.

Після аналізу предметної області було виконано проєктування структури програмної системи. На даному етапі визначалися основні компоненти

застосунку, способи взаємодії між ними та структура збереження статистичних даних.

Одним із важливих етапів стала технологічна підготовка проєкту. Для реалізації серверної частини було обрано Spring Boot, для зберігання статистики – MongoDB, а для контейнеризації та запуску системи — Docker. Використання сучасних технологій дозволило спростити процес розробки та забезпечити гнучкість програмної системи.

Наступним етапом стала реалізація серверної частини застосунку. У межах цього етапу виконувалося створення REST API, реалізація серверної логіки та налаштування взаємодії з базою даних MongoDB.

Після створення основних компонентів системи було виконано налаштування контейнеризації за допомогою Docker. Це дозволило забезпечити стабільне середовище виконання та автоматизувати процес запуску програмної системи.

Важливим етапом процесу розробки стало тестування застосунку. Під час тестування перевірялася коректність роботи REST API, взаємодія серверної частини з базою даних та правильність обробки статистичних даних.

Окрему увагу приділено аналізу результатів роботи системи. Для цього використовувалися статистичні дані круток та результати моделювання гача-механік.

Етап	**Основний зміст робіт**
Аналіз предметної області	Дослідження гача-систем
Проєктування	Побудова структури системи
Розробка backend	Створення REST API
Робота з MongoDB	Збереження статистики
Docker	Контейнеризація системи
Тестування	Перевірка роботи застосунку

Таким чином, процес розробки програмного забезпечення складався з послідовних етапів аналізу, проєктування, реалізації та тестування системи.

Використання сучасних технологій дозволило створити гнучку програмну систему для аналізу гача-механік та статистики круток у грі Genshin Impact.

5.2 Оцінка необхідних ресурсів для реалізації проєкту

Для реалізації програмної системи аналізу гача-механік необхідне використання як апаратних, так і програмних ресурсів. Оскільки розробка проєкту пов’язана зі створенням серверної частини застосунку, роботою з базою даних та контейнеризацією системи, важливо забезпечити стабільне середовище для розробки та тестування програмного забезпечення.

Основним апаратним ресурсом для реалізації проєкту є персональний комп’ютер або ноутбук із достатнім рівнем продуктивності для запуску середовища розробки, серверної частини застосунку та Docker-контейнерів. Для роботи із Spring Boot, MongoDB та Docker необхідна наявність сучасного процесора, достатнього обсягу оперативної пам’яті та стабільного підключення до мережі Інтернет.

Рис. 5.2 – Схема основних апаратних ресурсів.

У межах реалізації проєкту використовуються такі програмні ресурси:

- операційна система Windows;
- Java Development Kit (JDK);
- Spring Boot;
- MongoDB;

- Docker;
- середовище розробки IntelliJ IDEA;
- веббраузер для тестування REST API.

Для створення серверної частини застосунку використовується Spring Boot, який забезпечує реалізацію REST API та обробку статистичних даних. MongoDB використовується для зберігання інформації про результати круток та статистику гача-механік. Docker застосовується для контейнеризації та автоматизації запуску системи.

Важливою перевагою даного проєкту є використання переважно безкоштовного програмного забезпечення. Більшість технологій, які застосовуються під час розробки, мають відкритий вихідний код або безкоштовні версії для навчальних та некомерційних проєктів.

Окрему роль у процесі реалізації проєкту відіграє доступ до мережі Інтернет. Інтернет-з’єднання використовується для отримання документації, завантаження бібліотек та залежностей, а також для тестування окремих компонентів системи.

Крім програмних та апаратних ресурсів, важливим фактором є часові ресурси, необхідні для аналізу предметної області, проєктування системи, реалізації серверної частини та проведення тестування.

Таким чином, для реалізації програмної системи аналізу гача-механік необхідно використання сучасного комп’ютерного обладнання, серверних технологій та програмних засобів розробки. Використання безкоштовних і відкритих технологій дозволяє зменшити витрати на реалізацію проєкту та забезпечує гнучкість програмної системи.

5.3 Розрахунок витрат на розробку програмної системи

Під час розробки програмної системи аналізу гача-механік необхідно враховувати витрати, пов’язані з використанням апаратних та програмних ресурсів. Оскільки проєкт має навчальний характер, значна частина програмного забезпечення використовується безкоштовно або у вигляді відкритих версій.

Основними витратами під час реалізації проєкту є використання персонального комп’ютера, електроенергії та мережі Інтернет. Додаткові витрати можуть бути пов’язані з використанням програмного забезпечення або сервісів для розробки та тестування системи.

У межах даного проєкту для створення серверної частини застосунку використовуються Spring Boot, MongoDB та Docker, які мають безкоштовні версії та можуть застосовуватися без додаткових фінансових витрат.

Для оцінки витрат було враховано орієнтовні витрати на електроенергію, використання мережі Інтернет та амортизацію комп’ютерного обладнання.

Найменування	**Орієнтовна вартість**
Електроенергія	900 грн
Інтернет	200 грн
Амортизація ПК	700 грн
IntelliJ IDEA Community	Безкоштовно
Spring Boot	Безкоштовно
MongoDB Community	Безкоштовно
Docker Desktop	Беззкоштовно

Загальні витрати на реалізацію програмної системи є відносно невисокими, оскільки більшість технологій, використаних у проєкті, поширюються за відкритими ліцензіями та не потребують придбання платного програмного забезпечення.

Використання безкоштовних інструментів розробки є важливою перевагою сучасних програмних технологій. Це дозволяє створювати програмні системи без значних фінансових витрат та забезпечує доступність розробки навіть для навчальних або індивідуальних проєктів.

Окремо слід зазначити, що використання Docker спрощує процес розгортання системи та дозволяє уникнути додаткових витрат на налаштування середовища виконання. MongoDB забезпечує безкоштовне зберігання статистичних даних у межах локального використання, а Spring Boot дозволяє створювати серверні застосунки без необхідності використання платних серверних платформ.

Таким чином, витрати на реалізацію програмної системи є мінімальними та переважно пов’язані з використанням апаратних ресурсів. Використання сучасних безкоштовних технологій дозволяє значно знизити вартість розробки програмного забезпечення та забезпечити ефективну реалізацію системи аналізу гача-механік.

5.4 Оцінка ефективності програмної системи

Ефективність програмної системи визначається її здатністю виконувати поставлені завдання, забезпечувати зручну обробку даних та стабільну роботу основних компонентів застосунку. У межах даного проєкту основною метою системи є аналіз гача-механік гри Genshin Impact, обробка статистичних даних та зберігання результатів круток.

Однією з головних переваг створеної системи є автоматизація процесу аналізу статистики. Використання серверної частини на базі Spring Boot дозволяє автоматично обробляти великі обсяги даних, виконувати обчислення pity, аналізувати результати механізму 50/50 та формувати статистичні результати без необхідності ручної обробки інформації.

Важливою характеристикою ефективності є швидкість роботи системи. Використання MongoDB дозволяє швидко зберігати та отримувати статистичні дані, а документно-орієнтована структура бази даних спрощує роботу з JSON-документами та результатами симуляцій.

Ще однією перевагою є масштабованість програмної системи. У разі збільшення кількості статистичних даних або додавання нових функцій система може бути розширена без суттєвого перепроєктування архітектури застосунку. Використання Docker також спрощує процес розгортання та підтримки програмного забезпечення.

Окрему роль відіграє зручність взаємодії між компонентами системи. REST API забезпечує обмін даними між frontend та backend частинами застосунку, а JSON-формат дозволяє спростити передачу статистичної інформації.

Використання сучасних технологій також позитивно впливає на ефективність системи. Spring Boot забезпечує швидку реалізацію серверної логіки, MongoDB – гнучке зберігання статистики, а Docker – стабільне середовище виконання застосунку.

У межах дослідження програмна система дозволяє:

- аналізувати статистику круток;
- обчислювати середній pity;
- визначати результати механізму 50/50;
- зберігати історію круток;
- виконувати статистичний аналіз отриманих результатів.

Крім функціональної ефективності, система має навчальну та дослідницьку цінність. Отримані результати можуть бути використані для подальшого дослідження гача-механік, аналізу систем випадкових винагород та створення симуляційних моделей у сфері відеоігор.

Таким чином, розроблена програмна система є ефективним інструментом для аналізу гача-механік та статистики круток у грі Genshin Impact. Використання сучасних серверних технологій забезпечує стабільну роботу системи, автоматизацію обробки даних та можливість подальшого розвитку програмного застосунку.

ВИСНОВОК

У межах виконання дипломної роботи було проведено дослідження гача-системи гри Genshin Impact, розглянуто основні механізми випадкових винагород та виконано аналіз статистики круток. Особливу увагу приділено механізмам pity, soft pity та 50/50, які є одними з основних елементів сучасних гача-систем.

Під час виконання роботи було проаналізовано принципи функціонування банерів у грі Genshin Impact, досліджено особливості івентових, постійних та зброярських банерів, а також офіційні шанси випадіння персонажів і зброї. У результаті проведеного аналізу було встановлено, що гача-система гри поєднує елементи випадковості та гарантованого отримання нагород, що дозволяє підтримувати баланс між випадковим випадінням та механізмами гарантії.

У роботі також було побудовано математичну модель отримання п’ятизіркових персонажів та виконано аналіз механізму 50/50. На основі статистичних даних було розглянуто вплив pity-системи на середню кількість круток до отримання персонажа, а також досліджено закономірності випадіння рідкісних нагород.

Окрему увагу приділено аналізу особистої статистики круток. У межах практичної частини було проведено дослідження результатів круток за різними типами банерів, проаналізовано кількість круток до отримання п’ятизіркових персонажів та виконано оцінку результатів механізму 50/50. Проведений аналіз дозволив порівняти теоретичні закономірності роботи гача-системи з реальними результатами.

Крім математичного та статистичного аналізу, у дипломній роботі було розглянуто програмну реалізацію системи аналізу гача-механік. Для створення серверної частини застосунку використано Spring Boot, для зберігання

статистичних даних – MongoDB, а для контейнеризації та автоматизації запуску системи – Docker.

Під час виконання роботи було досліджено особливості Spring Boot, принципи створення REST API та реалізацію серверної логіки застосунку. Також було розглянуто переваги використання MongoDB для роботи зі статистичними даними та особливості використання Docker для створення ізольованого середовища виконання програмної системи.

У розділі охорони праці було проаналізовано умови праці під час розробки програмного забезпечення, розглянуто шкідливі та небезпечні фактори під час роботи за комп'ютером, а також питання пожежної безпеки та організації робочого місця.

В організаційно-економічному розділі було розглянуто процес розробки програмної системи, оцінено необхідні ресурси для реалізації проєкту та виконано аналіз ефективності створеної системи.

Отримані результати можуть бути використані для подальшого дослідження гача-механік, створення систем статистичного аналізу та симуляції круток у сучасних відеоіграх. Крім цього, результати роботи можуть бути корисними для аналізу систем випадкових винагород та дослідження поведінки користувачів у F2P-проєктах.

Таким чином, у межах дипломної роботи було успішно виконано поставлені завдання, проведено дослідження гача-системи Genshin Impact та розглянуто принципи створення програмної системи для аналізу статистики круток із використанням сучасних серверних технологій.

СПИСОК ЛІТЕРАТУРИ

1 Hoyoverse [Електронний ресурс] // Genshin Impact Official Website – Режим доступу: https://genshin.hoyoverse.com/.

2 Hoyolab [Електронний ресурс] // Wish System Details – Режим доступу: https://www.hoyolab.com/.

3 Spring.io [Електронний ресурс] // Spring Boot Documentation – Режим доступу: https://spring.io/projects/spring-boot.

4 MongoDB Documentation [Електронний ресурс] // MongoDB Manual – Режим доступу: https://www.mongodb.com/docs/.

5 Docker Docs [Електронний ресурс] // Docker Documentation – Режим доступу: https://docs.docker.com/.

6 Oracle [Електронний ресурс] // Java Documentation – Режим доступу: https://docs.oracle.com/en/java/.

7 MDN Web Docs [Електронний ресурс] // JSON Structure and Syntax – Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.

8 Baeldung [Електронний ресурс] // REST API with Spring Boot – Режим доступу: https://www.baeldung.com/rest-with-spring-series.

9 GeeksforGeeks [Електронний ресурс] // Introduction to NoSQL Databases – Режим доступу: https://www.geeksforgeeks.org/introduction-to-nosql/.

10 Atlassian [Електронний ресурс] // What is Docker Containerization – Режим доступу: https://www.atlassian.com/microservices/cloud-computing/what-is-containerization.

11 Red Hat [Електронний ресурс] // What is REST API – Режим доступу: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

12 IBM [Електронний ресурс] // What is MongoDB – Режим доступу: https://www.ibm.com/topics/mongodb.

13 Oracle [Електронний ресурс] // Java Collections Framework – Режим доступу: https://docs.oracle.com/javase/tutorial/collections/.

14 W3Schools [Електронний ресурс] // JSON Tutorial – Режим доступу: https://www.w3schools.com/js/js_json_intro.asp.

15 DigitalOcean [Електронний ресурс] // Introduction to Docker and Containers – Режим доступу: https://www.digitalocean.com/community/tutorials/an-introduction-to-docker-containers.

ДОДАТОК А

Фрагмент реалізації сервісу аналітики AnalyticsService

```
package com.kolesnikova.genshinanalytics.analytics.service;

import com.kolesnikova.genshinanalytics.analytics.dto.FeaturedPityDistributionPoint;
import com.kolesnikova.genshinanalytics.analytics.dto.FeaturedPityDistributionResponse;
import com.kolesnikova.genshinanalytics.analytics.dto.HeroPieChartResponse;
import com.kolesnikova.genshinanalytics.analytics.dto.MetricResponse;
import com.kolesnikova.genshinanalytics.analytics.dto.PieChartItem;
import com.kolesnikova.genshinanalytics.analytics.dto.PityAnalyticsPoint;
import com.kolesnikova.genshinanalytics.analytics.dto.PityAnalyticsResponse;
import com.kolesnikova.genshinanalytics.analytics.dto.ProbabilityResponse;
import com.kolesnikova.genshinanalytics.wishhistory.entity.WishPullDocument;
import com.kolesnikova.genshinanalytics.wishhistory.repository.WishPullRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.function.Predicate;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class AnalyticsService {

private static final int HARD_PITY_LIMIT = 90;

private final WishPullRepository wishPullRepository;

public ProbabilityResponse anyFiveStarProbability(String playerId, String bannerName) {

    List<WishPullDocument> pulls = findFilteredPulls(playerId, bannerName, null);

    long hits = pulls.stream().filter(WishPullDocument::isFiveStar).count();

    return ProbabilityResponse.builder()

            .playerId(playerId)

            .bannerName(bannerName)

            .target("Any 5★")

            .pulls(pulls.size())
```

```
            .hits(hits)
            .probabilityPercent(round(percent(hits, pulls.size())))
            .build();
}
public ProbabilityResponse specificHeroProbability(String heroName, String playerId, String bannerName) {
    List<WishPullDocument> pulls = findFilteredPulls(playerId, bannerName, heroName);
    long hits = pulls.stream()
            .filter(WishPullDocument::isFiveStar)
            .filter(pull -> pull.getResult() != null)
            .filter(pull -> equalsIgnoreCase(heroName, pull.getResult().getItemName()))
            .count();
    return ProbabilityResponse.builder()
            .playerId(playerId)
            .bannerName(bannerName)
            .target(heroName)
            .pulls(pulls.size())
            .hits(hits)
            .probabilityPercent(round(percent(hits, pulls.size())))
            .build();
}
public HeroPieChartResponse fiveStarHeroPieChart(String playerId, String bannerName) {
    List<WishPullDocument> pulls = findFilteredPulls(playerId, bannerName, null);
    long totalPulls = pulls.size();
    Map<String, Long> hitsByHero = pulls.stream()
            .filter(WishPullDocument::isFiveStar)
            .filter(pull -> pull.getResult() != null)
            .filter(pull -> pull.getResult().getItemName() != null)
            .collect(Collectors.groupingBy(
                    pull -> pull.getResult().getItemName(),
                    TreeMap::new,
```

```
                Collectors.counting()
        ));
    List<PieChartItem> series = hitsByHero.entrySet().stream()
        .map(entry -> PieChartItem.builder()
                .name(entry.getKey())
                .value(round(percent(entry.getValue(), totalPulls)))
                .totalPulls(totalPulls)
                .hits(entry.getValue())
                .build())
        .toList();
    return HeroPieChartResponse.builder()
        .title("Ймовірність випадіння 5★ персонажів")
        .subtitle(createSubtitle(totalPulls, bannerName))
        .series(series)
        .build();
}
public PityAnalyticsResponse fiveStarByPity(String playerId, String bannerName) {
    List<WishPullDocument> pulls = findFilteredPulls(playerId, bannerName, null);
    Map<Integer, Long> totalAtPity = pulls.stream()
        .filter(pull -> pull.fiveStarPityOnPull() > 0)
        .collect(Collectors.groupingBy(
                WishPullDocument::fiveStarPityOnPull,
                TreeMap::new,
                Collectors.counting()
        ));
    Map<Integer, Long> dropsAtPity = pulls.stream()
        .filter(WishPullDocument::isFiveStar)
        .filter(pull -> pull.fiveStarPityOnPull() > 0)
        .collect(Collectors.groupingBy(
                WishPullDocument::fiveStarPityOnPull,
                TreeMap::new,
```

```
                    Collectors.counting()
            ));
    Set<Integer> pityValues = new TreeSet<>();
    pityValues.addAll(totalAtPity.keySet());
    pityValues.addAll(dropsAtPity.keySet());
    List<PityAnalyticsPoint> points = pityValues.stream()
            .map(pity -> PityAnalyticsPoint.builder()
                    .pity(pity)
                    .fiveStarDrops(dropsAtPity.getOrDefault(pity, 0L))
                    .totalPullsAtPity(totalAtPity.getOrDefault(pity, 0L))
                    .empiricalProbabilityPercent(round(percent(
                            dropsAtPity.getOrDefault(pity, 0L),
                            totalAtPity.getOrDefault(pity, 0L)
                    )))
                    .build())
            .toList();
    return PityAnalyticsResponse.builder()
            .title("Емпірична ймовірність 5★ за pity")
            .points(points)
            .build();
}
public MetricResponse fiveStarExpectedPulls(String playerId, String bannerName) {
    ProbabilityResponse probability = anyFiveStarProbability(playerId, bannerName);
    double p = probability.getHits() == 0 ? 0.0 : (double) probability.getHits() /
probability.getPulls();
    double expectedValue = p == 0.0 ? 0.0 : 1.0 / p;
    return MetricResponse.builder()
            .name("E[X] для першого будь-якого 5★")
            .value(round(expectedValue))
            .unit("pulls")
            .sampleSize(probability.getPulls())
```

```
            .probabilityPercent(probability.getProbabilityPercent())
            .formula("E[X] = 1 / p")
            .build();
    }
    public MetricResponse fiveStarVariance(String playerId, String bannerName) {
        ProbabilityResponse probability = anyFiveStarProbability(playerId, bannerName);
        double p = probability.getHits() == 0 ? 0.0 : (double) probability.getHits() /
probability.getPulls();
        double variance = p == 0.0 ? 0.0 : (1.0 - p) / (p * p);
        return MetricResponse.builder()
            .name("Var(X) для першого будь-якого 5★")
            .value(round(variance))
            .unit("pulls²")
            .sampleSize(probability.getPulls())
            .probabilityPercent(probability.getProbabilityPercent())
            .formula("Var(X) = (1 - p) / p²")
            .build();
    }
    public MetricResponse featuredFiveLimitExpectedPulls(String playerId, String bannerName) {
        List<Integer> intervals = featuredFiveIntervals(playerId, bannerName);
        double probabilityPercent = featuredProbability(playerId, bannerName) * 100.0;
        double expectedValue = intervals.stream()
            .mapToInt(Integer::intValue)
            .average()
            .orElse(0.0);
        return MetricResponse.builder()
            .name("E[X] для Featured 5★ Limit")
            .value(round(expectedValue))
            .unit("pulls")
            .sampleSize(intervals.size())
            .probabilityPercent(round(probabilityPercent))
```

```
            .formula("середня кількість pulls між Featured 5★ drops")
            .build();
    }
    public MetricResponse featuredFiveLimitVariance(String playerId, String bannerName) {
        List<Integer> intervals = featuredFiveIntervals(playerId, bannerName);
        double probabilityPercent = featuredProbability(playerId, bannerName) * 100.0;
        double mean = intervals.stream().mapToInt(Integer::intValue).average().orElse(0.0);
        double variance = intervals.isEmpty()
                ? 0.0
                : intervals.stream().mapToDouble(interval -> Math.pow(interval - mean, 2)).sum() /
    intervals.size();
        return MetricResponse.builder()
                .name("Var(X) для Featured 5★ Limit")
                .value(round(variance))
                .unit("pulls²")
                .sampleSize(intervals.size())
                .probabilityPercent(round(probabilityPercent))
                .formula("емпірична дисперсія кількості pulls між Featured 5★ drops")
                .build();
    }
    public FeaturedPityDistributionResponse featuredFivePityDistribution(String playerId, String
    bannerName) {
        Map<Integer, Long> dropsByPity = findFilteredPulls(playerId, bannerName, null).stream()
                .filter(WishPullDocument::isFeaturedFiveStar)
                .filter(pull -> pull.fiveStarPityOnPull() > 0)
                .collect(Collectors.groupingBy(
                        WishPullDocument::fiveStarPityOnPull,
                        TreeMap::new,
                        Collectors.counting()
                ));
        long totalDrops = dropsByPity.values().stream().mapToLong(Long::longValue).sum();
```

```
    long cumulativeDrops = 0;
    List<FeaturedPityDistributionPoint> points = new ArrayList<>();
    for (int pity = 1; pity <= HARD_PITY_LIMIT; pity++) {
        long dropsAtPity = dropsByPity.getOrDefault(pity, 0L);
        cumulativeDrops += dropsAtPity;
        points.add(FeaturedPityDistributionPoint.builder()
                .pity(pity)
                .dropsAtPity(dropsAtPity)
                .cumulativeDrops(cumulativeDrops)
                .probabilityPercent(round(percent(dropsAtPity, totalDrops)))
                .cumulativeProbabilityPercent(round(percent(cumulativeDrops, totalDrops)))
                .build());
    }
    return FeaturedPityDistributionResponse.builder()
            .title("CDF випадіння Featured 5★ за pity")
            .totalFeaturedDrops(totalDrops)
            .points(points)
            .build();
}
private List<WishPullDocument> findFilteredPulls(String playerId, String bannerName, String
featuredHeroName) {
    Predicate<WishPullDocument> filter = pull -> true;
    if (hasText(playerId)) {
        filter = filter.and(pull -> playerId.equals(pull.getPlayerId()));
    }
    if (hasText(bannerName)) {
        filter = filter.and(pull -> pull.getBanner() != null)
                .and(pull -> equalsIgnoreCase(bannerName, pull.getBanner().getBannerName()));
    }
    if (!hasText(bannerName) && hasText(featuredHeroName)) {
        filter = filter.and(this::hasBanner)
```

```
                .and(pull -> pull.getBanner().getFeatured5Star() != null)
                .and(pull -> pull.getBanner().getFeatured5Star().stream()
                        .anyMatch(hero -> equalsIgnoreCase(featuredHeroName, hero)));
    }
    return wishPullRepository.findAll().stream()
            .filter(filter)
            .sorted(Comparator.comparing(
                    WishPullDocument::getTimestamp,
                    Comparator.nullsLast(Comparator.naturalOrder())
            ))
            .toList();
}
private List<Integer> featuredFiveIntervals(String playerId, String bannerName) {
    List<Integer> intervals = new ArrayList<>();
    int pullsSinceLastFeatured = 0;
    for (WishPullDocument pull : findFilteredPulls(playerId, bannerName, null)) {
        pullsSinceLastFeatured++;
        if (pull.isFeaturedFiveStar()) {
            intervals.add(pullsSinceLastFeatured);
            pullsSinceLastFeatured = 0;
        }
    }
    return intervals;
}
private double featuredProbability(String playerId, String bannerName) {
    List<WishPullDocument> pulls = findFilteredPulls(playerId, bannerName, null);
    long featuredDrops = pulls.stream().filter(WishPullDocument::isFeaturedFiveStar).count();
    return pulls.isEmpty() ? 0.0 : (double) featuredDrops / pulls.size();
}
private String createSubtitle(long totalPulls, String bannerName) {
    if (hasText(bannerName)) {
```

```
        return totalPulls + " pulls · " + bannerName;
    }
    return totalPulls + " pulls";
}
private boolean hasBanner(WishPullDocument pull) {
    return pull.getBanner() != null;
}
private boolean hasText(String value) {
    return value != null && !value.isBlank();
}
private boolean equalsIgnoreCase(String left, String right) {
    return left != null && right != null && left.equalsIgnoreCase(right);
}
private double percent(long numerator, long denominator) {
    return denominator == 0 ? 0.0 : (double) numerator * 100.0 / denominator;
}
private double round(double value) {
    return Math.round(value * 10_000.0) / 10_000.0;
}
```

ДОДАТОК В

Приклад структури JSON-документа для зберігання статистики круток


```
{
    "game": "Genshin Impact",
    "playerId": "player_005",
    "accountRegion": "EU",
    "sessionId": "sess_001",
    "eventType": "WISH_PULL",
    "timestamp": "2024-01-10T18:00:07Z",
    "banner": {
      "bannerType": "character_event",
      "bannerName": "The Heron's Court",
      "version": "4.3",
      "featured5Star": [
        "Kamisato Ayaka"
      ],
      "featured4Star": [
        "Rosaria",
        "Sayu",
        "Candace"
      ]
    },
    "pullNumberInSession": 1,
    "pullNumberOnBanner": 1,
    "pullType": "single",
    "result": {
      "itemType": "weapon",
      "itemName": "Thrilling Tales of Dragon Slayers",
      "rarity": 3,
```

```
    "featured": false,
    "guaranteed": false,
    "won5050": null
  },
  "pity": {
    "pity4Before": 0,
    "pity5Before": 0,
    "pity4After": 1,
    "pity5After": 1,
    "guaranteedFeatured5StarBefore": false,
    "guaranteedFeatured5StarAfter": false
  },
  "economy": {
    "currencyType": "Intertwined Fate",
    "currencySpent": 1,
    "primogemsBalanceAfter": 52000,
    "intertwinedFatesAfter": 84,
    "masterlessStarglitterAfter": 0,
    "masterlessStardustAfter": 15
  },
  "metadata": {
    "source": "synthetic_generation",
    "verified": false,
    "importedAt": "2026-05-23T10:00:00Z",
    "tags": [
      "gacha",
      "3-star"
    ]
  }
}
```