ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Методи та засоби управління ігровими подіями через інтеграцію з донат-сервісами»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Данило ПОСЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-62
Данило ПОСЕНКО

Керівник: Владислав ЯСКЕВИЧ
д.т.н.,

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Посенку Данилу Романовичу

1. Тема кваліфікаційної роботи: «Розробка методів та засобів управління ігровими подіями через інтеграцію з донат-сервісами»

керівник кваліфікаційної роботи Владислав ЯСКЕВИЧ к.т.н., доцент, доцент кафедри ІПЗ

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «15» жовтня 2024 р. № 320.

2. Строк подання кваліфікаційної роботи «17» грудня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, параметри сканування місцевості, методи обробки та візуалізації даних ультразвукового сенсора, вимоги до точності та представлення даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження методів інтеграції донат сервісів.
2. Моделювання та реалізація запропонованого методу.
3. Аналіз ефективності та точності запропонованого методу

5. Перелік ілюстративного матеріалу: *презентація*

1.Мета, об’єкт, предмет
2.Порівняння методу
3. Діаграма пакетів
4. Блок схема алгоритму методу
5. Результат роботи методу
6. Результат роботи методу

6. Дата видачі завдання «16» жовтня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	20.10-25.10.2024	
2	Вивчення матеріалів для аналізу розвитку технологій інтерактивної взаємодії середовища з глядачем	26.10-01.11.2024	
3	Дослідження методів інтеграції донат-систем в комп’ютерні ігри	02.11-04.11.2024	
4	Дослідження способів інтеграції гри з програмними інтерфейсами	05.11-07.11.2024	
5	Аналіз особливостей аспектів гри для успішної інтеграції методу	08.11-12.11.2024	
6	Налаштування методу та тестування його ефективності	13.11-17.11.2024	
7	Оформлення роботи: вступ, висновки, реферат	18.11-22.11.2024	
8	Розробка демонстраційних матеріалів	23.11-01.12.2024	
9	Попередній захист роботи	27.11-12.12.2024	

Здобувач вищої освіти _____ Данило ПОСЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 92 стор., 7 табл., 25 рис., 30 джерел.

Мета роботи – реалізація процесу дистанційного управління ігровими подіями та підвищення залученості глядачів за рахунок використання методів інтеграції з донат-сервісами, що дозволяють глядачу прямих трансляцій взаємодіяти з контентом гри.

Об'єкт дослідження – процес дистанційного управління ігровими подіями в онлайн-іграх, що залучає глядача.

Предмет дослідження – засоби та технології інтеграції донат-сервісів з ігровими платформами для ефективного управління подіями та взаємодії з користувачами.

В роботі використані різні та актуальні на момент написання роботи програмні інтерфейси, методи об'єктно-орієнтовного програмування.

Акцентовано увагу на правильній взаємодії між інтерфейсом сервісів та програмним кодом для досягнення інтерактивності.

Проведено аналіз наразі існуючих сервісів для отримання донатів. Вивчено принцип відправки донатів, перелік даних, що можуть повертати сервіси для стрімінгу та донатів.

Розроблено методи для виклику ігрових подій через інтерактивні донати, які перетворюються в ігрові події під час прямих трансляцій. Застосовано принципи об'єктно-орієнтовного програмуваннння для розробки програмного забезпеченння.

Проведено тестування для отримання оцінки ефективності використаних методів. Оцінено продуктивність та безперебійність роботи системи інтерактивних події.

Дане дослідження підтверджує перспективність та доцільність користування інтерактивними донатами під час прямих трансляцій. Розроблено методи які можуть використовуватись для розвитку українського стрімінгу та прогрессивнішого методу збору коштів.

КЛЮЧОВІ СЛОВА: ДОНАТ, СТРИМЕР, СТРИМІНГ, ІНТЕРАКТИВ, API.

ABSTRACT

The text part of the qualification work for obtaining the master's degree: 92 pages, 7 table, 25 figures, 30 sources.

The purpose of the work is implementing the process of remote control of gaming events and increasing viewer engagement through the use of integration methods with donation services that allow viewers of live broadcasts to interact with the game content.

The object of research is the process of remote control of game events in online games that involves the viewer.

The subject of research is the means and technologies for integrating donation services with gaming platforms for effective event management and user interaction.

The work uses various and relevant at the time of writing the work programming interfaces, object-oriented programming methods.

The focus is on the correct interaction between the service interface and the program code to achieve interactivity.

The analysis of currently existing services for receiving donations was conducted. The principle of sending donations, the list of data that streaming and donation services can return were studied.

Methods was developed to trigger game events through interactive donations, which are transformed into game events during live broadcasts.

The principles of object-oriented programming were applied to develop the software.

Testing was conducted to assess the effectiveness of the methods used. The performance and continuity of the interactive event system were assessed.

This study confirms the potential and feasibility of using interactive donations during live broadcasts. Software has been developed that can be used to develop Ukrainian streaming and a more progressive method of fundraising.

KEY WORDS: DONATION, STREAMER, STREAMING, INTERACTIVE, API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

OXO - Хрестики нулики (Noughts and Crosses).

NS - Nintendo Switch.

AR - Доповнена реальність (Augmented reality).

VR - Віртуальна реальність (Virtual reality).

GSON - Серіалізація JSON від Google (Google Gson).

JSON -Запис об'єктів JavaScript (JavaScript Object Notation).

CFG - Файл конфігурації (Configuration file).

MC - Майнкрафт (Minecraft).

API - Прикладний програмний інтерфейс (Application Programming Interface).

QTE - Швидкі події (Quick Time Events).

MCSM - Minecraft Story Mode.

OBS - Open Broadcaster Software.

VOD - Відео по запиту (Video On Demand).

RPC - Виклик відаленних процедур (Remote Procedure Call)

gRPC - Виклик відаленних процедур від Google (Google Remote Procedure Call)

JAR - Архів Java(Java ARchive).

UUID - Унікальний ідентифікатор пристрою (Universally Unique Identifier).

IDE - Інтегрована середа розробки (Integrated development environment).

HTTP - Протокол передачі гіпертексту (Hypertext Transfer Protocol).

ВСТУП

Завдяки стрімкому розвитку цифрових технологій, інтерактивність стає невід'ємною складовою сучасного стрімінгу. Інтеграція інтерактивних елементів відкриває нові горизонти для взаємодії між стрімерами та глядачами, перетворюючи звичайний процес перегляду на захопливий досвід. Це дозволяє не лише залучити більшу аудиторію, а й створити унікальні сценарії, що роблять контент справді незабутнім.

Основною ідеєю цієї розробки є створення методу, який дозволяє глядачам впливати на ігровий процес у реальному часі. Завдяки цьому методу, дії глядачів, такі як донати або команди в чаті, автоматично перетворюються на події у грі. Така система стимулює інтерактивність, роблячи глядачів не просто пасивними спостерігачами, а активними учасниками. Події можуть варіюватися від генерації ресурсів до виклику складних випробувань, залежно від фінансової підтримки чи активності аудиторії.

Ця робота спрямована на дослідження сучасних підходів до інтерактивного стрімінгу та розробку методу для Minecraft, що базується на інноваційних алгоритмах. Розробка включає аналіз існуючих рішень, створення інструментів для керування ігровими подіями та інтеграцію глибокої взаємодії між глядачами й стрімером.

Загалом, впровадження інтерактивних рішень у стрімінгові платформи є важливим кроком до створення нового рівня контенту, що поєднує технологічну інноваційність із креативністю. Такі проекти не лише розвивають індустрію розваг, а й сприяють формуванню позитивного іміджу українського цифрового продукту у світі.

Мета роботи: Реалізація процесу дистанційного управління ігровими подіями та підвищення залученості глядачів за рахунок використання методів інтеграції з донат-сервісами, що дозволяють глядачу прямих трансляцій взаємодіяти з контентом гри.

Об'єкт дослідження: Процес дистанційного управління ігровими подіями в онлайн-іграх, що залучає глядача.

Предмет дослідження: Засоби та технології інтеграції донат-сервісів з ігровими платформами для ефективного управління подіями та взаємодії з користувачами.

Для досягнення мети вирішувалися наступні завдання:

1. Практичний огляд. Проведено аналіз і оцінку існуючих методів щоб отримати повне уявлення про різноманітні методи та способи, щоб отримати повне представлення про інтерактивні донати.
2. Дослідження. Застосовано теорії та методики передачі та отримання даних. Проведені порівняння та оцінки ефективності методів.
3. Програмування. Розроблено методи інтегрованих донатів, використовуючи ресурси гри та мережевих технологій.
4. Оцінка. Була проведена оцінка перспективності методів для українського стрімінгу та які позитивні результати показують методи.
5. Вибір необхідних інструментів. Було ретельно підібрано технології та методи, які відповідають сучасним потребам інтерактивного стрімінгу. Для створення інтерактивних донатів були обрані надійні мережеві інструменти, платформи для розробки методів та ефективні алгоритми для синхронізації даних у реальному часі. Також були враховані вимоги до простоти налаштування і використання методу для стрімерів.
6. Перспективний розвиток. На основі проведених досліджень запропоновано напрямки подальшого вдосконалення інтерактивних рішень для стрімінгу. Розглядаються можливості інтеграції додаткових функцій, таких як аналітика глядацької активності, адаптивні сценарії подій та підвищення безпеки обробки даних.

Вирішення проблеми та аналіз потенційних рішень роблять крок в розвитку інтерактивності між глядачами та стрімером, а також збільшує імерсивність глядів в події.

1. АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ТА РІШЕНЬ

1.1 Історія та розвиток інтерактивних та імерсивних елементів в іграх та стрімінгу

Інтерактивність є ключовим концептом у багатьох галузях, включаючи теорію інформації, інформатику, програмування, телекомунікації, соціологію, педагогіку, проектування взаємодії, музейну справу, а також промисловий дизайн. Незважаючи на відсутність єдиного усталеного визначення серед фахівців цих сфер, загальне розуміння інтерактивності можна сформулювати наступним чином:

Інтерактивність — це метод організації системи, при якому досягнення мети здійснюється через обмін інформацією між її складовими. Складовими інтерактивності є всі елементи співпрацюючої системи, які забезпечують взаємодію з іншою системою або з людиною (користувачем). Ця взаємодія відбувається через обмін даними, командами або інформацією, що дозволяє здійснювати зворотний зв'язок та взаємодіяти з системою в реальному часі.

Інтерактивність у комп'ютерних іграх є ключовим елементом, що забезпечує взаємодію гравця з ігровим середовищем, створюючи унікальний і захоплюючий досвід. Витоки інтерактивності можна простежити від найперших комп'ютерних експериментів до сучасних інноваційних технологій. Інтерактивність у комп'ютерних іграх - це ключовий аспект, який забезпечує гравцям можливість активно взаємодіяти з ігровим світом, відчувати контроль над подіями та вирішувати завдання залежно від власних виборів. Вона дозволяє гравцям не лише спостерігати за подіями, але й впливати на них, що робить ігровий досвід більш особистим та захоплюючим.

Розвиток інтерактивності в комп'ютерних іграх відбувався разом з технологічними та культурними змінами. Починаючи з простих аркадних ігор з мінімальним рівнем взаємодії, був перехід до складних іммерсивних ігор, де гравці можуть взаємодіяти з різними персонажами, об'єктами та оточуючим світом.

Сучасні інновації, такі як розширена реальність, віртуальна реальність та штучний інтелект, відкривають нові можливості для ще більшої інтерактивності в іграх. Гравці можуть взаємодіяти з віртуальним світом за допомогою жестів, голосових команд або навіть просто зі своїм власним тілом. Розвиток інтерактивності відбувався разом із розвитком самих ігор так як гравцям завжди хотілось отримувати більш імерсивний досвіді, щоб з більшою силою можна було зануритись у ігровий процес.

1.1.1 Розвиток та еволюція імерсивності в іграх

"OXO", або "Noughts and Crosses", була створена в 1952 році Олексом Дугласом у рамках його докторської дисертації з взаємодії людини з комп'ютером у Кембриджському університеті. Гра працювала на комп'ютері EDSAC (Electronic Delay Storage Automatic Calculator) і демонструвала новаторський підхід до інтерактивності.

Користувач вводить свої ходи через спеціальний пристрій введення, а комп'ютер відображав гру на екрані, використовуючи електронно-променеву трубку. Це створювало відчуття взаємодії з машиною в режимі реального часу, що було вражаючим для тогочасних технологій. Хоча "OXO" не мала комерційного успіху і не була широко доступною, вона стала важливим етапом у розвитку ігрової індустрії.

Гра підкреслювала ключові принципи, які залишаються актуальними до сьогодні: необхідність зручного інтерфейсу для взаємодії з користувачем і потенціал комп'ютерів для створення інтерактивних розваг. Як одна з перших інтерактивних комп'ютерних ігор, "OXO" заклала основу для майбутнього розвитку відеоігор.

Рис. 1.1 Гра «ОХО»

"Теніс для двох" було створено в 1958 році фізиком Вільямом Гіґінботемом, який прагнув продемонструвати, як сучасні технології можуть використовуватись не лише в наукових цілях, але й для розваг. Ця гра стала однією з перших, що відтворювала спортивне змагання, забезпечуючи двом гравцям можливість змагатися в реальному часі.

У грі використовувався осцилограф, який слугував екраном для відображення спрощеного тенісного корту. Управління здійснювалося за допомогою ручок, що дозволяли гравцям змінювати кут і силу удару, впливаючи на траєкторію м'яча. Ця механіка забезпечувала інтерактивність і додавала змагальний елемент, який привертав увагу до нових можливостей електронних пристроїв.

"Теніс для двох" став важливим етапом у розвитку комп'ютерних ігор. Хоча створювався як науковий експеримент, він продемонстрував потенціал інтерактивних систем і заклав основу для подальших спортивних симуляторів у відеоігровій індустрії.

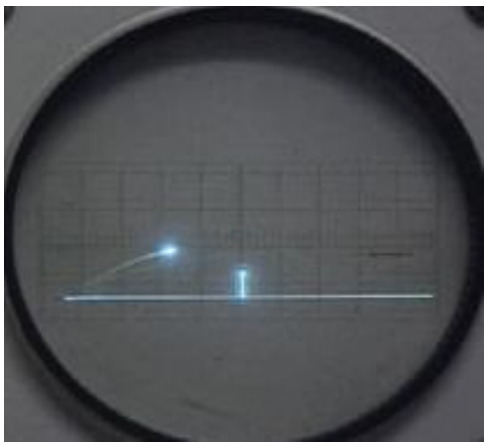


Рис. 1.2 Гра «Tennis for two»

«Spacewar!» була розроблено на початку 1960-х років групою студентів Массачусетського технологічного інституту для одного з перших інтерактивних комп’ютерів — PDP-1. Її створення стало не лише демонстрацією технічних можливостей тогочасних обчислювальних машин, але й важливим кроком у становленні концепції відеоігор як інтерактивного медіа. Ця гра пропонувала двом гравцям керувати космічними кораблями, які могли стріляти один в одного та маневрувати у відкритому космосі. Основною перешкодою для них було сильне гравітаційне поле центральної зірки, яке могло затягнути корабель, якщо той не рухався правильно. Усі ці елементи були засновані на реалістичних фізичних моделях, що враховували гравітацію, інерцію та траєкторії польотів, додаючи до гри стратегічну складність. Вона також виділялася своєю багатокористувацькою взаємодією, адже двоє гравців могли одночасно змагатися, що було новаторським рішенням для тієї епохи. Вона також стала однією з перших ігор, які втілювали елементи наукової фантастики, що згодом стало популярною темою в ігровій індустрії.

"Spacewar!" стала першим прикладом співпраці ентузіастів і розробників, що дало поштовх культурі спільної розробки програмного забезпечення.

«Space Invaders», випущена компанією Taito у 1978 році, стала одним із найбільш впливових проєктів в історії відеоігор, встановивши нові стандарти як для геймплею, так і для залучення гравців. У грі гравець керував гарматою, розташованою в нижній частині екрану, і мав знищувати хвилі прибульців, які

поступово наближалися до землі. Завдання ускладнювалося тим, що прибульці рухалися дедалі швидше, а їхні атаки ставали агресивнішими, змушуючи гравця миттєво реагувати на зміну ситуації.

Однією з найважливіших інновацій гри було інтегрування прогресивного збільшення складності. Кожен рівень додавав новий виклик: збільшення швидкості руху ворогів, скорочення часу для ухилення від атак і підвищення інтенсивності дій. Крім того, музичний супровід змінювався відповідно до темпу гри, підсилюючи відчуття напруги та небезпеки. Це була одна з перших ігор, яка використовувала звук як засіб емоційного впливу на гравця.

Дизайн гри також був інноваційним в ті часи і мав певну мету, а саме посприяти розвитку стратегічного мислення гравця. Гравець мав обирати між атаками ворогів, які могли швидко досягти його позиції, і захистом своїх бар'єрів, які поступово руйнувалися під вогнем. Такий підхід створював багатошаровий ігровий досвід, який тримав гравця у напрузі до самого кінця.

«Space Invaders» також справила значний вплив на культуру того часу. Її популярність стала настільки великою, що в Японії спостерігався дефіцит монет, необхідних для гри в аркадних автоматах. Вона започаткувала еру масової популярності відеоігор, заклавши основу для створення жанру шутерів і вплинувши на численні ігрові проєкти наступних десятиліть.

Pac-Man, розроблена компанією Namco у 1980 році, стала знаковою грою, яка змінила уявлення про аркадні розваги та відеоігри загалом. В основі геймплею лежала проста, але інноваційна концепція: гравець керував персонажем Pac-Man, який мав з'їсти всі точки на ігровому полі, уникаючи привидів. Кожен із чотирьох привидів мав власний унікальний алгоритм поведінки, що додавало грі стратегічної складності. Гра вимагала від гравця постійного аналізу ситуації, швидкого прийняття рішень і вміння передбачати рух ворогів. З'їдання спеціальних «великих точок» дозволяло Pac-Man тимчасово перехоплювати ініціативу, роблячи привидів вразливими. Це створювало додаткові тактичні можливості, які урізноманітнювали ігровий процес. На відміну від багатьох аркадних ігор того часу, Pac-Man запропонувала яскравого та харизматичного персонажа, який став символом цілої

епохи. Її геймплей був зрозумілим для широкої аудиторії, що зробило гру надзвичайно популярною як серед досвідчених гравців, так і серед новачків. Інтуїтивний дизайн рівнів і поступове зростання складності забезпечили високий рівень залучення.

Популярність Pac-Man поширилася далеко за межі аркадних залів, перетворивши гру на глобальний феномен. Її персонаж став героєм мультфільмів, товарів та рекламних кампаній, а сама гра вплинула на дизайн ігор у різних жанрах. Pac-Man залишила вагомий слід в історії, показавши, як відеоігри можуть поєднувати інноваційний геймплей, культурний вплив і комерційний успіх.

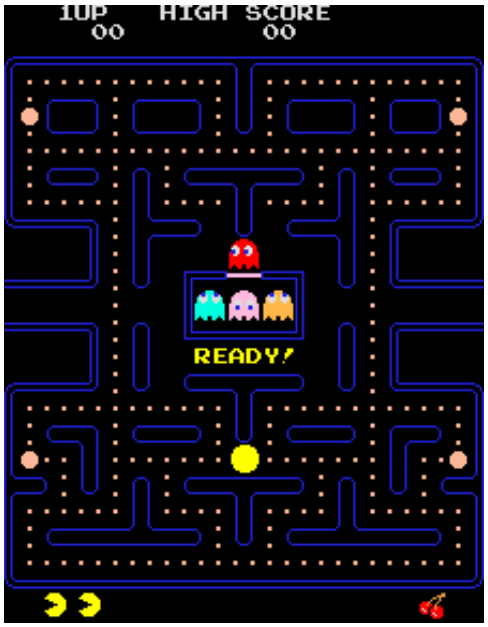


Рис. 1.3 Гра «Pac-man»

King’s Quest, розроблена компанією Sierra On-Line в 1984 році, стала однією з перших графічних пригодницьких ігор, яка змінила уявлення про інтерактивні розваги. Гравець керує персонажем, який подорожує через різноманітні сцени, взаємодіючи з об'єктами та вирішуючи головоломки що сприяє розвитку та просуванню сюжету. Відмінною рисою гри було поєднання текстових команд із графічними зображеннями, що надавало грі більш глибокого і захоплюючого характеру порівняно з подібними іграми цього жанру. Інтерактивність гри полягала в текстових командах які гравець міг сам вводити для взаємодії з навколишнім

світом, до прикладу для розмови з персонажами або виконання певних дій. Цей елемент дозволяв створювати більш відкриті та нелінійні варіанти проходження, що забезпечувало великий потенціал для експериментів. Одночасно графіка, хоч і була обмежена технологіями того часу, надавала візуальну підтримку для цих команд, що дозволяло зануритись у фантастичний світ гри.

King’s Quest не тільки визначила новий стандарт для графічних пригодницьких ігор, але й стала основою для створення цілої серії, яка продовжувала розвиватися на протязі багатьох років. Гра стала популярною завдяки своєму інноваційному підходу до інтерактивності і здатності поєднувати захоплюючий сюжет з технологією, що дозволяла гравцям вільно взаємодіяти з ігровим світом, відкриваючи нові можливості для розваг і для нелінійного проходження ігр.

Рис 1.4 Гра «King`s Quest»

Doom, створений компанією id Software у 1993 році, став одним із найважливіших етапів у розвитку відеоігор, особливо в жанрі шутерів від першої особи. Дана гра продемонструвала можливість переміщуватись тривимірними рівнями, знищуючи монстрів і шукаючи різні предмети, такі як ключі або зброю, для прогресії в грі. У той час, коли більшість ігор мали 2D-графіку або обмежену тривимірну перспективу, Doom зміг представити повноцінну 3D-графіку, яка

дозволяла більш імерсивно сприймати навколишнє середовище та бойові сцени.

Інтерактивність у Doom досягалася через кілька важливих аспектів. Перше, гра мала швидкий темп, що вимагало від гравця миттєвих рішень під час боїв і маневрів на різних рівнях. Гравці могли взаємодіяти з різними об'єктами — від збирання зброї до відкриття дверей і секретних проходів, що створювало динамічну та захоплюючу атмосферу, також важливим елементом була складна система ворогів із різними типами монстрів, кожен із яких мав свої патерни атаки, що ще більше стимулювало гравців вивчати ворогів і знаходити їм протидію.

Однією з найбільших інновацій Doom була її багатокористувацька складова. Гра вперше запропонувала повноцінний мережевий режим для двох або більше гравців, що дозволяло їм змагатися в реальному часі через локальну мережу. Це був важливий крок до розвитку онлайн-ігор, які згодом стали однією з основних частин індустрії відеоігор. Можливість грати в мережі дозволила створити новий соціальний аспект відеоігор, де гравці могли змагатися один з одним у зручний для себе час.

Doom став не лише знаковою грою для свого часу, а й заклав основи для подальшого розвитку ігрових технологій, таких як 3D-графіка, штучний інтелект ворогів і онлайн-гра. Її вплив на подальший розвиток жанру шутерів важко переоцінити. Ігрові механіки, що були введені в Doom такі як: інтуїтивно зрозумілий геймплей, швидка динаміка і підтримка мережевих боїв, стали стандартами, які згодом стали стандартом і використовувались майже у всіх іграх цього жанру.

Рис. 1.5 Гра «Doom»

Legend of Zelda, розроблена Nintendo в 1986 році, стала справжньою революцією в жанрі пригодницьких і рольових ігор, заклавши основи для багатьох сучасних відеоігор з відкритим світом. Гравець потрапляє в роль Лінка, молодого героя, який подорожує величезним, відкритим світом, досліджуючи різні локації, та борючись з різними ворогами, та також розгадує безліч цікавих головоломок, які приводять до отримання корисних предметів, або ж продвигають головного героя далі по сюжету. Головною метою було врятувати принцесу Зельду і зупинити злісного Ганона, але шлях до цього був наповнений численними завданнями, секретами та випробуваннями. Legend of Zelda є особливою, так як це відкритий світ, який дозволяв гравцям досліджувати його на власний розсуд, без чітко визначених лінійних обмежень. Гравець може обирати власний шлях, виконувати місії в будь-якому порядку і знаходити предмети, які відкривали нові можливості та локації. Така свобода стала новаторською для свого часу і дозволяла зануритись у світ, де кожна дія і вибір мали своє значення. Інтерактивність в Legend of Zelda також тісно пов’язана з розв’язанням головоломок і участю в битвах. Гравці повинні взаємодіяти з об'єктами в середовищі, що вимагало логічного мислення і кмітливості. Крім того, система боїв була насичена, з різними ворогами і складними боссами, що змушувало гравця адаптувати свою стратегію в залежності від ситуації. Використання спеціальних предметів, таких як меч, лук, бомби та щити, додавало ще більшу глибину в бої та взаємодію з навколишнім світом.

«The legend of Zelda» залишила помітний слід у культурі і продовжує впливати на ігрову індустрію до сьогодні, так як багато розробників використовують тей самий підхід для створення великих ігрових світів з елементами свободи, відкритості та глибокої інтерактивності.

1.2 Вплив платформ для стрімінгу та розвиток інтерактивних елементів

Платформи для стрімінгу, такі як Twitch, YouTube Gaming, Kick , значно вплинули на розвиток інтерактивних елементів у відеоіграх і медіа. Ці платформи перетворили пасивне споживання контенту на активну взаємодію між стрімерами та їхньою аудиторією, що привело до численних інновацій в інтерактивних технологіях.

Платформи дозволяють глядачам безпосередньо взаємодіяти зі стрімерами в режимі реального часу через чати, емодзі та донати. Це створює двосторонню комунікацію, де глядачі можуть впливати на події, що відбуваються на екрані.

Стрімери часто залучають аудиторію до вибору ігрових дій, прийняття рішень, участі в інтерактивних подіях. Twitch Plays Pokemon, масовий соціальний експеримент, де тисячі глядачів керували персонажем у грі Pokemon через команди в чаті. Це показало можливість колективного контролю та спільної гри.

Стрімінгові платформи дозволяють стрімерам використовувати інтерактивні оверлеї та модулі, що додають нові шари взаємодії. Це можуть бути опитування, інтерактивні карти, сповіщення про події в грі або можливість глядачам впливати на ігровий процес за допомогою спеціальних команд. Інструмент який надає такі можливості має назву Streamlabs, з його допомогою відкривається ширший спектр інтеркативної взаємодії між глядачем та стрімером. З цим інструментом додаються сповіщення про нових підписників, донати, виведення чату на екран, та інтеграція з соціальними мережами.

Платформи для стрімінгу заохочують створення інтерактивного контенту, що стимулює розробників ігор додавати нові можливості для взаємодії. Це може бути інтеграція з API стрімінгових платформ, створення ігор, спеціально розроблених для стрімінгу, або додавання інтерактивних режимів до існуючих ігор. Найголовніша гра в цій сфері є JackBox. Гра стала однією з перших яка була розроблена для взаємодії з аудиторією в прямому ефірі. У грі глядачі беруть учать у грі через свої мобільні телефони, або через браузерну версію на комп’ютері, вводять відповіді або голосуючи за ті чи інші варіанти в реальному часі.

Завдяки стріминговим платформам створюються нові спільноти навколо ігор та стрімерів. Це формує тісні зв'язки, залучує глядачів до активної участі на трансляції. Соціальна взаємодія стає важливою складовою інтерактивного досвіду. Завдяки платформі Discord стрімери утворюють спільноти де глядачі спілкуються, взаємодіють між собою не тільки в час трансляцій а й поза ними.

Інтерактивні елементи на стрімінгових платформах відкривають нові комерційні можливості для стрімерів і розробників. Це включає монетизацію через донати, платні підписки, інтегровану рекламу та спонсорські угоди. Інтерактивність допомагає створювати унікальні пропозиції та залучати аудиторію до активної підтримки улюблених стрімерів. Платформа для стрімів Twitch створила свою віртуальну валюту Bits яка є аналогом звичайних донатів, але її особливість полягає в тому що для «донату» цією валютою користувачу не потрібно використовувати сторонні сервіси для оплати, так як він може оплатити прямо на Twitch, і надати більше інтерактивності на прямій трансляції. За допомогою Bits користувач може відправити своє повідомлення стрімеру, замовити пісню або відео. Також поміж відображення на прямій трансляції унікальне повідомлення користувача буде видно і в чаті.

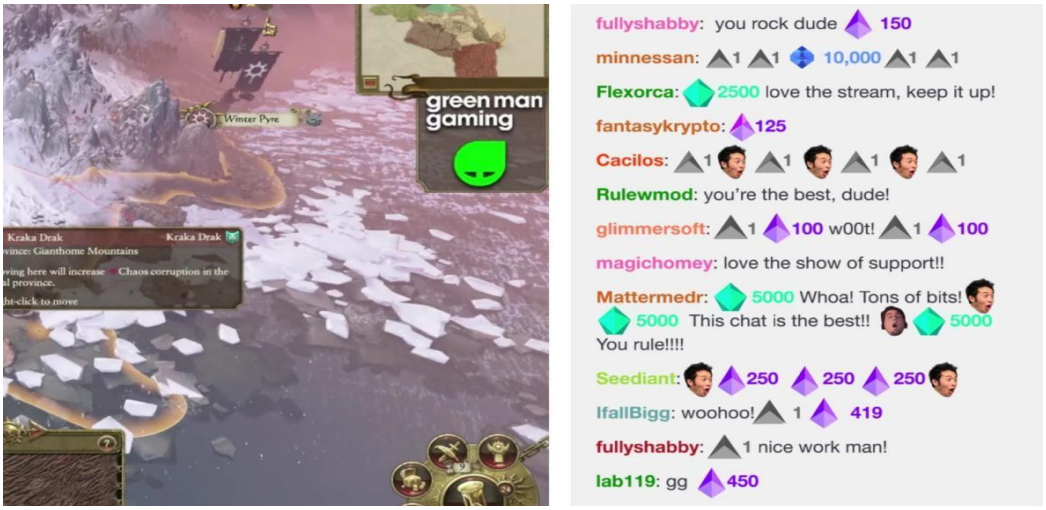


Рис. 1.6 Інтерфейс платформи Twitch та інтеграція Bits шляхом зображення рангів користувачів

1.3 Сучасні технології для інтеграції інтерактивних елементів

Інтерактивність у сучасних відеоіграх стала ключовою складовою, що забезпечує глибокий і захоплюючий ігровий досвід. Розвиток технологій дозволяє розробникам створювати інноваційні та інтерактивні елементи, які підвищують залученість гравців. Ці елементи не лише вдосконалюють геймплей, але й сприяють більшій персоналізації та залученню гравців, роблячи ігри більш інтерактивними та соціальними.

•Розширена Реальність (AR) та Віртуальна Реальність (VR): Технології розширеної реальності дозволяють інтегрувати віртуальні об'єкти у реальний світ, роблячи ігровий досвід більш захоплюючим і інтерактивним.

ARKit від Apple та ARCore від Google - ці платформи надають інструменти для розробки AR-додатків на мобільних пристроях. "Pokémon Go" від Niantic використовує AR, щоб дозволити гравцям ловити покемонів у реальному світі, накладаючи віртуальні об'єкти на реальні сцени через камеру смартфона. Це створює унікальну інтерактивність, де гравці можуть взаємодіяти з ігровими об'єктами, розташованими у їхньому реальному оточенні.

Віртуальна реальність занурює гравця у повністю цифрове середовище, де всі аспекти ігрового світу створюються штучно, що дозволяє досягти максимальної інтерактивності. Oculus Rift, HTC Vive, PlayStation VR ці гарнітури VR дозволяють гравцям повністю зануритися у віртуальний світ. Ігри, такі як "BeatSaber"або "Half-Life: Alyx", використовують контролери руху та просторове відстеження для забезпечення інтуїтивного управління та взаємодії з віртуальним середовищем. Для гри в «Beat Saber» гравці використовують контролери, щоб розбивати блоки під музику, відчуваючи реалістичний зворотний зв'язок через вібрації та тактильні відчуття.

•Хмарні Технології: Хмарний геймінг дозволяє грати у високоякісні ігри без потреби в потужному апаратному забезпеченні на стороні користувача, оскільки всі обчислення виконуються на віддалених серверах. Google Stadia, NVIDIA GeForce Now, Microsoft xCloud: Ці платформи дозволяють гравцям запускати ігри

на будь- якому пристрої з інтернет-з'єднанням, оскільки обчислення та рендеринг виконуються в хмарі. Це не тільки забезпечує доступ до ігор на будь-якому пристрої, але й сприяє інтерактивності через низьку затримку та можливість миттєвої взаємодії з ігровим контенто, Google Stadia надає можливість гравцям миттєво приєднуватися до гри через посилання, поширене на стрімі, без необхідності завантаження чи встановлення.

Інструменти для Стрімінгу Стрімінгові платформи дозволяють глядачам взаємодіяти зі стрімерами у реальному часі, що робить процес перегляду більш залученим і інтерактивним. Twitch Extensions: Ці розширення дозволяють стрімерам додавати інтерактивні елементи до своїх стрімів, такі як опитування, інтерактивні міні-ігри та віджети для взаємодії з аудиторією. Глядачі можуть голосувати за наступний крок стрімера у грі або брати участь у інтерактивних конкурсах, що додає новий рівень залученості та соціальної взаємодії.

- Інтелектуальні Системи та Штучний Інтелект (AI): Технології розпізнавання голосу та обличчя дозволяють створювати більш інтерактивні та персоналізовані ігрові досвіди. Інтеграція голосових помічників у ігри дозволяє гравцям керувати ігровим процесом за допомогою голосових команд. Штучний інтелект може адаптувати ігровий контент відповідно до дій та уподобань гравця, забезпечуючи унікальний та індивідуальний досвід. Використання AI для аналізу поведінки гравця дозволяє створювати адаптивні сценарії, де складність гри або сюжет змінюються залежно від дій гравця. AI може аналізувати стиль гри користувача та автоматично коригувати рівень складності або пропонувати індивідуальні завдання.

1.4 Інтерактивність у мобільних іграх

Інтерактивність у мобільних іграх - це не тільки виконання певних дій на екрані, але і відчуття активної участі у віртуальному світі, створеному розробниками. Це не просто обмежені можливості натискання кнопок чи рухів по екрану, але глибокий взаємодійний процес, який запрошує гравця до власного світу фантазії. У мобільних іграх вона відображається у кожному аспекті від геймплею до візуальної та звукової оболонки. Кожен елемент гри - від персонажів до оточуючого середовища - має відреагувати на дії гравця так, щоб він відчував себе частиною цього вигаданого світу. Також інтерактивність створює можливості для спілкування та взаємодії з іншими гравцями. Це може бути як спільна гра, де кожен гравець вносить свій внесок у загальний результат, так і змагання, де навички та стратегії випробовуються у реальному часі. Більше того, інтерактивність може змінюватися залежно від контексту чи ситуації в грі. Важливо, щоб гра реагувала на дії гравця, змінюючи світ навколо нього або надаючи нові можливості в залежності від виборів, зроблених гравцем.

Інтерактивність в мобільних іграх може виявлятися через широкий спектр різноманітних ігрових механік. Спектр механік досить таки широкий від глибокої системи прогресії та персоналізації персонажа до складних головоломок, рішення яких вимагає від гравця креативності та стратегічного мислення.

Приклади інтерактивних ігор на телефон: Pokemon GO: Використовує розширену реальність для того, щоб гравці могли вирушати у реальний світ та ловити віртуальних покемонів. Гравці можуть рухатися по реальним локаціям, взаємодіяти з іншими гравцями, знаходити покемонів та брати участь у битвах.

1.5 Інтерактивність на Nintendo Switch

Прояв інтерактивності можна розповісти на прикладі ігрової приставки NS - Nintendo Switch. Приставка вийшла в 2017 році та вивела поняття інтерактивності між гравцями на новий рівень.

До комплектації консолі входять два мікроконтролера(Joycons), які використовуються для гри в різних стилях.

Рис 1.7 Контролери Joycons

Інтерактивність полягає в тому, що контролери наділені сучасними технологіями:

- Обидва контролери наділені технологією HD Rumble, що дозволяє налаштовувати вібрації контролерів, що застосовано в іграх на Nintendo Switch. При якихось подіях контролери починають вібрувати з певною силою.
- Датчики руху. Контролери можуть вимірювати відстані, це означає, що вона можуть використовуватись, як ігрове знаряддя і гра буде реагувати на дії гравця, якщо він змінить відстань контролерів до певного об'єкту.
- Інфрачервоний датчик. Правий контролер має вбудований датчик, що

дозволяє йому отримувати певну картину подій, вимірювати відстань.

- Безпровідність. Контролери використовують технологію Bluetooth, тому їх можна використовувати на відстані від приставки.

Контролери joycons надають більше можливостей для інтерактивності під час ігрового процесу, необхідно переглянути деякі приклади, як саме застосовані дані технології:

1-2 Switch: одна з найперших ігор на момент виходу приставки, гра була навмисно створена, щоб продемонструвати технічні можливості контролерів та показати варіативність інтерактивності між гравцями. Гра представляє з себе бібліотеку міні-ігор, які повністю відрізняються від одне одного, та демонструють інтерактивність по різному.

Рис 1.8 Одна з міні-ігор в 1-2 Switch

Гра надає велику бібліотеку ігор, які пов'язані з інтерактивністю, сама гра отримала позитивні відгуки на старті приставки, за весь час було продано понад 3.5 млн. копій гри.

Ring Fit Adventure: гра, що вийшла через 2 роки, після виходу приставки, в 2019 році. це пригодницька гра з елементами фітнесу. Гравець виконує фізичні вправи в реальному житті, які перетворюються на дії персонажа в грі. Мета —

допомогти головному герою пройти різноманітні рівні, подолати ворогів і перемогти головного антагоніста — дракона на ім'я Драгекс, який є втіленням фітнес-бодібілдера.

Гра використовує ексклюзивні для неї аксесуари:

Ring-Con — гнучкий кільцевий контролер, до якого приєднується Joy- Con. Він використовується для вправ верхньої частини тіла, таких як стискання (імітація жиму) або розтягування (робота з м'язами рук і грудей).

Leg Strap — ремінець для кріплення Joy-Con на нозі, який відстежує рухи ніг (біг, присідання, стрибки).

Гра підтримує такі види вправ:

- Кардіо.
- М'язові вправи.
- Вправи для гнучкості.
- Дихальні вправи.

Ring Fit Adventure - новий стандарт на той час, інтерактивних ігор. Гра розроблена таким чином, щоб підвищити імерсивність гравців і одночасно з цим підлаштовуватись під гравця, при цьому цей продукт гейміфіковує фітнес, тому він буде одночасно як цікавим, так і корисним для здоров'я.

Рис 1.9 Гра Ring Fit Adventure

1.6 Інтерактивні фільми

Інтерактивність властива не тільки ігровій індустрії, а також кіноіндустрії.

В цьому виді фільмів сам глядач може вирішувати за героїв фільму, змінюючи хід сюжету в режимі реального часу. Це забезпечується через систему розгалужених сценаріїв, кожен з яких залежить від вибору користувача. Формат найчастіше реалізується за допомогою потокових платформ, відеоігор або спеціальних додатків. На момент написання роботи, найкомпетентнішими платформами для перегляду інтерактивних фільмів є Netflix та Stornaway.io.

Технології реалізцації:

- QTE — швидкі події, які потребують миттєвого вибору.
- Вбудовані інтерфейси у потокових сервісах, які дозволяють робити вибір через пульт дистанційного управління.
- Через мобільні додатки: Натискаючи на відповідні кнопки на екрані для вибору події, що вплине на подальше розвинення подій.

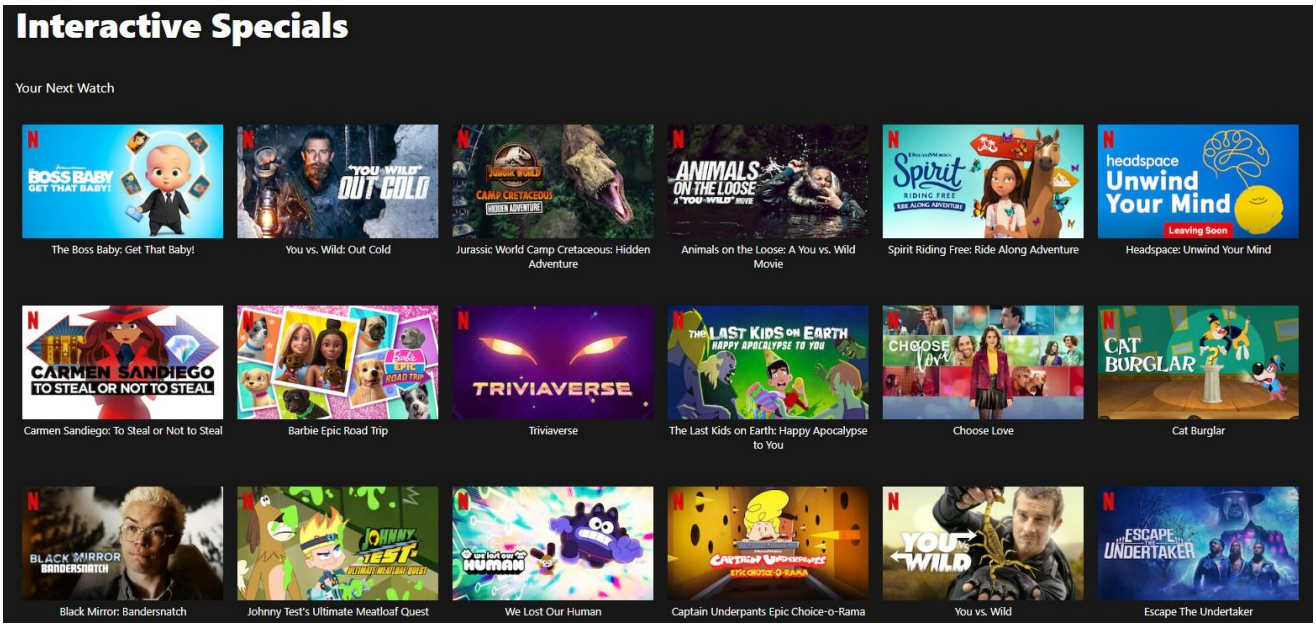


Рис 1.10 Головна сторінка Netflix з інтерактивними фільмами

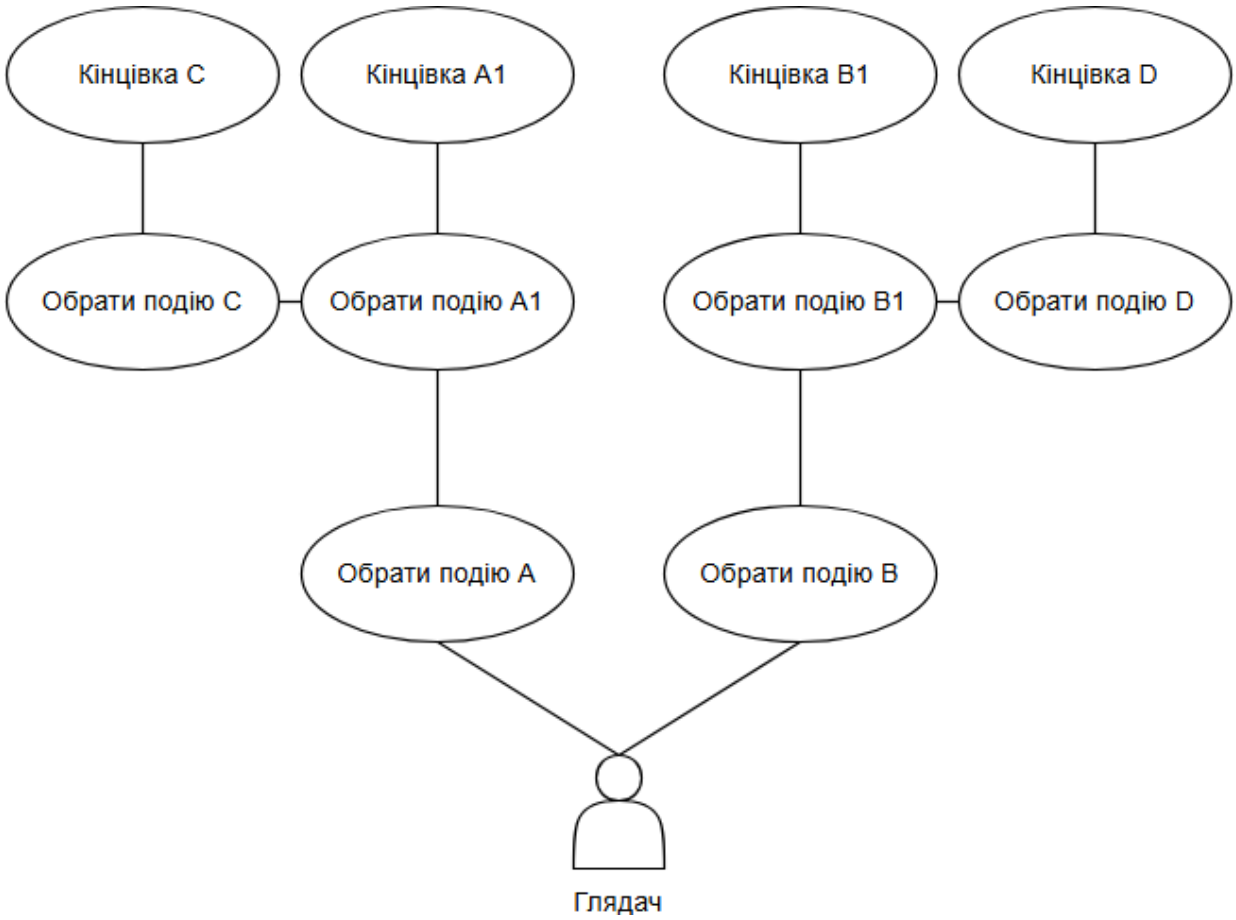


Рис. 1.11 Схема роботи інтерактивних фільмів

Приклади інтерактивних фільмів:

Black Mirror: Bandersnatch: глядач протягом перегляду постійно стикається з виборами, які визначають дії головного героя — молодого програміста, що працює над створенням гри.

Рішення охоплюють як дрібні дії (наприклад, яку пластівці їсти на сніданок), так і життєво важливі кроки (наприклад, убивати чи ні певного персонажа), при цьому фільм має понад 5 основних кінцівок, кожна з яких відображає наслідки виборів глядача. Сюжет розгалужується на десятки сценаріїв, включаючи секретні сцени, які можна відкрити лише за специфічної послідовності рішень.

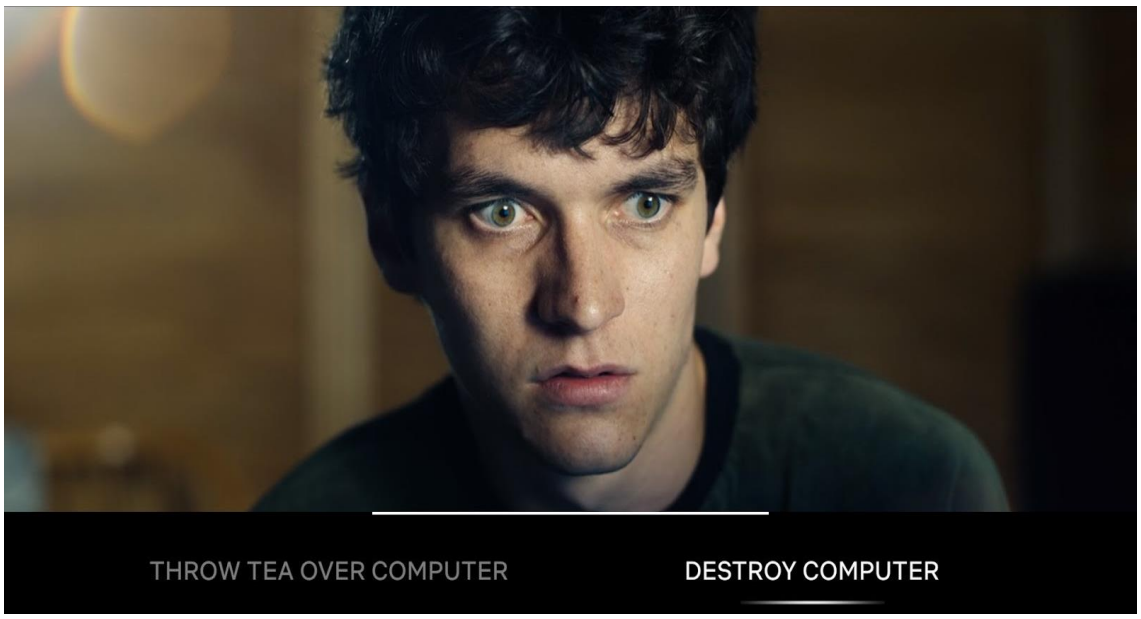


Рис. 1.12 Сцена вибору в Black Mirror: Bandersnatch

Завдяки інтерактивності, глядач стає частиною історії, відчуваючи тиск через відповідальність за долю персонажів. Це формує глибший емоційний зв’язок із сюжетом.

Чорне Зеркало Бандерснатч став самим відмінним прикладом своєї категорії фільмів, що відкрило новий перспективний напрямок в кіноіндустрії та підтвердило важливість інтерактивності та імерсивності глядачів.

Minecraft: Story Mode Netflix Edition: Ще один приклад інтерактиву. Візьмемо гру, яка відноситься до гри Minecraft, яка буде розглядатись в роботі. З самого початку Minecraft Story Mode представляла з себе інтерактивну гру, жанру Point and Click. В силу своєї специфічності гра була також портована на стрімінгову платформу Netflix.

Гравець постійно приймає рішення, які визначають взаємини персонажів, хід подій та деякі сюжетні лінії. Наприклад, вибір союзника чи спосіб вирішення конфлікту впливають на розвиток історії.

Minecraft Story Mode відзначається простим підходом до інтерактивності, що робить його доступним для молодшої аудиторії. Гравець виконує швидкі дії (QTE), відповідає на питання або обирає репліки в діалозі.

Рис. 1.13 Сцена вибору в Minecraft: Story Mode

Велика різниця між Black Mirror та Minecraft Story Mode полягає в тому, що перший фільм є тяжким драматичним інтерактивним хорором, а MCSM - націлений на любу категорію людей, при цьому обидва фільми є інтерактивними і за рахунок підвищують імерсивніть глядача під час перегляду.

Це не єдині приклади своєї категорії і на платформі Netflix присутні й інші роботи, які розміщені на спеціальній сторінці для інтерактивних фільмів, глядач може обрати те, що він бажає і отримати занурючий досвід в перегляді інтерактивних фільмів.

Інтерактивні фільми - цікава технологія, яка підвищує імерсивність глядачів та створює унікальний досвід, але вона має суттєвий недолік - дана технологія на момент написання роботи тільки розвивається, вона потребує більше ресурсів для реалізації, ніж звичайні класичні фільми. Але з розвитком інтерактивних технологій та плином часу вона отримає більше поширення.

Інтерактивні фільми є важливим етапом у переході від традиційного до персоналізованого медіаконтенту. Їхній розвиток відкриває перспективи для створення глибших і складніших історій, які не тільки розважають, але й спонукають до рефлексії. У поєднанні з технологіями віртуальної та доповненої

реальності інтерактивні фільми мають потенціал стати основою нового рівня імерсивного мистецтва, що розширює горизонти медіаіндустрії.

У підсумку, інтерактивні фільми є не лише цікавим інструментом для розповіді історій, а й новою формою взаємодії між мистецтвом і глядачем, що веде до створення унікального досвіду, що залишається в пам’яті на тривалий час.

1.7 Інші приклади інтерактивності

Як вже було визначено, інтерактивність використовується не тільки в ігровій індустрії, а також в кіноіндустрії, для збільшення зацікавленості глядачів в процесі перегляду.

Окрім цього є випадки, коли інтерактивність використовується в благих цілях, розберемо питання на прикладі додатку "Hello Street Cat".

Hello Street Cat - додаток, розроблений китайською компанією Guangxi Hachong Network Technology Co. В квітні 2023 компанія створила унікальну платформу, яка була спрямована на покращення життя безпритульних котів за допомогою взаємодії з користувачами та залучення громади.

Основною ідеєю створення додатку було рішення проблеми популяції кішок в Китаї, особливо в Шанхаї, де проблема перенаселення була на критичному рівні.

Розробники вирішили вирішити проблему гуманним способом, створивши портал для годування кішок, їх піймані та стерилізації. Додаток дозволяв створювати чати між користувачами, обговорювати деталі, ділитись фотографіями котів та новинами. При цьому додаток міг транслювати котів в реальному часі. Користувачі могли коментувати під час прямої трансляції те, що відбувається на екрані та годувати тварин.

Саме годування тварин відбувалось через станції, які в додатку можна сортувати за популярністю, розташуванням чи за необхідністю наповнення кормом. Під час перегляду трансляції користувачі мають можливість взаємодіяти з котами через функцію подачі їжі. Для цього використовуються спеціальні «Жетони любові», які можна придбати у додатку. Такий підхід дозволяє не тільки

підтримувати тварин, але й підвищувати рівень залученості користувачів.

Додаток активно підтримує програму TNR (Trap-Neuter-Return), яка спрямована на гуманний контроль чисельності безпритульних котів. Завдяки цьому, користувачі мають змогу відстежувати прогрес стерилізації, отримувати дані про кількість стерилізованих тварин, а також переглядати профілі котів і статистику діяльності волонтерів. Це сприяє систематичному підходу до вирішення проблеми та створює прозорість у звітуванні про результати.

Окрім цього, додаток має функцію «Котячий сад», де користувачі можуть віртуально «усиновлювати» котів, що відвідують годувальні станції.

Кожна тварина отримує детальний профіль, створений на основі фотографій, що дозволяє індивідуалізувати підхід до кожного кота.

Для користувачів, які прагнуть більшої активності, передбачена можливість стати волонтерами через додаток. Волонтери, яких називають «Людьми-котами», можуть займатися різними завданнями, зокрема виловом котів для стерилізації, обслуговуванням годувальних станцій чи іншою діяльністю. Така інтеграція волонтерської допомоги сприяє залученню більшої кількості небайдужих людей до програми.

Окрім вищезазначеного, додаток пропонує освітні матеріали та аналітичні дані про свій вплив. Наприклад, користувачі можуть бачити, скільки котів стерилізовано чи скільки їжі було надано за певний період. Ця функція не тільки демонструє ефективність діяльності, але й заохочує інших користувачів долучатися до допомоги.

На рік написання даної роботи деякі безпритульні тварини стали популярними в мережі Інтернет, відео на відеохостингу YouTube збирали сотні тисяч переглядів.

Hello Street Cat – це інноваційний соціальний проєкт, який ставить за мету змінити ставлення суспільства до безпритульних тварин. Додаток сприяє формуванню культури турботи та відповідальності, показуючи, що тварини, які опинилися на вулиці, заслуговують на другий шанс і можуть стати чудовими домашніми улюбленцями.

Він акцентує увагу на важливості усиновлення замість купівлі тварин, допомагаючи зменшити кількість безпритульних котів і собак та боротися з проблемою незаконного розведення. Інтегруючи сучасні технології з ідеєю соціальної відповідальності, Hello Street Cat пропонує інтерактивні способи залучення людей до вирішення цієї суспільно важливої проблеми.

Додаток створює спільноту однодумців по всьому світу, об’єднаних спільною метою – допомагати тваринам, робити світ гуманнішим і більш турботливим. Це не лише технологічний продукт, а й потужний інструмент для популяризації емпатії, взаємодопомоги та відповідального ставлення до природи.

Рис. 1.14 Один з найпопулярніших котів, який отримав ім'я Mr Fresh

1.8 Методи інтеграції донату в ігри

Сучасна ігрова індустрія все частіше використовує методи монетизації, засновані на донатах або у внутрішньоігрових покупках. Це пов’язано з тим, що значна частина розробників переходить на безкоштовну бізнес-модель (free-to-play), яка дозволяє залучати велику аудиторію і зберігати високий прибуток.

Основні підходи до впровадження донатів в ігри:

•Продаж товарів, які змінюють зовнішній вигляд персонажів, об’єктів чи інтерфейсу гри, але ніяк не впливають на ігровий процес, також це можуть бути додаткові внутрішні грошові одиниці (монети, алмази, енергія, тощо). Наприклад є популярні скіни в таких іграх, як Fortnite чи Counter-Strike: Global Offensive. Основна перевага цього методу – відсутність впливу на баланс гри, що підвищує довіру гравців.

•Системи "плати за перевагу" (pay-to-win). Продаж предметів чи послуг, які покращують ігрові можливості користувача, наприклад, зброї, що краща за стандартне спорядження, чи ресурсів, що пришвидшують прогрес. Цей метод приносить значний прибуток, але він часто критикується, так створює дисбаланс між платними та безкоштовними гравцями.

•Підписки та сезонні пропуски (battle passes), які гравці купують для тимчасового доступу до ексклюзивного контенту, такого як завдання, предмети або події. Ця модель дозволяє підтримувати стабільний дохід розробників і створює мотивує до регулярної гри. Ці методи використовуються у іграх Apex Legends та подібних.

•Гачкові механіки (loot boxes). Цей метод передбачає купівлю "ящиків", у яких міститься випадковий набір винагород і є досить популярним у таких іграх, як Overwatch. Даний метод часто критикують за схожість із азартними іграми та ризик створення залежності.

•Мікротранзакції для пришвидшення прогресу, де гравці купують ресурси чи предмети, що дозволяють швидше досягати ігрових цілей. Цей підхід зазвичай використовується в мобільних іграх, таких як Clash of Clans чи Candy Crush Saga.

•Додавання у гру рекламних платформ або отримання додаткових переваг у грі за перегляд рекламного ролику.

•Інтеграції з брендами, коли створюються спеціальні пакети у середині гри з брендами партнерами.

Методи інтеграції донатів використовуються залежно від жанру гри. У багатокористувацьких онлайн-іграх (MMORPG) часто використовуються

косметичні предмети або платні послуги для пришвидшення прогресу. У той же час, шутери чи мобільні казуальні ігри активно впроваджують сезонні пропуски та скіни.

Розробники також намагаються забезпечити "вмотивоване витрачання", створюючи умови, де гравці відчувають реальну користь від вкладень, але не стикаються з необхідністю обов’язкових витрат.

Не зважаючи на популярність донатів у іграх, їх інтеграція може викликати негатив у користувачів. Розробникам важливо уникати ситуацій, коли платні гравці мають занадто значну перевагу, мати відкриту інформацію щодо шансів на виграш у випадкових ящиках і не допускати надмірного впливу на вразливі категорії гравців, зокрема дітей.

Інтеграція донатів в ігри є багатогранним процесом, що вимагає ретельного планування, щоб балансувати між фінансовими інтересами розробників і потребами спільноти гравців.

1.9 Грошові інтерактивні елементи на прямих трансляціях

Сьогодні прямі трансляції є важлими для створення контенту і комунікації з аудиторією. Один з факторів їх популярності це інтерактивність, можливість взаємодіяти з підписниками в реальному часі, де вони не тільки пасивно спостерігають за подіями. Також набирає тенденцій впровадження грошових інтерактивних елементів, що дають можливість глядачам фінансово підтримувати авторів контенту та впливати на події трансляції.

Ключові типи грошових інтерактивних елементів:

- Донати (пожертви) – найпоширеніший спосіб грошової взаємодії на стрімах. Глядачі можуть перераховувати кошти автору трансляції як підтримку, часто залишаючи коментарі, які зачитуються чи відображаються в ефірі. Такий підхід підвищує залученість аудиторії та створює особистий зв'язок між автором і глядачами.

- Такі платформи, як Twitch або YouTube, дозволяють глядачам оформлювати платні підписки. Це надає доступ до ексклюзивного контенту, таких як спеціальні емодзі, значки для коментарів, контент виключно для спонсорів чи закриті чати. Такі елементи підвищують відчуття належності до спільноти.

- Суперчати (Super Chat) та наклейки, на платформах, таких як YouTube, глядачі можуть платити за те, щоб їхні повідомлення виділялися в чаті. Це особливо ефективно під час великих трансляцій, коли звичайні повідомлення швидко губляться.

- Гейміфіковані системи пожертв, де глядачі можуть впливати на контент трансляції через донати. Часто до окремих сум стрімери підв’язують аудіозаписи і картинки або ж у рамках стрімів ігор, пожертви можуть додавати або ускладнювати завдання для стрімера.

- У рамках трансляцій стрімери організовують розіграші, участь у яких доступна лише тим, хто задонатив. Такий підхід стимулює більшу кількість глядачів до фінансової підтримки.

Грошові інтерактивні елементи мають переваги як для авторів контенту, так і для аудиторії. Для стрімерів вони забезпечують додатковий дохід, мотивують створювати якісний контент і підвищувати залученість і інтерес глядачів до власного контенту через взаємодію з глядачами. А для глядачів можливість взаємодії надає унікальний досвід участі в контенті, що створює емоційний зв'язок зі стрімером і підвищує лояльність.

Окрім переваг є і ризики, такі як зловживання фінансовою підтримкою від підписників, інколи глядачі можуть образливо або недоречно висловлюватись через коментарі до донатів, тому варто це контролювати фільтрувати даний контент. Для цього стрімери можуть надавати статуси модераторів. Надмірна залежність від донатів може обмежити творчий підхід авторів, змушуючи їх орієнтуватися лише на запити аудиторії і робити контент лише за гроші.

Грошові інтерактивні елементи продовжують еволюціонувати, стаючи ще більш персоналізованими та інноваційними. Використання штучного інтелекту дозволяє аналізувати реакції аудиторії та автоматично адаптувати інтерактивні елементи під їх інтереси. Інший перспективний напрям – впровадження блокчейн-технологій для створення прозорих систем підтримки стрімерів.

2. ЗАГАЛЬНИЙ АНАЛІЗ НЕОБХІДНИХ ТЕХНОЛОГІЙ

2.1 Стрімінгові сервіси

Стрімінгові платформи відіграють важливу роль у сучасному цифровому середовищі, дозволяючи користувачам дивитися стрімера в реальному часі. Для організації стрімів необхідні програми та налаштувати параметри ведення трансляції та безпосередньо самий стрімінгова платформа, на якому буде вестись трансляція.

Стрімінгових платформ на момент написання роботи достатньо багато, але найкрупнішу та найзручнішими для роботи є YouTube та Twitch, обидві платформи мають потужні інструменти для трансляції та взаємодії з аудиторією, що сприяє розвитку нових форм контенту та залучення глядачів. особливо Twitch, коли справа заходить про інтерактивність між глядачами та стрімером через чат. Обидві платформ можуть зберігати запис ефіру, мають чат для глядачів, та підтримують взаємодію з різними сервісами, як StreamLabs та Tipeeestream для налаштування донатів, ефектів та інших аспектів.

Головною особливістю платформи Twitch являється підтримка розширень, які можуть бути написані іншими розробниками, додаючи унікальності стріму.

Окрім цього існує система валюти - Bits. Вона надає можливість відправляти ексклюзивні емодзі в чат, надаючи унікальності повідомленню, за допомогою розширень Twitch, існує можливість працювати з Bits розширено, наприклад складати таблиці з глядачами, що найбільше підтримали валютою стрімера. За кожну одиницю Bits стрімер отриимує 1 цент.

Головною особливістю платформи YouTube являється технологія VOD, обидві платформи можуть зберігати записи, але в YouTube він може з легкістю бути доступним записаним і бути доступним для глядачів в любий момент, коли на Twitch відео зникають, якщо їх не встигнути архівовувати. Також YouTube надає можливість інтеграції з системою монетизації AdSense.

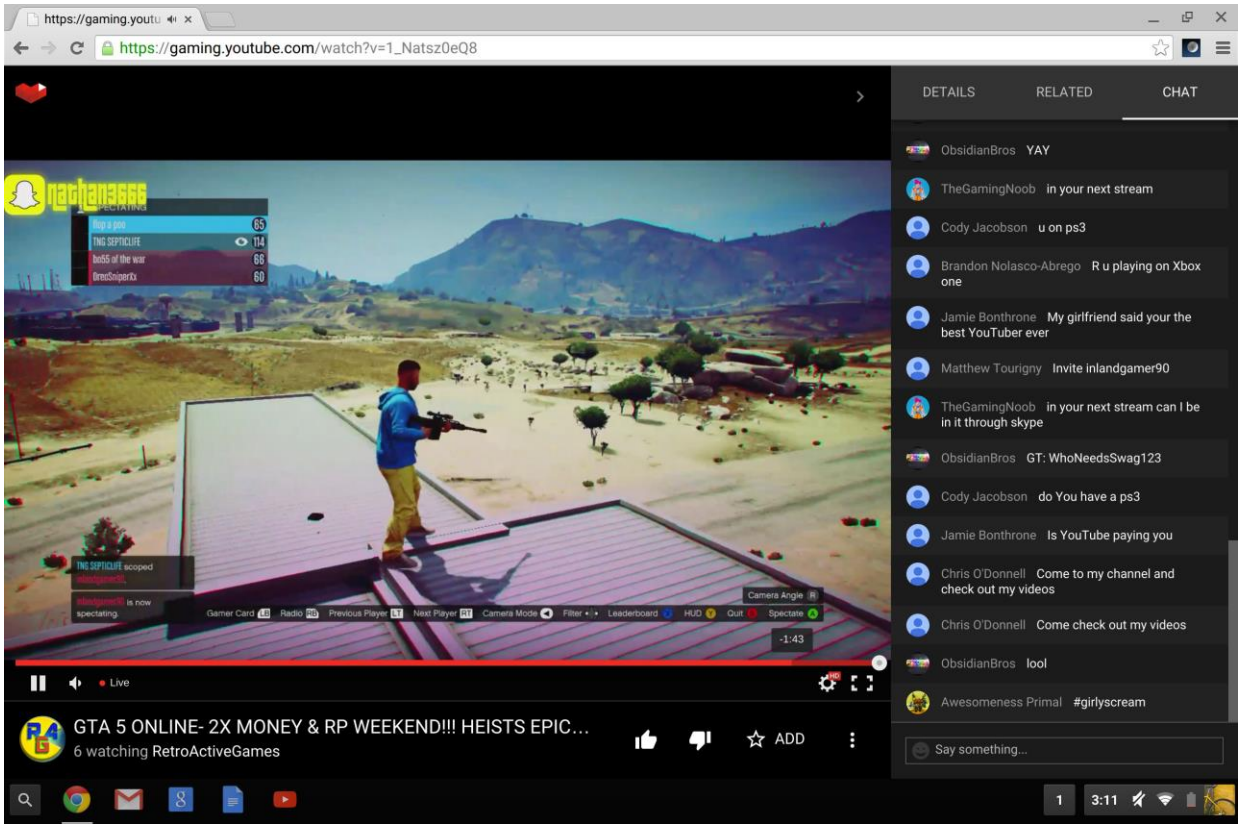


Рис. 2.1 Інтерфейс прямої трансляції на YouTube

2.2 Інструменти для стрімінгу

Для організації та налаштувань прямих трансляцій тільки YouTube та Twitch буде недостатньо, на момент написання роботи через ці платформи можливо почати пряму трансляцію напряму тільки з мобільних додатків, тому існують спеціалізовані інструменти та програми, які вирішують дану проблему, вони надають можливість почати пряму трансляцію та надають інструменти для їх керуванням. Одним із найпопулярніших рішень є OBS Studio — безкоштовний метод, який надає інструменти для налаштування сцен, джерел відео та звуку, а також управління потоками. Воно підтримує інтеграцію з основними стрімінговими платформами, забезпечуючи гнучкість і професійний рівень якості.

Окрім OBS Studio, варто звернути увагу на Streamlabs, яке надає вбудовані інструменти для інтерактивності, шаблони для оформлення та автоматизацію роботи зі стрімами. Для тих, хто потребує додаткових функцій, таких як багатокамерні трансляції чи віртуальна студія, доступний XSplit. Додатковими

прикладами застосунками для стрімінгу можуть бути Tipeestream та Restream.

Ці інструменти дозволяють покращити якість прямих ефірів, налаштувати індивідуальний стиль, забезпечити високий рівень взаємодії з аудиторією, що неможливо зробити лише за допомогою стандартних функцій YouTube або Twitch.

Серед застосунків для стрімінгу особливою популярністю користується OBS Studio — універсальний інструмент, який підходить як новачкам, так і досвідченим стримерам. Це безкоштовний метод з відкритим кодом забезпечує потужні можливості для створення багатошарових сцен, налаштування джерел відео та аудіо, а також управління потоками.

Рис. 2.2 UI застосунку OBS Studio

Гнучкий інтерфейс OBS Studio дозволяє адаптувати програму під будь-який формат трансляції, а підтримка плагінів відкриває доступ до розширених функцій, таких як інтеграція з додатковими сервісами чи налаштування віртуальних студій. Сумісність із основними платформами, такими як YouTube, Twitch, що робить OBS Studio ключовим інструментом для забезпечення якісних і стабільних стрімів. Крім того, OBS Studio підтримує запис відео локально, що дозволяє створювати контент для подальшого редагування та публікації.

Програма працює на Windows, macOS та Linux, забезпечуючи кросплатформену доступність. Завдяки своїй гнучкості та надійності, OBS Studio є одним із найкращих виборів для організації професійних стрімів.

Таблиця 2.1

Порівняння програмних засобів для проведення стрімів

Функція	OBS Studio	Streamlabs	XSplit
Простота використання	Висока, але потребує базового налаштування	Дуже висока, зручний інтерфейс	Середня, інтуїтивний інтерфейс, але багато опцій
Функціональність	Максимально гнучка, підтримка плагінів	Вбудовані інтерактивні інструменти	Потужні функції, зокрема багатокамерні трансляції
Сумісність	Windows, macOS, Linux	Windows, macOS	Тільки Windows

З таблиці видно, що OBS Studio є беззаперечним лідером серед застосунків для стрімінгу, коли йдеться про поєднання потужності, універсальності та доступності. Безкоштовна та кросплатформенна, вона забезпечує всі необхідні інструменти для створення професійних трансляцій, залишаючись водночас відкритою для модифікацій. У порівнянні з конкурентами, OBS Studio пропонує максимальну гнучкість і можливість налаштування, що робить її незамінною для тих, хто прагне повного контролю над своїми ефірами.

Особливу увагу варто звернути на підтримку плагінів, які відкривають додаткові горизонти для творчості та оптимізації стрімів, перетворюючи OBS Studio на справжній швейцарський ніж для стримерів.

2.3 Методи та модифікації

Методи та модифікації — це невід'ємна частина сучасного програмного забезпечення, яка дозволяє значно розширити базову функціональність застосунків. Вони дають змогу налаштовувати інтерфейс, додавати нові інструменти та можливості, автоматизувати процеси і адаптувати програму під конкретні потреби користувача. Завдяки методам можна інтегрувати різноманітні сторонні сервіси, створювати унікальні ефекти та зручності, яких не вистачає в стандартному наборі інструментів.

Методи можуть бути як простими додатками, що додають одну-дві нові функції, так і комплексними модулями, що повністю змінюють спосіб взаємодії з програмою. Наприклад, це можуть бути методи для покращення продуктивності, забезпечення сумісності з іншими додатками або додавання нових візуальних і аудіоефектів. Вони дають змогу користувачам налаштовувати робоче середовище під свої вимоги, роблячи програму потужнішою та зручнішою у використанні.

Модифікації ж часто спрямовані на зміни базового коду програми для досягнення більш глибокої кастомізації. Це може включати як незначні оновлення, так і великі оновлення інтерфейсу або навіть повністю нові функціональні блоки. Часто модифікації застосовуються для вирішення специфічних завдань, яких неможливо досягти стандартними методами.

Суть методів і модифікацій у тому, що вони дозволяють кожному користувачеві налаштувати програму відповідно до своїх особистих потреб, роблячи її максимально ефективною і відповідною до специфіки роботи. Вони є невід'ємною частиною екосистеми програмного забезпечення, що дозволяє створювати ідеальні інструменти для конкретних завдань, спрощуючи роботу й відкриваючи нові можливості для користувачів.

Таблиця 2.2

Можливості методів та модифікацій

Можливість	Методи	Модифікації
Можуть додавати нові функції, не змінюючи основний код програми	+	+
Можуть додавати нові функції, змінюючи основний код програми	-	+
Не складні в написанні	+	-

Методи та модифікації — це чудові інструменти для доповнення можливостей застосунків ігор, які дають користувачам змогу адаптувати програми під власні потреби.

Обидва ці підходи підходять для вирішення різноманітних завдань, однак існує суттєва різниця між ними, яка стосується їхньої реалізації та доступності. Не всі програми підтримують модифікації, а лише дозволяють створювати методи. Наприклад, такі програми, як Visual Studio, OBS Studio та IntelliJ IDEA,Twitch Extensions надають користувачам можливість інтегрувати додаткові методи, що значно розширює їх функціональність.

З іншого боку, існують програми та ігри, які взагалі не підтримують модифікації або створення методів. У таких випадках користувачі обмежені стандартними функціями, і неможливо розширити їх за допомогою сторонніх інструментів. Програми, які не підтримують модифікації, зазвичай мають фіксовану архітектуру, що ускладнює адаптацію під індивідуальні потреби користувача. З ще однієї сторони, деякі програми не мають підтримки методів, але можуть бути модифіковані.

Тим не менше, вибір між використанням методів чи модифікацій залежить від конкретного об'єкта, з яким працює користувач. Якщо програма дозволяє створення методів, це буде оптимальним способом розширення її можливостей без втручання в базовий код. Модифікації, в свою чергу, зазвичай використовуються тоді, коли методи не можуть задовольнити потреби користувача або коли потрібно

змінити глибші аспекти роботи програми. Однак, варто зазначити, що модифікації часто вимагають більш глибоких технічних знань і можуть змінювати стабільність програми.

Отже, вибір між методами та модифікаціями залежить від рівня доступу до внутрішніх механізмів програми, цілей користувача та доступних інструментів для розширення функціональності.

У світі стрімінгу методи та модифікації стали важливими інструментами для покращення взаємодії між стрімерами та їхньою аудиторією. Вони дозволяють розширювати функціональність платформ, таких як Twitch, YouTube, та застосунків для стрімінгу, як OBS, додаючи нові можливості та автоматизацію процесів та інтерактивності як між глядачами, так і між стрімером.

StreamElements – один з найпопулярніших методів, що дозволяє значно збільшити інтерактивність стрімів завдяки багатьом функціям. Цей метод дозволяє створювати інтерактивні віджети, що відображають дії глядачів, наприклад, підписки, донати або коментарі в чаті. Але найголовніше – це підтримка інтерактивних ігор та голосувань у чаті, де глядачі можуть впливати на те, що відбувається на екрані, наприклад, вибирати рішення для стрімера в іграх або впливати на обрану гру через голосування.

Інтерактивність проявляється таким чином:

- Глядачі можуть голосувати за те, яку гру стрімер буде грати наступною.
- Взаємодія з донатами та підписками через анімації, які з'являються на екрані в реальному часі, мотивуючи глядачів активно підтримувати стрім.

Marbles on Stream – дозволяє глядачам брати участь у "гонках кульок" під час трансляцій на Twitch. Глядачі можуть вводити команду в чат і брати участь у гонці, під'єднуючись до стрімера. Під час гонки кульки, що належать глядачам, рухаються на екрані, а переможець отримує приз. Це дуже простий, але захоплюючий спосіб залучити аудиторію і зробити стрім більш інтерактивним.

Crowd Control – дозволяє глядачам впливати на гру через донати або інші інтерактивні елементи. Під час трансляцій стрімер може налаштувати певні події, які активуються після того, як глядачі роблять донати або виконують інші умови.

2.4 Прикладні програмні інтерфейси

Прикладні програмні інтерфейси (API) - це набір методів і протоколів, що дозволяють різним програмним продуктам взаємодіяти, зокрема через веб- сервіси. Вони визначають набір правил для запиту та обміну даними між програмами. API забезпечує спосіб для різних застосунків і сервісів обмінюватися даними або виконувати операції без необхідності вручну втручатися в кожен окремий процес. По суті, API визначає, як один компонент програмного забезпечення може взаємодіяти з іншим, вказуючи на точний формат запитів, відповіді та методи доступу до ресурсів. Це може бути як запит на отримання даних, так і на виконання певної дії, наприклад, оновлення інформації, ініціація процесів тощо.

API можна класифікуватии на такі типи:

API баз даних: дозволяють взаємодіяти між додатками та системами керування базами даних. За допомогою цих API розробники можуть виконувати запити для отримання або зміни даних, а також керувати структурою бази даних.

API операційних систем: забезпечують додатки доступом до ресурсів і послуг, які надає операційна система. Ці API дають змогу програмам використовувати такі системні ресурси, як пам’ять, файли, пристрої та процеси. Кожна операційна система має власний набір унікальних API, наприклад, Windows API або Linux API.

Віддалені API (Remote APIs): дають змогу додаткам взаємодіяти з ресурсами, що знаходяться на різних машинах або пристроях. Ці API визначають правила для обміну даними між додатками, що працюють на віддалених системах. Важливим аспектом таких API є те, що клієнт і сервер розташовані фізично на різних машинах, що дозволяє здійснювати обмін даними через мережу.

Web API : забезпечують обмін даними та функціональністю між клієнтськими і серверними додатками через веб. Ці API працюють через HTTP і дозволяють веб-клієнтам (наприклад, браузерам) надсилати запити на сервери, а потім отримувати відповіді. Web API активно використовуються для реалізації хмарних сервісів, мобільних додатків та веб-додатків.

RPC (Remote Procedure Call): є протоколом, який дозволяє клієнтському додатку запитувати дані або послугу від серверного додатку, що працює на іншій

машині. Це дає можливість взаємодії між програмами, що знаходяться на віддалених машинах, і дозволяє клієнту використовувати сервер як локальний об'єкт. RPC визначає правила для таких взаємодій, забезпечуючи їх правильне виконання між додатками в мережі.

SOAP (Service Object Access Protocol): протокол для передачі структурованої інформації в розподіленому середовищі. SOAP API визначають правила для створення повідомлень запиту та відповіді в форматі XML. Цей протокол часто використовується в корпоративних додатках, де важливі висока безпека і цілісність даних. SOAP підтримує комунікацію через HTTP або SMTP.

REST (Representational State Transfer): архітектурний стиль для побудови API, який забезпечує ефективну комунікацію між системами через веб. На відміну від SOAP, REST є набором принципів, що визначають безстатеве спілкування, де кожен запит від клієнта містить усю необхідну інформацію для його обробки сервером. RESTful API широко використовуються для створення веб-сервісів, пропонуючи простоту і масштабованість.

gRPC (gRPC Remote Procedure Call): відкритий фреймворк для побудови API, який використовує принципи RPC для взаємодії між мікросервісами. Розроблений Google і випущений в 2015 році, gRPC дозволяє клієнтським додаткам викликати методи серверного додатку, немов вони є локальними об'єктами. Це спрощує розробку розподілених систем, оскільки взаємодія між сервісами стає швидшою і ефективнішою завдяки використанню структурованих даних через Protocol Buffers.

У контексті стрімінгу, API дозволяють стрімерам інтегрувати різні функції, такі як сповіщення про підписки або донати, інтерактивні чати, голосування чи зміна ігрового процесу в реальному часі. Це створює взаємодію між стрімерами та їхньою аудиторією, надаючи їй можливість впливати на події трансляції. За допомогою API можна налаштувати програми для того, щоб глядачі могли надавати команди або змінювати елементи трансляції, тим самим активно долучаючись до процесу. Донат сервіси, такі як Donatello, Dyaka та банки, такі як Monobank, Twitch, YouTube, OBS мають відкритий API, який можна поєднати між собою, а ще маючи можливість створювати плагіни та модифікації для ігор - це все можна поєднати і отримати максимально інтерактивну трансляцію з високим рівнем імерсивності.

3. СТВОРЕННЯ МЕТОДУ ДЛЯ ІНТЕРАКТИВНИХ ДОНАТІВ

3.1 Перелік необхідних технологій

В часи становлення та популяризації українського контенту в мережі, гостро стоїть питання підтримки градусу зацікавленості, недопущення повернення глядачів, що коливаються в пост радянський простір, що досяжно завдяки нарощенню виробництва контенту та його монетизації.

На разі утримання глядача є великою проблемою, оскільки в сучасному світі важко заохотити когось переглянути певний контент, навіть якщо його було зроблено з усіма побажаннями споживача контенту.

Сучасні стримінгові сервіси пропонують стандартні опції як для глядача, так і для мтримера. А саме:

- Перегляд;
- Живий чат;
- Спонсорство;
- Підписка на канал.

Однак, цих стандартних функцій часто недостатньо для довготривалого утримання аудиторії. Сучасний глядач прагне інтерактивності, особистої залученості та нових форматів взаємодії. Для цього потрібні інструменти, які дозволяють створювати унікальний досвід для кожного користувача. В минулому розділі було проаналізовано такі технології для досягнення рішення проблеми:

API: Дозволяє інтегрувати сторонні сервіси та платформи у стрімінговий процес. У нашому випадку ми активно використовуємо API донатних сервісів, які забезпечують передачу інформації про донати в реальному часі.

Це дає змогу не лише відображати повідомлення від глядачів на екрані, але й інтегрувати донати у взаємодію з іншими елементами стріму, наприклад, запускати спеціальні ефекти, змінювати стани ігрових подій чи виводити статистику. Завдяки API можна налаштувати передачу даних з високою точністю, забезпечуючи

стабільну роботу системи навіть при великих обсягах інформації.

Плагіни: Плагіни є невід'ємною частиною сучасних стрімінгових платформ, адже вони дозволяють розширювати стандартний функціонал програмного забезпечення. У нашому випадку плагіни забезпечують зв'язок між стрімерськими методами або грою зі серверами донатних сервісів, це додає можливості для інтерактивної взаємодії з глядачами, або з грою через сервер, якщо ігровий відбувається через нього.

Модифікації**:** На відміну від плагінів, які додають новий функціонал до існуючого програмного забезпечення, модифікації передбачають внесення змін безпосередньо в код програми або гри. У нашому випадку ми модифікуємо код ігрового середовища, щоб адаптувати його під інтерактивні сценарії, якщо розробники гри надали можливість модифікувати код гри.

Комбінація цих трьох технологій дозволяє досягти максимальної гнучкості та інтерактивності під час стрімів. Завдяки API стрімер отримує можливість працювати з актуальними даними від сторонніх сервісів. Плагіни спрощують налаштування нових функцій, розширюючи можливості стрімінгових платформ. А модифікації забезпечують унікальні ігрові сценарії, які інтегруються в загальну концепцію стріму.

3.2 Метод DonateEvents

Для демонстрації роботи метода було обрано гру Minecraft, яка є однією з найпопулярніших ігор жанру Sandbox. Вибір саме цієї гри обумовлений її широкими можливостями для інтеграції методів і модифікацій, а також відкритим кодом, який дозволяє адаптувати геймплей під різноманітні потреби.

Minecraft має унікальну архітектуру, яка забезпечує повну підтримку методів та модифікацій. Це дозволяє реалізувати складні сценарії взаємодії між грою та глядачами.

Метод DonateEvents побудований на використанні API донатних сервісів, що забезпечує стабільний і швидкий обмін даними між серверами донатної платформи

та грою. Для обробки інформації використовується асинхронний підхід, що дозволяє уникнути затримок і забезпечити плавну роботу системи навіть при великій кількості запитів.

Важливим елементом є можливість налаштування методу під конкретні потреби стрімера. Пдагін дозволяє вибирати, які саме події будуть активуватися в залежності від розміру донату або типу повідомлення, а також можна додавати свої кастомні події.

Ціль методу:

- Підвищення залученості глядачів. Глядачі отримують можливість впливати на гру, що робить трансляцію більш захоплюючою.
- Збільшення доходів стрімера. Завдяки інтерактивності глядачі частіше роблять донати, оскільки це дає їм змогу брати активну участь у грі.
- Персоналізація контенту. Кожен стрім стає унікальним завдяки сценаріям, створеним за участю аудиторії.
- Розширення функціоналу гри. Метод додає нові можливості, які недоступні в базовій версії Minecraft.

Таблиця 3.1

Порівняння вже існуючих методів з розробленим

Методи	Легкість налаштування	Можливість кастомізації	Реакція в реальному часі	Можливість донату за допомогою українських банків та донат сервісів	Інтеграція з різними платформами	Взаємодія зі стрімерами	Аналіз дій користувачів
DonateIvents	Простий інтерфейс із покроковими інстукціями для налаштування.	Дозволяє змінювати візуальний стиль донатів (колір, текст, анімації).	Відображення донатів на стрімі без затримок.	Інтеграція з Monobank, ПриватБанк, LiqPay.	Підтримка YouTube, Twich.	Включає додаткові функції взаємодії.	Звіти з деталями про суми та активність донатерів.

Продовження таблиці 3.1

Порівняння вже існуючих методів з розробленим

Методи	Легкість налаштування	Можливість кастомізації	Реакція в реальному часі	Можливість донату за допомогою українських банків та донат сервісів	Інтеграція з різними платформами	Взаємодія зі стрімерами	Аналіз дій користувачів
TwichCraft	Інтуїтивно зрозумілий процес налаштування через інтерфейс Twich.	Обмежений вибір шаблонів, мінімальні налаштування.	Середня швидкість відображення донатів із затримкою до 5 секунд.	Не підтримується, працює лише з міжнародними платформами на кшталт PayPal.	Обмежена лише Twich.	Проста взаємодія через прості повідомлення від донатерів.	Відсутній або обмежений базовими логами.
MC Interactive	Складна процедура налаштування для новачків.	Мінімальні опції, кастомізація переважно недоступна.	Реакція залежить від роботи сервера ,часто бувають затримки.	Відсутня, працює з криптовалютою та міжнародними сервісами.	Підтримка YouTube, Twich.	Донати безпосередньо інтегруються з ігровими подіями.	Лише базові логи, активності на сервері.
StreamingDrops	Проста процедура налаштування через зрозумілий вебінтерфейс	Можливість змінювати лише базові елементи дизайну.	Висока швидкість реакції завдяки інтеграції API.	Не підтримується, доступні лише міжнародні сервіси.	Обмежена стрімінговими платформами, YouTube, Twich.	Інтерактивні елементи обмежені простою подякою.	Відсутній інструмент аналітики.

Виходячи з таблиці існуючі методи не підтримують українські донат-сервіси та банки, що створює складність в їх використанні для досягнення імерсивності на стрімі.

Minecraft, одна з найвідоміших і найпопулярніших ігор у світі, була написана мовою програмування Java. Цей вибір технології забезпечив їй надзвичайну гнучкість і кросплатформеність. Завдяки Java, гра може працювати на різних

операційних системах, таких як Windows, macOS та Linux, без необхідності внесення значних змін у вихідний код. Мова Java, відома своєю модульністю та підтримкою широкого кола інструментів, дозволяє легко створювати, інтегрувати та впроваджувати новий функціонал для Minecraft у вигляді методів або модифікацій.

Формат файлів модифікацій і методів для Minecraft базується на стандарті .jar, що є нативним для Java. Це формат стиснених архівів, який містить зкомпільовані класи Java, файли конфігурацій та ресурси, необхідні для роботи розширень. Розробники створюють методи, додаючи додатковий функціонал у гру, змінюючи її механіку або розширюючи можливості серверів Minecraft.

Для роботи методів або модифікацій потрібне спеціальне серверне ядро. Ядра слугують платформою для запуску методів і забезпечують API, через які методи можуть взаємодіяти з ігровим світом. Серед основних серверних ядер для Minecraft можна виділити наступні:

- Bukkit — основа для створення методів із простим API, яке підходить для новачків.
- Spigot — вдосконалена версія Bukkit із поліпшеною продуктивністю та розширеними можливостями.
- Paper — ще більш оптимізоване ядро на базі Spigot, яке забезпечує вищу продуктивність і додатковий функціонал.

У нашому випадку було обрано серверне ядро Paper, яке поєднує високу продуктивність із широкими можливостями для розробників. Paper базується на Spigot, але додає свої унікальні поліпшення. Це ядро забезпечує стабільну роботу серверу навіть при великій кількості гравців і складних обчисленнях.

Таблиця 3.2

Переваги та недоліки плагінів для розробки методу

Переваги	Bukkit	Spigot	Paper
Має активну спільнота розробників	Помірна, але вже застаріла	Активна	Дуже активна, із частими оновленнями
Оптимізація	Базова	Покращена порівняно з Bukkit	Максимальна, підходить для великих серверів
Можливість API	Обмежена	Розширена	Найбільше розширення API
Зворотня сумісність	Підтримує власні плагіни	Підтримує плагіни Bukkit	Підтримує плагіни Bukkit і Spigot

Bukkit можна розглядати як платформу, що слугувала основою для сучасніших рішень, але її актуальність значно знизилася через зупинку активного розвитку. Помірна спільнота розробників не здатна забезпечити необхідного рівня інновацій та швидких оновлень, що особливо важливо для сучасних серверів Minecraft. Її базова оптимізація означає, що Bukkit добре працює на невеликих серверах або для простих завдань, але недостатньо ефективна для великих чи ресурсомістких серверів, де потрібна краща продуктивність. Обмеженість API платформи робить її менш привабливою для розробників, які бажають створювати складні та унікальні плагіни. Водночас сумісність із власними плагінами дозволяє підтримувати певну спадковість, що може бути корисним для тих, хто вже має досвід роботи з цим середовищем.

Spigot, як вдосконалення Bukkit, став значно популярнішим завдяки активній підтримці та оновленням. Спільнота розробників залишається динамічною, що забезпечує швидке вирішення помилок і додавання нових функцій. Покращена оптимізація робить платформу придатною для роботи з більш навантаженими серверами, забезпечуючи кращу продуктивність і ефективне використання ресурсів. Розширені можливості API дають змогу розробникам створювати більш гнучкі та функціональні плагіни, що значно розширює потенціал кастомізації серверів. Зворотна сумісність із плагінами Bukkit гарантує плавний перехід для користувачів, які хочуть оновити серверну платформу, не втрачаючи доступу до старих рішень.

Paper являє собою вершину еволюції цих платформ, пропонуючи найвищий рівень оптимізації, функціональності та актуальності. Її дуже активна спільнота розробників забезпечує швидкий випуск оновлень, що важливо для підтримки сумісності з останніми версіями Minecraft і для інтеграції нових функцій. Paper спеціалізується на максимальній оптимізації, що дозволяє значно підвищити продуктивність навіть на великих і високонавантажених серверах. Це особливо корисно для проєктів із великою кількістю гравців або складними ігровими механіками. API Paper є найбільш розширеним, відкриваючи необмежені можливості для розробників. Вони можуть створювати унікальні плагіни, які додають новий рівень інтерактивності та кастомізації. Paper також підтримує плагіни Bukkit і Spigot, що робить її універсальним вибором, придатним як для нових користувачів, так і для тих, хто хоче перейти на більш сучасну платформу, зберігши старі напрацювання.

Таблиця 3.3

Порівняння API донат-сервісів

API	Donatello	Dyaka	Monobank
Працює з методами	+	+	Технічно + Але швидко блокується
Мають UUID	+	-	+
Рівень складності документації	Базова	Базова	Детальна, з прикладами з інших мов
Можливості API	Інформація про донати, статистика	Інформація про донати, обробка зборів	Перекази, перевірка балансу, вебхуки, управління рахунками
Швидкість відповіді API	Швидка	Середня	Дуже швидка
Вебхуки	Є, але лише для нових донатів	Обмежені	Є, з гнучкими налаштуваннями
Масштабованість	Підходить для невеликих стрімів	Підходить для невеликих зборів	Підтримує великі обсяги запитів
Аутентифікація	Токен доступу	Токен доступу	OAuth 2.0
Інтеграція	Twitch, YouTube	Twitch, YouTube	Зовнішні сервіси через API
Формат даних	JSON	JSON	JSON, XML
Додаткові можливості	Статистика зборів	Статистика зборів	Аналітика транзакцій, кастомізація запитів

В даній таблиці було порівняно API донат-сервісів Donatello, Dyaka та єдиного банку з відкритим API - Монобанку. На момент написання роботи метод працює тільки з API Donatello.

API Dyaka не надає можливості створювати UUID, що не є зручним і не дає можливості відстежити повторний донат.

API Monobank надає більше можливостей, але API-токен з певних причин блокується і не надає можливості працювати з ним далі.

Тому робота з грою відбувається в поєднані зі донат-сервісом Donatello.

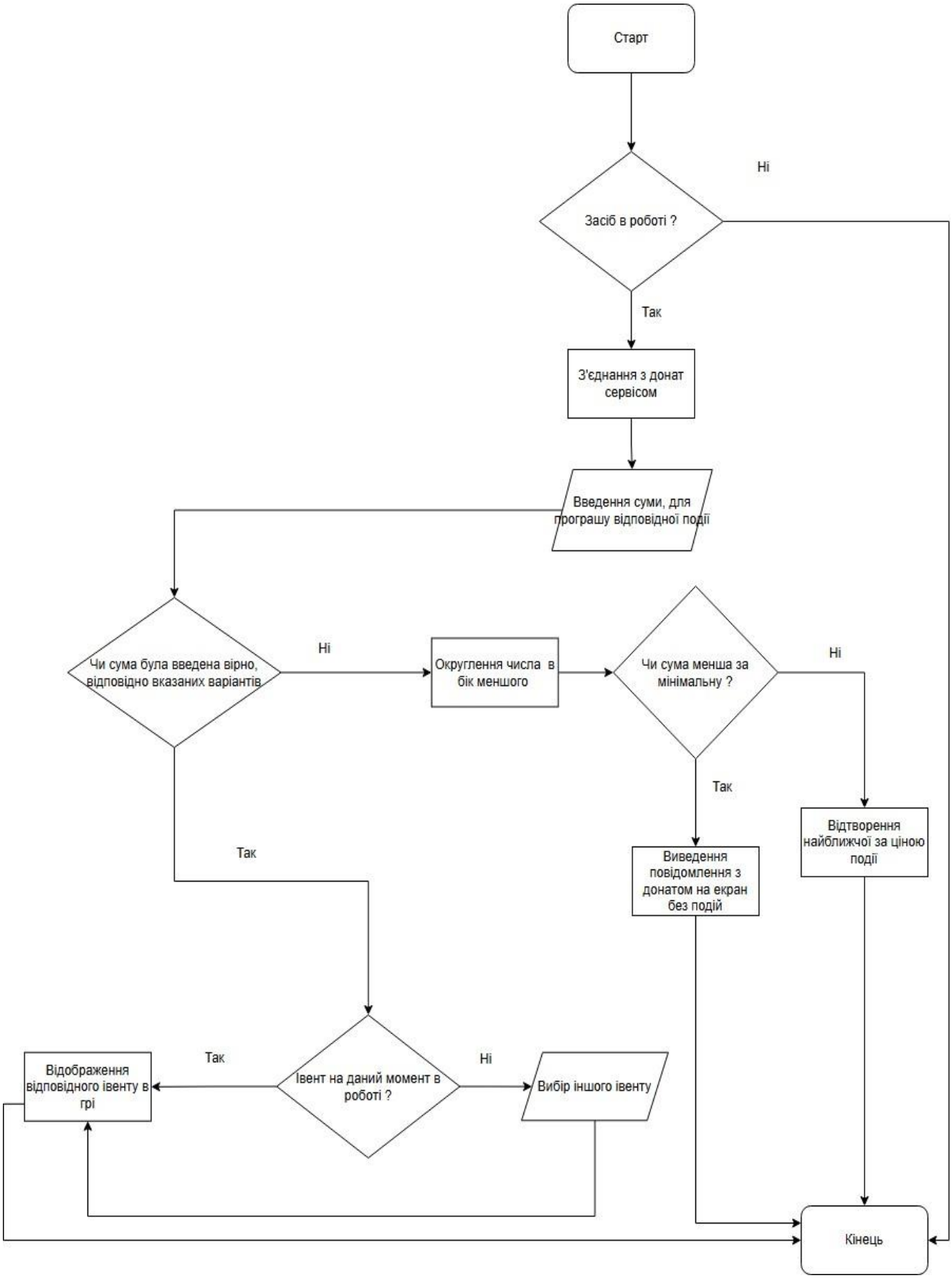


Рис. 3.1 Блок схема алгоритму методу DonateEvents

Метод DonateEvents розроблено в Intellij Idea - інтегрованому середовищі розробки (IDE) від компанії JetBrains. IntelliJ IDEA є одним із найпопулярніших інструментів для написання програмного забезпечення, особливо на мові Java. Ця IDE надає потужний функціонал для розробників завдяки поєднанню продуктивності, зручності використання та багатого набору інструментів.

Основна особливість IntelliJ IDEA полягає в інтелектуальному аналізі коду.

IDE забезпечує автоматичне завершення коду, перевірку синтаксису в режимі реального часу, а також глибокий аналіз залежностей у проектах. Це особливо важливо для розробки методів, які працюють з великими кодовими базами, де будь-яка помилка може спричинити неполадки під час виконання програми.

Для розробки метода було обрано IntelliJ IDEA завдяки її широкому функціоналу для роботи саме з Java-проєктами. IDE спрощує процес налагодження, надаючи можливість точкового запуску методу в тестовому середовищі. Крім того, у IntelliJ IDEA реалізовано зручний інтерфейс роботи з Maven і Gradle – системами автоматизації збірки, які використовуються для управління залежностями.

У контексті створення метода для інтеграції у Minecraft важливими виявилися такі можливості IntelliJ IDEA:

- Підтримка розробки багатомодульних проєктів, що спрощує інтеграцію з іншими методами або бібліотеками, такими як API ядра Paper.
- Зручна робота з файлами конфігурації, що є важливою частиною метода для Minecraft, оскільки саме через конфігурацію користувач може вносити зміни до роботи метода без зміни коду.

Метод складає з себе поєднання файлів, що працюють між собою, які є класами, що працюють з API.

3.3 Структура методу

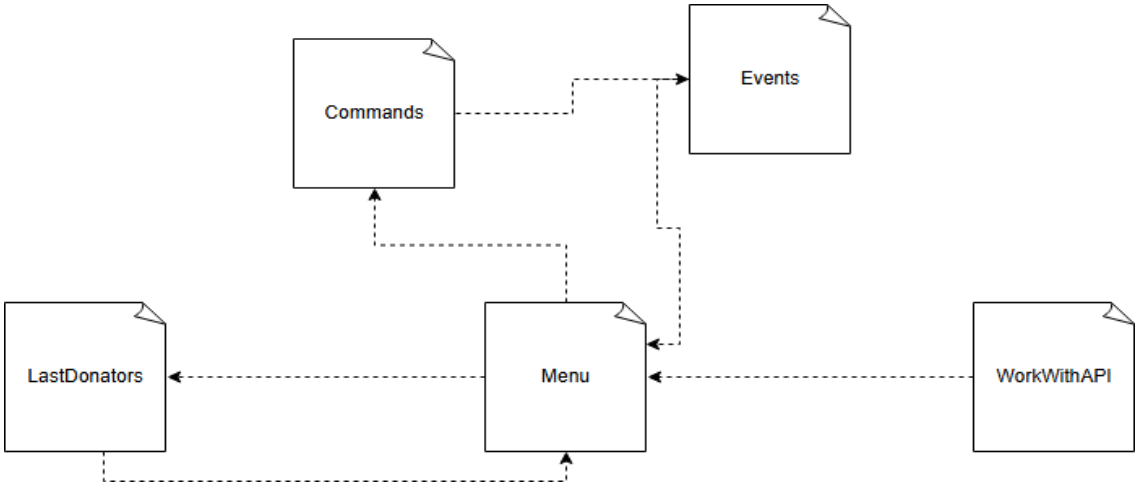


Рис. 3.2 Діаграма пакетів методу DonateEvents

Директорія lastDonators включає в собі два файла:

- DonationsInformation$DateTime.class
- DonationsInformation.class

DonationsInformation.class - це файл з класом, де зберігається інформація про донат, а саме:

- Сума
- Ім'я
- Повідомлення від донора.
- Об'єкт dateTime типу класу DateTime

Клас DonationsInformation$DateTime.class - вкладений клас до класу DonationsInformation і зберігає в собі такі дані:

- Рік
- Місяць
- День
- Година
- Хвилина
- Секунда

Клас має методи порівняння донатів, щоб виключити просрочені, виконані донати та конвертує їх в тип даних уууу- MM-dd HH:mm:ss.

Для збереження та виведення інформації, в цьому файлі використовується файл JsonObjects бібліотеки Gson(Json від Google).

В головній директорії методу знаходяться 4 файли:

- config.yml
- events.json
- plugin.yml
- usage.txt

config.yml: CFG(Configuration) файл з 3-ома полями, MonoApi,DyakaAPI та DonatelloAPI, в кожному з них є виділене поле де необхідно вставити API токен сервісу. Тобто, якщо користувач бажає налаштувати отримання донатів через сервіс Donatello - йому необхідно буде вставити в поле замість "change me" токен.

events.json: JSON файл з подіями для гри Майнкрафт. В даному файлі записані всі події, які були написані для методу. Кожна подія має такі поля:

- eventName - назву події.
- icon - зоображенння(іконка), яка буде відображись в грі.
- price {startSum:endSum} - діапазон сум, в якому може виникти подія, тобто подія не виникне, якщо сума донату складає 200 грн, а подія має діапазон startSum та endSum - між 40 грн до 100.
- commands {type:command:addition} - поле з командою, яке розділено на підполя, де визначається від чийого лиця буде виконана команда, сама команда та аргументи до команди, слід подивитись на прикладі однієї з подій.

commands": [

{

"type": "STRING_PLAYER_STRING",

"command": "execute as %s at @s run %s",

"addition": "summon wither ~ ~1 ~5 {CustomName:\u0027[{\"text\":

\"Помиральна машина\"}]\u0027,CustomNameVisible:1b,Glowing:1b}"

}

Команда буде виконана від лиця гравця - аргумент %s, в місці, де знаходится гравець - аргумент @s, run %s - аргумент яку команду виконувати, тут команда вкладена в поле addition і дана команда викликає ігрового боса - Візера на таких координатах ~(на тому місці, де знаходиться гравець по координаті x), ~1(на 1 блок вище координати гравця по y), ~5(на 5 блоків дальше від гравця по координаті z).

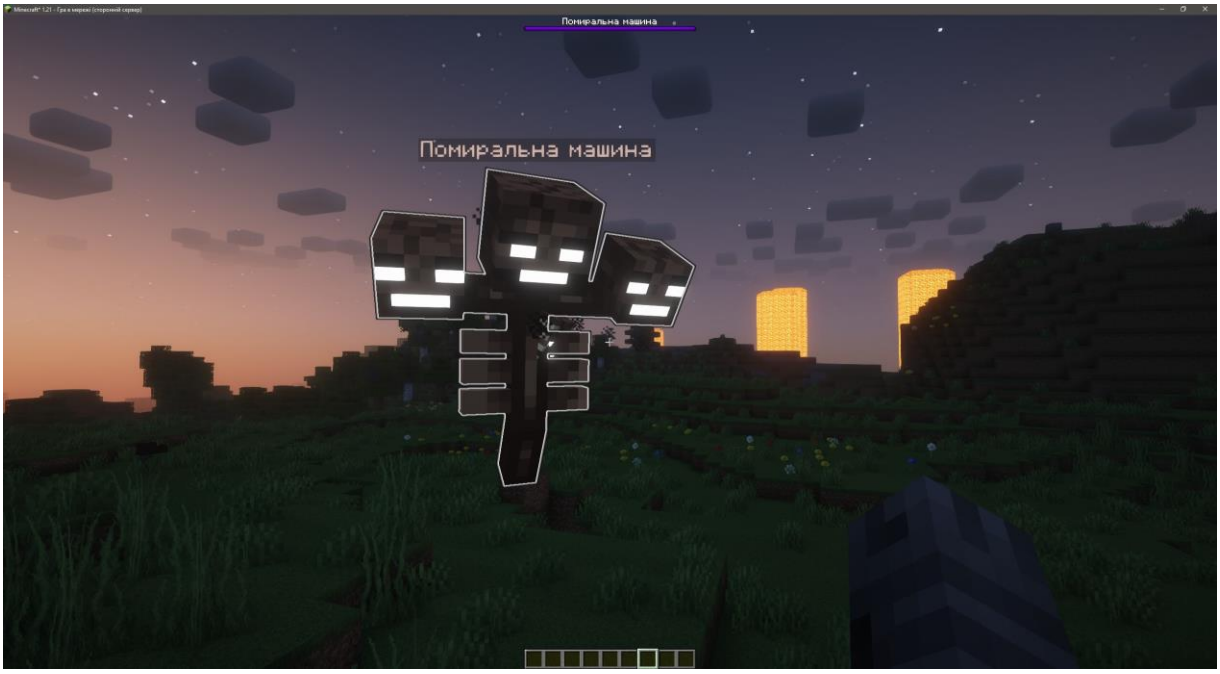


Рис 3.3 Як виглядає реалізація події безпосередньо в грі

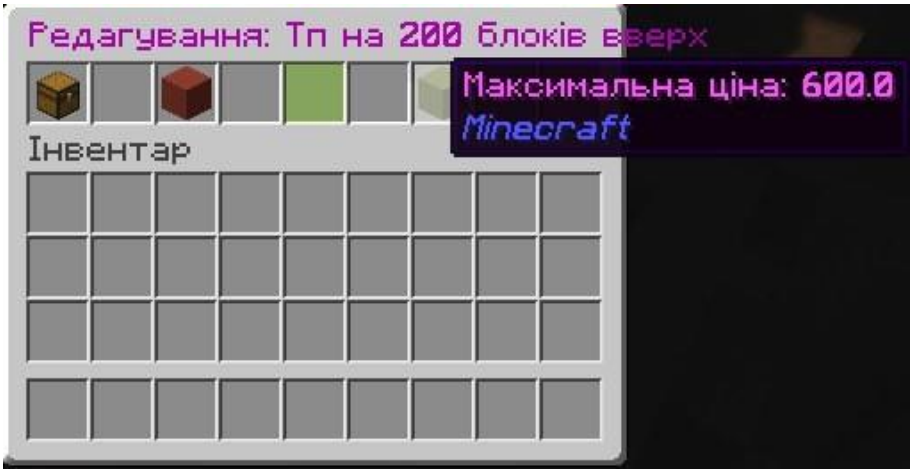


Рис 3.4 Також можна задавати їх діапазони (поля startSum та endSum)

Метод за рахунок технологій JSON в поєднанні з можливостями гри Minecraft дозволяє створювати динамічні ігрові сценарії, автоматизувати завдання та забезпечувати інтерактивність для гравців. Завдяки JSON можна легко передавати та обробляти дані, що відкриває широкі можливості для кастомізації команд і налаштувань. Це робить метод зручним інструментом для розробників серверів та стрімерам, дозволяючи їм швидко реалізовувати складні механіки гри.

plugin.yml та usage.txt

plugin.yml файл конфігурації, який містить перелік команд на сервер, автора та версію методу. Даний файл налічує команди:

Показ останніх донатів

Встановити ключ API. Даний метод встановлення ключа має таку перевагу, як безпека, тому що ключ зберігається в оперативній пам'яті серверу, де його витягнути майже неможливо. Але такий спосіб має недолік - ключ скидається після перезапуску серверу.

Відкрити меню налаштування.

Випадкова подія з задовільною сумою N.

Завантажити події з файлу.

Текстовий файл usage - Інструкція користування методом, з переліком команд та їх поясненням, корисно для розробників серверів, які будуть використовувати метод на своїх серверсах.

Директорія WorkWithApi включає в собі 4 файла:

DonatelloAPI.class

DyakaAPI.class

Ім'я: MonobankAPI.class

CustomMenu.class

Виходячи з назви файлів становиться зрозуміло, що - це файли класів, для роботи з API сервісів. Для отримання зв'язку з API використовується вид з'єднання REST API, через HTTP запити.

DyakaAPI.class:

Клас DonatelloAPI

Поля MAX_DONATIONS та plugin типу даних DonateEvents для взаємодії API Donatello з методом.

Поля MAX_DONATIONS та plugin типу даних DonateEvents для взаємодії API Donatello з методом.

Конструктор plugin типу даних DonateEvents для взаємодії API з класом. Зміні типу HttpClient та HttpRequest для створення запиту. Директорія Menu включає в собі 1 файл:

Файл містить клас CustomMenu. Даний клас відповідає за налаштування методу через інвертар

Клас створює зміні типу DonationEvent, Inventory, EventsArrays, після цього створює конструктор і викликає метод типу void InitializeMenu. Метод InitializeMenu і додає перевірку, яка в кінці показує, чи вимкнуті події або вімкнуті.

Далі йде обробник подій onMenuClick, який перевіряє хто та який блок в інвентарі натиснув, якщо це адміністратор натиснув на червоний блок - події будуть вимкнути, якщо зелений - ввімкнути.

Наступним йде метод openEditMenu. Він відповідає за редагування події в вікні редагування подій через інвентар, та реалізовує логіку встановлення максимальної ціни, що передається в поле endSum та мінімальну ціну startSum, відключення відкритої в інвентарі події або повернутись до загального списку подій .

Метод onMenuEditClick доповнює минулий метод, надаючи в консолі можливість вести суму для мінімальної та максимальної ціни події, тим самим задати діапазон при якому подія може статись.

Останній метод onPlayerJoin виводить гравцю інформацію про введення API токена через команду /SetApiKey або змінивши файл config.yml.

Рис 3.5 Виведення інформації в чат гри

Директорія Commands включає в собі 5 файлів:

- GetLastDonationsCommand.class
- OpenSettingsMenu.class
- ReLoadEvents.class
- SetApiKeyCommand.class
- StartRandEvent.class

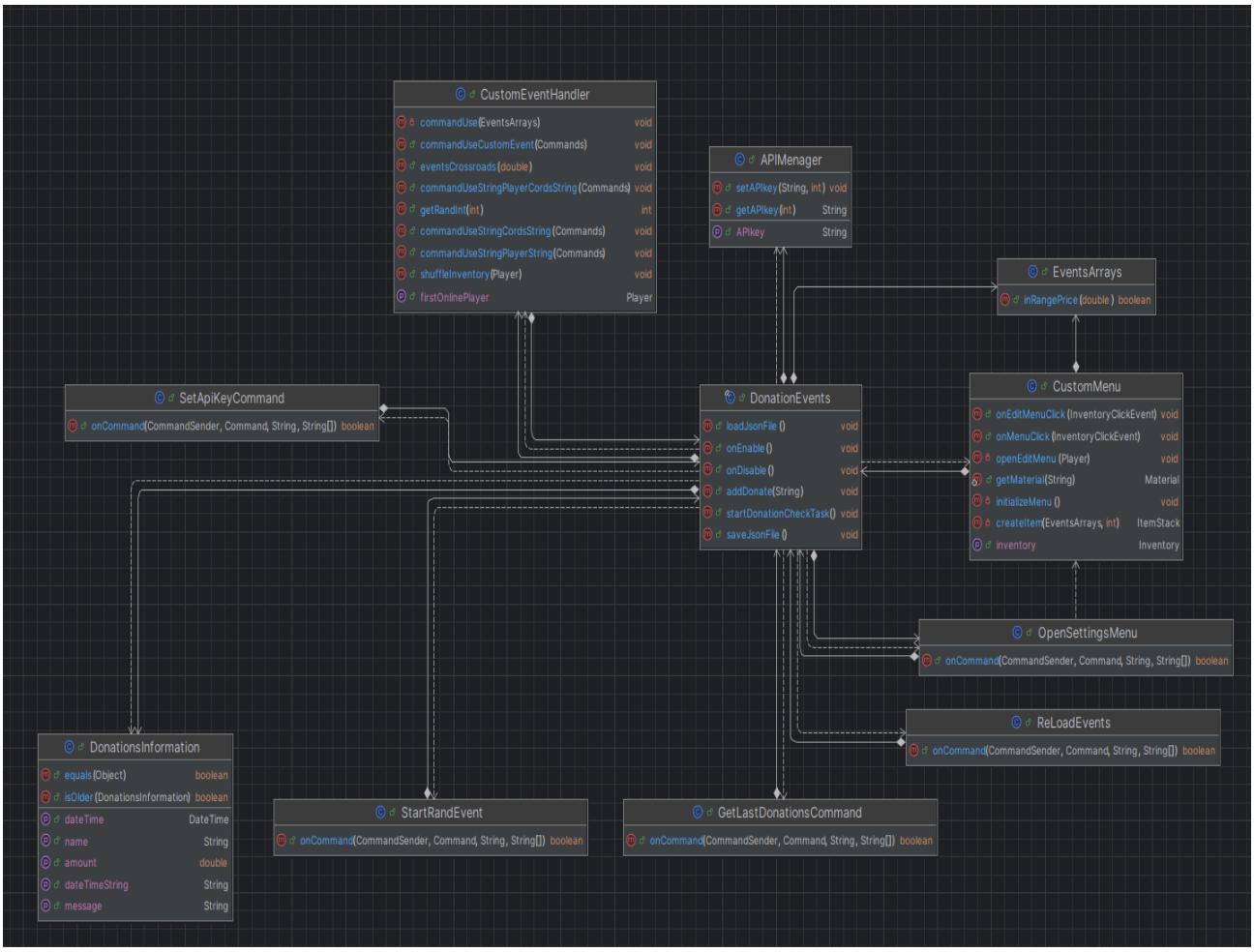


Рис. 3.5 Діаграма класів методу «DonateEvents»

На діаграмі класів представлено структуру методу, який складається з кількох основних компонентів, що відповідають за виконання різних функцій у системі. Центральну роль відіграє клас CustomEventHandler, який реалізує обробку подій та взаємодіє з іншими компонентами. Він містить методи для виконання команд, роботи з гравцями, масивами подій та управління інвентарем. Цей клас безпосередньо зв’язується з іншими модулями, такими як APIManager, DonationEvents та CustomMenu.

Клас APIManager забезпечує управління API-ключами, дозволяючи їх встановлювати та отримувати. Ця функціональність є критично важливою для інтеграції з зовнішніми сервісами або забезпечення автентифікації.

DonationEvents відповідає за обробку подій, пов’язаних із донатами. Він містить методи для завантаження і збереження JSON-файлів, активації й деактивації функціоналу, додавання донатів та перевірки їхньої актуальності. Донати представлені через клас DonationsInformation, який містить дані про дату, суму, ім’я донора, повідомлення та функції для порівняння та перевірки актуальності записів.

CustomMenu дозволяє створювати й керувати кастомними інтерфейсами, такими як інвентарі. У ньому реалізовані функції для редагування меню, створення предметів та ініціалізації інтерфейсів. Цей клас інтегрується з іншими елементами системи для побудови інтерактивного користувацького досвіду.

Інші класи, такі як SetApiKeyCommand, StartRandEvent, GetLastDonationsCommand, OpenSettingsMenu та ReLoadEvents, забезпечують обробку команд, що виконуються користувачами. Ці команди можуть запускати події, відкривати налаштування, оновлювати події або працювати з даними про донати.

Уся система побудована на основі чітко визначених зв’язків між компонентами. Класи працюють разом для забезпечення функціональності, інтеграції з API, обробки подій і надання зручного інтерфейсу для користувачів та адміністраторів. Архітектура дозволяє ефективно додавати новий функціонал, зберігаючи при цьому модульність та зворотну сумісність.

GetLastDonations: створує клас GetLastDonationsCommand, що реалізовує інтерфейс класу CommandExecutor, де ініціалізуємо приватне поле типу DonateEvent. Далі йде конструктор GetLastDonationsCommand, який створює об'єкт plugin.

Далі йде обробка події OnCommand, що відповідає за реагування команди, вона приймає такі аргументи:

- sender: відправник команди (може бути гравець або консоль).
- command: інформація про команду, яка була викликана.
- label: ім'я команди.
- args: список аргументів, переданих із командою.

if (!sender.hasPermission("donation.view")) { sender.sendMessage("Ви не маєте прав для виконання цієї команди."); return false;

Даний відрізок коду перевіряє хто виконав команду - консоль чи гравець і в негативному випадку виводить повідомлення про відсутність прав.

String token = this.plugin.api.getAPIkey(2); if (token == null) {

this.plugin.getLogger().severe("Токен для Donatello API == null."); return false;

}

Наступний крок перевірки, який виводить в логи інформацію про відсутність токену Donatello API.

else

{ this.plugin.donatelloAPI.getJsonDonators(token).thenAccept((resp onse) -> {

if (response != null) { sender.sendMessage(response);

} else {

sender.sendMessage("Не вдалося отримати дані

донатів.");

}

Останній крок методу. getJsonDonators - метод із класу donatelloAPI, який відправляє асинхронний запит до Donatello API, використовуючи токен. Якщо відповідь не порожня, тобто не дорівнює null, відповідь надсилається викликачу, в

інакшому випадку надходить відповідь про помилку. Якщо всі перевірки пройдено та запит відправлено, метод повертає true, що означає успішне виконання команди.

OpenSettingsMenu: створюється аналогічний клас та конструктор з іменем OpenSettingsMenu. Виконується метод OnCommand який виконує перевірку:

if (sender instanceof Player player) {

CustomMenu menu = new CustomMenu(this.plugin); player.openInventory(menu.getInventory());

return true;

блок перевіряє чи відправник - гравець або консоль. Якщо команду виконує консоль або щось інше, що не є гравцем виводиться виконується цей блок:

else {

sender.sendMessage(String.valueOf(ChatColor.RED) + "Цю команду можна використовувати тільки гравцями!");

return false;

ReLoadEvents: Відповідний клас та конструктор з іменем ReLoadEvents.Виконується метод OnCommand:

public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args) {

this.plugin.loadJsonFile(); this.plugin.saveJsonFile();

sender.sendMessage(String.valueOf(ChatColor.GREEN) + "Івенти оновлено");

return true;

Цей метод завантажує Json файл і перезаписує його, оновивши події.

Метод OnCommand обробляє команду для встановлення API- ключа для різних платформ (Dyaka, Donatello, Mono). Спочатку йде перевірка чи є гравець, який викликав команду адміністратором, якщо ні - повідомляє про це і повертає нічого. Якщо умова виконується - наступним кроком буде перевірка правильності вказаних аргументів, якщо аргументів буде менше 2 - програма нічого не поверне. Також важливо, що одним аргументом має бути ID сервісу, до якого має бути встановлений токен, а другим - сам токен, після цього він буде збережений в пам'яті серверу.

StartRandEvent: Відповідний клас та конструктор з іменем StartRandEvent. Виконується метод OnCommand:

public boolean onCommand(CommandSender sender, Command cmd, String label, String[] args) {

if (args.length > 0)

{ this.plugin.customEventHandler.eventsCrossroads(Double.parse Double(args[0]));

sender.sendMessage("Викликано випадковий івент із сумою: " + Double.parseDouble(args[0]));} else

{ this.plugin.customEventHandler.eventsCrossroads((double)1.0F); sender.sendMessage("Викликано випадковий івент із

сумою: 1");

}

return true;

Перший блок - перевірка, чи передано більше 1 аргументу, якщо так - число конвертується в double і передається в метод eventsCrossroads класу з CustomEventsHandler і повідомляє гравця про виклик події, в випадку, якщо аргумент був переданий невірно - викликається подія по замовчуванню.

Директорія Events включає в собі 7 файлів:

- CustomEventHandler$1.class
- CustomEventHandler.class
- EventsArrays$Events$Commands.class
- EventsArrays$Events$Price.class
- EventsArrays$Events$Type.class
- EventsArrays$Events.class
- EventsArrays.class

CustomEventHandler.class: Створення методу eventsCrossroads, який приймає в якості аргумента суму донату типу double, яка обробляє суму донату, наступним йде створення списку доступних подій та зміна, які присвоється випадкове число в межах масиву, якщо розмір такого масиву буде менше 1 - консоль повідомить про

відсутність подій.

Далі програма визначить тип команди, їх 4:

- commandUseStringCordsString - для подій, які потребують координати.
- commandUseStringPlayerString - для подій, які зв'язані з гравцем.
- commandUseStringPlayerCordsString - Для подій, які зв'язані з координатами та гравцем.
- commandUseCustomEvent - Для нестандартних подій.

commandUseStringCordsString - метод, що видобуває координати гравця і динамічно підставляє їх через String.format() і викликає команду. commandUseStringPlayerString - відповідає за виконання команд, які включають ім'я гравця та додаткові параметри, викликає команду від імені гравця.

commandUseStringPlayerCordsString - метод, який добуває координати гравця та його ім'я.

commandUseCustomEvent - метод, який відповідає за виклик нестандартних(касмномних подій), на момент написання роботи існує єдина нестандартна подія - inventoryShake, вона перемішує предмети в інвентарі гравця.

CustomEventHandler$1.class: вкладений клас до класу CustomEventHandler, він містить в собі присвоєння enum певному числу, щоб він міг працювати з оператором case. В головному класі існує метод CommandUse, який містить в собі 4 case-сценарія, де кожен

відповідає за тип команди, які були оголошені в класі CustomEventHandler.

EventsArrays.class: Цей клас зручний для перевірки, чи підходить конкретна ціна до певної події. Клас має єдиний метод: public boolean inRangePrice(double priceOth) {

return this.events.price.startSum <= priceOth

} Він перевіряє, чи входить отримана сума до діапазону.

EventsArrays$Events.class: Вкладений клас до класу EventsArrays.class, містить два конструктора - події зі всіма її параметрами та конструктор тільки з

ціною. Перший конструктор приймає такі параметри, як:

- Ціна
- Ім'я
- Команда
- Картинка(іконка)

EventsArrays$Events$Type.class: вкладений клас. Містить в собі 1 конструктор, який приймає в якості аргументів типи подій:

- Нестандартні
- Координатні
- Гравці
- Гравця та координатні

EventsArrays$Events$Price: вкладений клас. Відповідає за поля startSum та endSum. Клас ініціалізує поля та включає в себе 2 гетери та 2 методи, що відповідають правильність встановлення діапазону цін для подій через інвентар, методи повернуть false, якщо початкова сума буде більше кінцевої.

EventsArrays$Events$Commands.class: вкладений клас. Описує окрему команду, яка може бути виконана в межах певної події. Він визначає тип команди, основний текст команди, а також додаткові дані для цієї команди. Має 1 метод, що використовує поля:

- Command
- Addition
- Type

3.3 Результати

Рис. 3.6 Зомбі на оцелоті - одна з подій.

Метод створює цікаві події, від жартівливих до неприємних, що створює унікальні моменти та досвід при грі з цим методом.

Таблиця 3.4

Порівняння глядачів та стримера

Глядачі	Стример
Підтримка українських донат-сервісів та банків	Підтримка українських донат-сервісів та банків
Взаємодія з стрімером в реальному часі	Можливість кастомізації
Просте використання	Просте налаштування
Варіативність подій	Аналіз дій гравців
Імерсивність та зацікавленість	Імерсивність та зацікавленість

Метод окрім цього надає можливість створювати свої події, оскільки метод відкритий для всіх - кожен, хто бажає може створити свої події і створювати унікальні сценарії під час гри стрімера в гру Minecraft, тим самим збільшуючи інтерактивність між глядачем та стрімером.

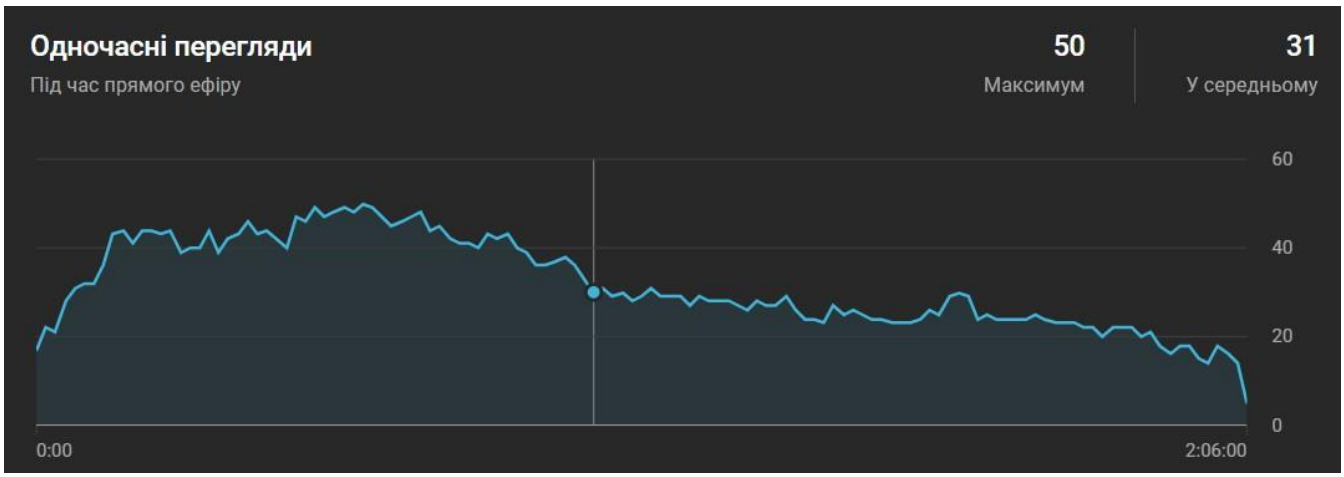


Рис. 3.7 Перегляди до застосування DonateEvents

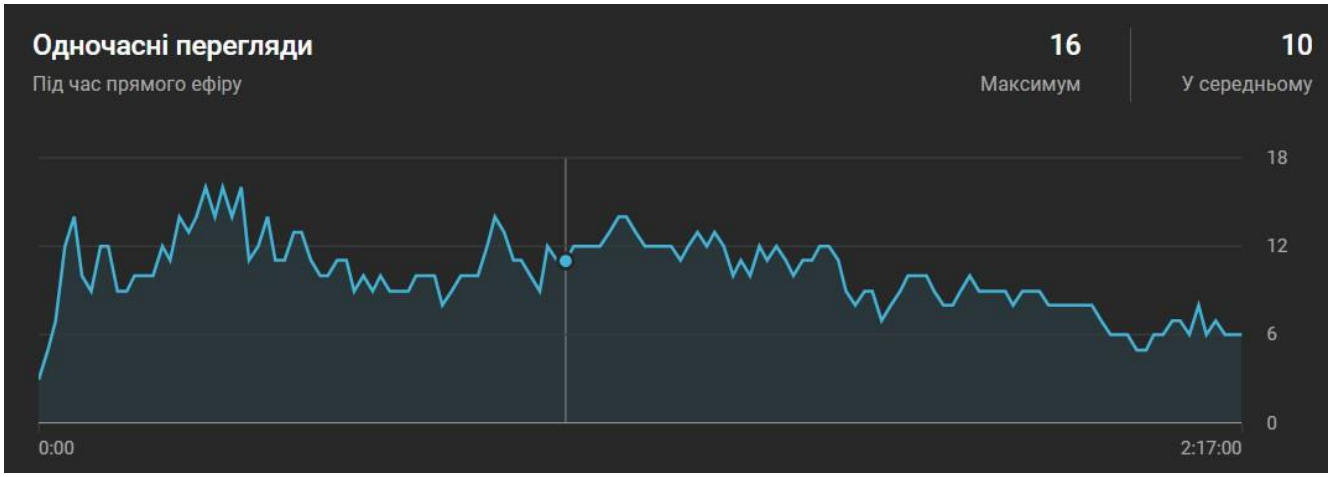


Рис. 3.8 Перегляди після застосування DonateEvents

Завдяки інтуїтивно зрозумілому інтерфейсу та можливостям адаптації, стрімер може легко налаштувати події під свої потреби. Метод підтримує локалізацію, що дозволяє використовувати українську мову для відображення повідомлень та команд. Українська спільнота стрімерів Minecraft зростає, і такі інструменти можуть стати важливим кроком для підтримки цієї тенденції. Інтерактивні методи:

- Підвищують конкурентоспроможність контенту
- Популяризують українську мову
- Стимулюють донати для підтримки контенту

ВИСНОВКИ

1. Проведено аналіз і оцінку існуючих методів щоб отримати повне уявлення про різноманітні методи та способи, щоб отримати повне представлення про інтерактивні донати.
2. Застосовано теорії та методики передачі та отримання даних. Проведені порівняння та оцінки ефективності методів.
3. Розроблено методи інтегрованих донатів, використовуючи ресурси гри та мережевих технологій.
4. Була проведена оцінка перспективності методів для українського стрімінгу та які позитивні результати показують методи.
5. Було ретельно підібрано технології та методи, які відповідають сучасним потребам інтерактивного стрімінгу. Для створення інтерактивних донатів були обрані надійні мережеві інструменти, платформи для розробки методів та ефективні алгоритми для синхронізації даних у реальному часі. Також були враховані вимоги до простоти налаштування і використання методу для стрімерів.
6. Запропоновано напрямки подальшого вдосконалення інтерактивних рішень для стрімінгу. Розглядаються можливості інтеграції додаткових функцій, таких як аналітика глядацької активності, адаптивні сценарії подій та підвищення безпеки обробки даних.

   Результати дослідження апробовано та опубліковано у наступних тезах доповіді на конференціях:

   1. Яскевич В.О., Посенко Д.Р. Розробка методів та засобів управління ігровими подіями через інтеграцію з донат-сервісами//Студентський науковий пошук 2024 – Київ: Київський столичний університет імені Бориса Грінченка, 2024. – Подано до друку.

   1. Яскевич В.О., Посенко Д.Р. Розробка інтерактивного модуля для minecraft з підтримкою взаємодії гравців через чат прямих трансляцій// Студентський науковий пошук- 2024 – Київ: Київський столичний університет імені Бориса Грінченка, 2024. – Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Designing Interactivity into Game Play. *University of Silicon Valley*. URL: https://usv.edu/blog/designing-interactivity-into-game-play/

2. Video Game History - Timeline & Facts. *HISTORY*. URL: https://www.history.com/topics/inventions/history-of-video-games .The history of interactive games and walls - Neo Xperiences. Neo Xperiences. URL: https://neoxperiences.com/en/2024/04/10/the-history-of-interactive- games-and-walls/

3. A Brief History of Online Games - The Strong National Museum of Play. The Strong National Museum of Play. URL: https://www.museumofplay.org/blog/ a-brief-history-of-online-games/

4. Modi N. Mobile games that made history. Medium. URL: https://medium.com/@nimishakmodi/mobile-games-that-made-history-599251366caf

5. https://www.theguardian.com/technology/2017/mar/01/nintendo-switch-console-review - Nintendo Switch загальні відомості.

6. https://www.researchgate.net/ publication/378435611_A_review_of_virtual_reality_technology - A review of virtual reality technology February 2024 Applied and Computational Engineering 38(1):1-6 DOI:10.54254/2755-2721/38/20230521

7. A Review on Augmented Reality Technology June 2018International Journal of Emerging Research in Management and Technology 7(1):22 DOI:10.23956/ ijermt.v7i1.19 -https://www.researchgate.net/ publication/326051772_A_Review_on_Augmented_Reality_Technology

8. The Editors of Encyclopaedia Britannica. Java | Definition & Facts | Britannica. Encyclopedia Britannica. URL: https://www.britannica.com/technology/Java-computer-programming-language

9. Interactivity: A review of the concept and a framework for analysis https://www.researchgate.net/ publication/240753324_Interactivity_A_review_of_the_concept_and_a_frame w ork_for_analysis

10. «Дяка» - Українська система подяк - https://diaka.ua/

11. Використовуйте API та виводьте на своєму сайті чи додатку статистику по подякам. https://diaka.ua/for-webmasters

12. Донателло / Donatello. *Donatello - підтримка українського контенту*. URL: https://donatello.to/

13. TipeeeStream is the best & cheapest way to collect tips as a streamer and display alerts https://www.tipeeestream.com/

14. The TipeeeStream API provides access to read data on TipeeeStream for streamers. https://api.tipeeestream.com/api-doc/

15. Streamlabs Desktop - Free | Free Streaming Software / https://streamlabs.com/

16. Java Platform Evolution - Dev.java. Dev.java: The Destination for Java Developers. URL: https://dev.java/evolution/

17. https://www.andrews-cooper.com/tech-talks/ac-teardowns-inside-the- nintendo-switch-joy-con/ Inside the Nintendo Switch Joy-Con | Engineering Teardown

18. What is Live Streaming Technology and How Does it Work?. *Dacast*. URL: https://www.dacast.com/blog/what-is-live-streaming/

19. Twitch.tv | About / https://www.twitch.tv/p/about/ Twitch is where millions of people come together live every day to chat, interact, and make their own entertainment together.

20. List of Interactive Movies & Series on Netflix / https://www.whats-on-netflix.com/library/interactive-titles-on-netflix/

21. https://en.wikipedia.org/wiki/Black_Mirror / Black mirror interactive film

22. Simplilearn. An Introduction to Methods in Java with Examples | Simplilearn. *Simplilearn.com*. URL: https://www.simplilearn.com/tutorials/java-tutorial/methods-in-java

23. What is an API? - Application Programming Interface Explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/api/

24. What is an API? (Application Programming Interface) | MuleSoft. *MuleSoft*. URL: https://www.mulesoft.com/api/what-is-an-api

27. Streetcatproject.org. Streetcatproject.org. URL: https://streetcatproject.org/

26. Ring Fit Adventure™ for Nintendo Switch™ – Official Site / https://

ringfitadventure.nintendo.com/

28. Open Broadcaster Software | OBS / https://obsproject.com/ OBS (Open Broadcaster Software) is free and open source software for video recording and live streaming.

29. Crowd Control - The Interactive Way / https://crowdcontrol.live/

30. What Is an API? Definition and FAQ on How to Work with APIs Read more on https://tech-stack.com/blog/what-is-an-api/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Методи та засоби управління ігровими подіями через інтеграцію з донат-сервісами»

Виконав: студент групи ПДМ-62 Посенко Данило Романович
Керівник: Яскевич В.О., к.т.н, доцент кафедри ІПЗ

Київ - 2025

МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: інтерактивність управління ігровими подіями та підвищення залученості користувачів за рахунок використання методів з донат-сервісами, що дозволяють автоматизувати взаємодію гравців із контентом гри.

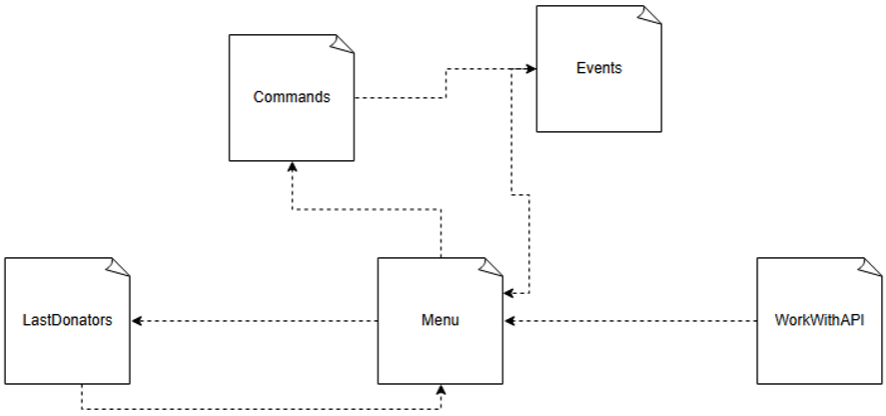
Об'єкт дослідження: Процес управління ігровими подіями в онлайн-іграх, що залучають користувачів до донат-систем.

Предмет дослідження: Засоби та технології інтеграції донат-сервісів з ігровими платформами для ефективного управління подіями та взаємодії з користувачами.

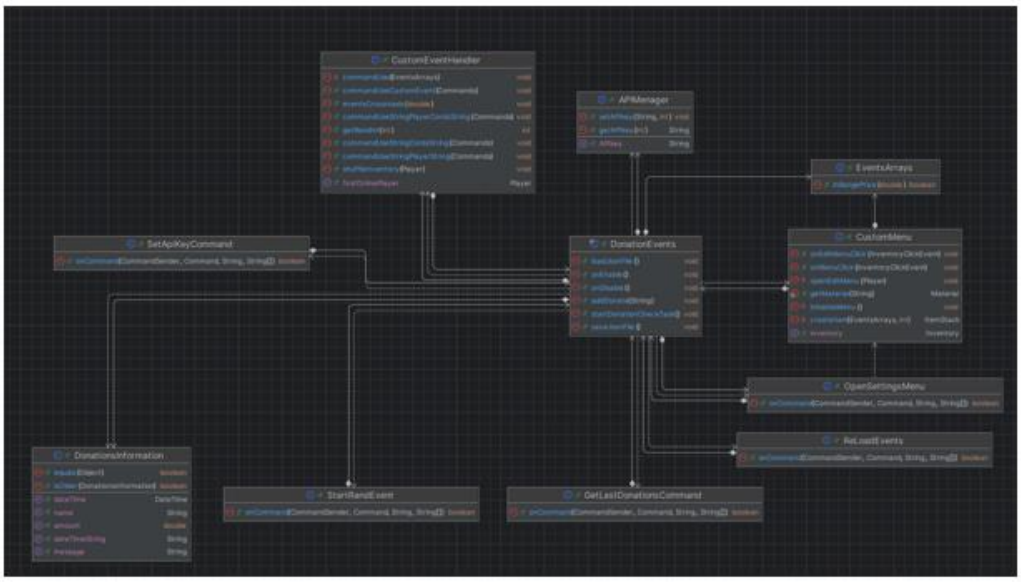
Порівняння методу «DonateEvents» з аналогами

Методи	Легкість налаштування	Можливість кастомізації	Реакція в реальному часі	Можливість донату за допомогою українських банків та донат сервісів	Інтеграція з різними платформами	Взаємодія зі стрімерами	Аналіз дій користувачів
Donatelvents	Простий інтерфейс із покрокомими інстукціями для налаштування.	Дозволяє змінювати візуальний стиль донатів (колір, текст, анімації).	Відображення донатів на стрімі без затримок.	Інтеграція з Monobank, ПриватБанк, LiqPay.	Підтримка YouTube, Twich.	Включає додаткові функції взаємодії.	Звіти з деталями про суми та активність донатерів.
TwichCraft	Інтуїтивно зрозумілий процес налаштування через інтерфейс Twich.	Обмежений вибір шаблонів, мінімальні налаштування.	Середня швидкість відображення донатів із затримкою до 5 секунд.	Не підтримується, працює лише з міжнародними платформами на кшталт PayPal.	Обмежена лише Twich.	Проста взаємодія через прості повідомлення від донатерів.	Відсутній або обмежений базовими логами.
MC Interactive	Складна процедура налаштування для новачків.	Мінімальні опції, кастомізація переважно недоступна.	Реакція залежить від роботи сервера ,часто бувають затримки.	Відсутня, працює з криптовалютою та міжнародними сервісами.	Підтримка YouTube, Twich.	Донати безпосередньо інтегруються з ігровими подіями.	Лише базові логи, активності на сервері.
StreamingDrops	Проста процедура налаштування через зрозумілий вебінтерфейс.	Можливість змінювати лише базові елементи дизайну.	Висока швидкість реакції завдяки інтеграції API.	Не підтримується, доступні лише міжнародні сервіси.	Обмежена стріминговими платформами, YouTube, Twich.	Інтерактивні елементи обмежені простою подякою.	Відсутній інструмент аналітики.

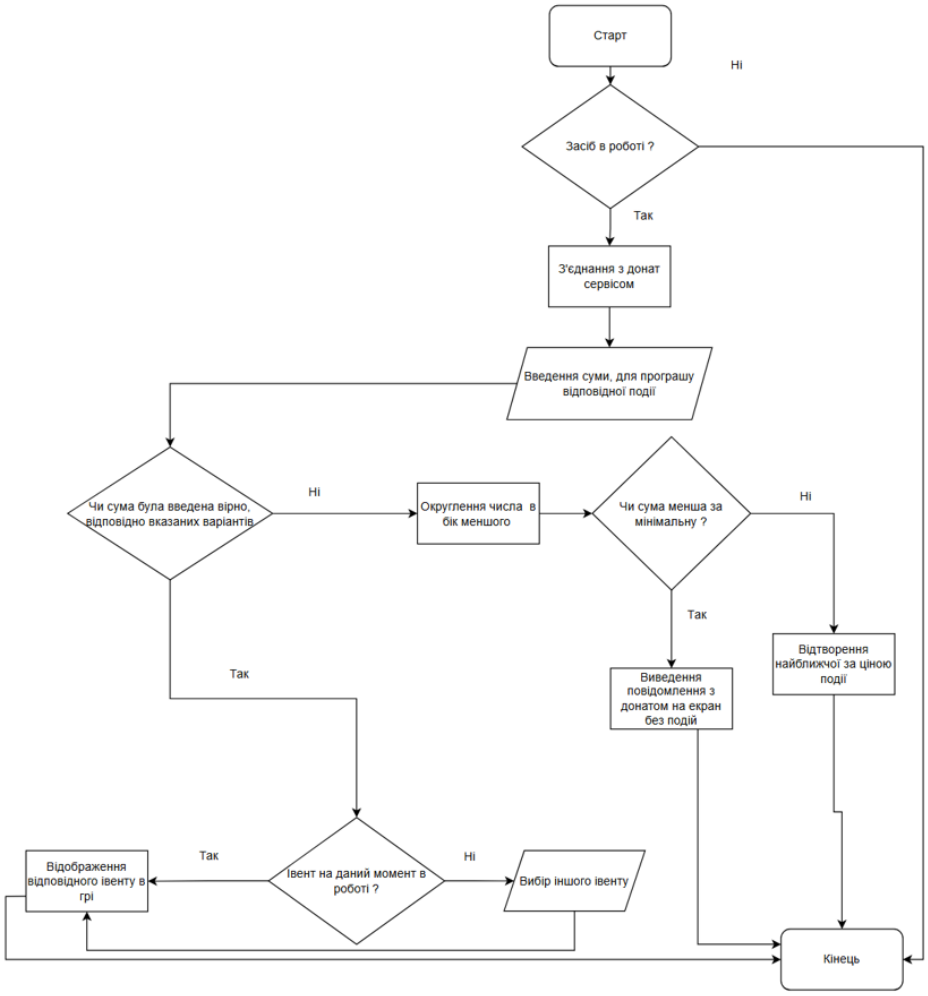
Діаграма пакетів



Діаграма класів



Блок схема алгоритму методу «Donatelvents»

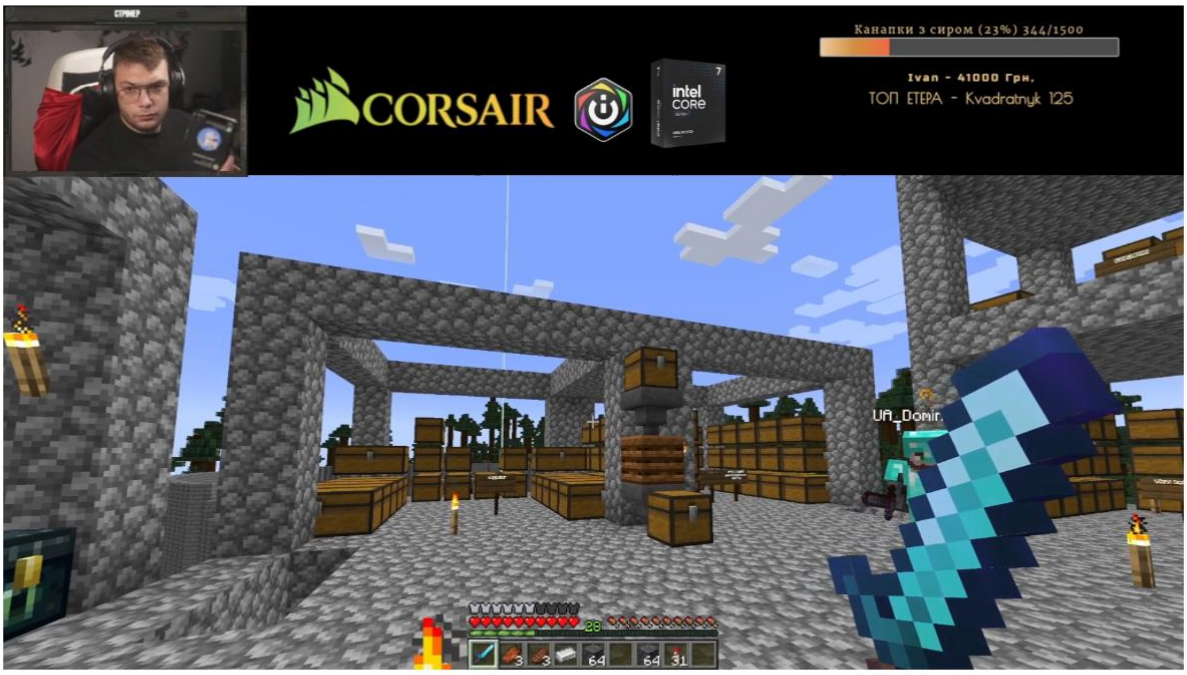


Результат роботи методу

Рисунок 1.1 Зміна цін на події в грі

Рисунок 1.2 Всі наявні івенти в методі «DonateIvents»

Результат роботи методу



ВИСНОВКИ

1. Проведено аналіз і оцінку існуючих методів щоб отримати повне уявлення про різноманітні методи та способи, щоб отримати повне представлення про інтерактивні донати.
2. Застосовано теорії та методики передачі та отримання даних. Проведені порівняння та оцінки ефективності методів.
3. Розроблено методи інтегрованих донатів, використовуючи ресурси гри та мережевих технологій.
4. Була проведена оцінка перспективності методів для українського стрімінгу та які позитивні результати показують методи.
5. Було ретельно підібрано технології та методи, які відповідають сучасним потребам інтерактивного стрімінгу. Для створення інтерактивних донатів були обрані надійні мережеві інструменти, платформи для розробки методів та ефективні алгоритми для синхронізації даних у реальному часі. Також були враховані вимоги до простоти налаштування і використання методу для стрімерів.
6. Запропоновано напрямки подальшого вдосконалення інтерактивних рішень для стрімінгу. Розглядаються можливості інтеграції додаткових функцій, таких як аналітика глядацької активності, адаптивні сценарії подій та підвищення безпеки обробки даних.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Статті:

Тези доповідей:

1. Яскевич В.О, Посенко Д.Р. РОЗРОБКА МЕТОДІВ ТА ЗАСОБІВ УПРАВЛІННЯ ІГРОВИМИ ПОДІЯМИ ЧЕРЕЗ ІНТЕГРАЦІЮ З ДОНАТ-СЕРВІСАМИ//Студентський науковий пошук-2024- Київ: Київський столичний університет імені Бориса Грінченка, 2024. – Подано до друку

2. Яскевич В.О, Посенко Д.Р. РОЗРОБКА ІНТЕРАКТИВНОГО МОДУЛЯ ДЛЯ MINECRAFT З ПІДТРИМКОЮ ВЗАЄМОДІЇ ГРАВЦІВ ЧЕРЕЗ ЧАТ ПРЯМИХ ТРАНСЛЯЦІЙ//Студентський науковий пошук-2024 - Київ: Київський столичний університет імені Бориса Грінченка, 2024. – Подано до друку

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
package org.dominator.donationEvents.commands;

import org.bukkit.command.Command;
import org.bukkit.command.CommandExecutor;
import org.bukkit.command.CommandSender;
import org.dominator.donationEvents.DonationEvents;

public class GetLastDonationsCommand implements
CommandExecutor {

   private final DonationEvents plugin;

   public GetLastDonationsCommand(DonationEvents plugin)
{
      this.plugin = plugin;
   }

   @Override
   public boolean onCommand(CommandSender sender,
Command command, String label, String[] args) {
      if (!sender.hasPermission("donationevents.view")) {
         sender.sendMessage("Ви не маєте прав для виконання
цієї команди.");
         return false;
      }

      String  token = plugin.api.getAPIkey(2);
      if (token == null) {
         plugin.getLogger().severe("Токен для Donatello API
== null.");
         return false;
      }


plugin.donatelloAPI.getJsonDonators(token).thenAccept(respo
nse -> {
         if (response != null) {
            sender.sendMessage(response);
         } else {
            sender.sendMessage("Не вдалося отримати дані
донатів.");
         }
      });

      return true;
   }
}

//=========================================
package org.dominator.donationEvents.commands;

import org.bukkit.ChatColor;
import org.bukkit.command.Command;
import org.bukkit.command.CommandExecutor;
import org.bukkit.command.CommandSender;
import org.bukkit.entity.Player;
import org.dominator.donationEvents.DonationEvents;
import org.dominator.donationEvents.menu.CustomMenu;

public class OpenSettingsMenu implements CommandExecutor
{
   private final DonationEvents plugin;

   public OpenSettingsMenu(DonationEvents plugin) {
      this.plugin = plugin;
   }

   @Override
   public boolean onCommand(CommandSender sender,
Command cmd, String label, String[] args) {

      if (sender instanceof Player) {
         Player player = (Player) sender;
         CustomMenu menu = new CustomMenu(plugin);
         player.openInventory(menu.getInventory());
         return true;
      }
      sender.sendMessage(ChatColor.RED + "Цю команду
можна використовувати тільки гравцями!");
      return false;
   }
}

//=========================================
package org.dominator.donationEvents.commands;

import org.bukkit.ChatColor;
import org.bukkit.command.Command;
import org.bukkit.command.CommandExecutor;
import org.bukkit.command.CommandSender;
import org.dominator.donationEvents.DonationEvents;

public class ReLoadEvents implements CommandExecutor {
   private final DonationEvents plugin;

   public ReLoadEvents(DonationEvents plugin) {
      this.plugin = plugin;
   }

   @Override
   public boolean onCommand(CommandSender sender,
Command cmd, String label, String[] args) {

      plugin.loadJsonFile();
      plugin.saveJsonFile();
      sender.sendMessage(ChatColor.GREEN + "Івенти
оновлено");
      return true;
   }
}

//=========================================
package org.dominator.donationEvents.commands;

import org.bukkit.command.Command;
import org.bukkit.command.CommandExecutor;
import org.bukkit.command.CommandSender;
import org.dominator.donationEvents.DonationEvents;

public class SetApiKeyCommand implements
CommandExecutor {
   private final DonationEvents plugin;

   public SetApiKeyCommand(DonationEvents plugin) {
      this.plugin = plugin;
   }

   @Override
   public boolean onCommand(CommandSender sender,
Command command, String label, String[] args) {
      if (!sender.hasPermission("setAPIKey.view")) {
         sender.sendMessage("Ви не маєте прав для виконання
```

```
цієї команди.");
            return false;
        }

        if(args.length < 2 ){
            sender.sendMessage("Встановіть ключ 1-Dyaka 2-
Donatello 3-Mono");
            return false;
        }

        plugin.api.setAPIkey(args[0], Integer.parseInt(args[1]));

        return true;
    }
}


//=========================================
package org.dominator.donationEvents.commands;

import org.bukkit.command.Command;
import org.bukkit.command.CommandExecutor;
import org.bukkit.command.CommandSender;
import org.dominator.donationEvents.DonationEvents;

public class StartRandEvent implements CommandExecutor {
    private final DonationEvents plugin;

    public StartRandEvent(DonationEvents plugin) {
        this.plugin = plugin;
    }

    @Override
    public boolean onCommand(CommandSender sender,
Command cmd, String label, String[] args) {
        if (args.length > 0) {

plugin.customEventHandler.eventsCrossroads(Double.parseDo
uble(args[0]));
            sender.sendMessage("Викликано випадковий івент із
сумою: " + Double.parseDouble(args[0]));
        }else {
            plugin.customEventHandler.eventsCrossroads(1);
            sender.sendMessage("Викликано випадковий івент із
сумою: " + 1);
        }

        return true;
    }
}


//=========================================
package org.dominator.donationEvents.events;

import org.bukkit.Bukkit;
import java.util.*;
import org.bukkit.entity.Player;
import org.dominator.donationEvents.DonationEvents;

public class CustomEventHandler {

    private final DonationEvents plugin;
    private final EventExecute execute;

    public CustomEventHandler(DonationEvents plugin){
        this.plugin = plugin;
        this.execute = new EventExecute(plugin);
    }

    public void eventsCrossroads(double amount) {
        plugin.getLogger().info("Обробка суми: " + amount);
        List<EventsArrays> newEvents = new ArrayList<>();
        for (EventsArrays oth : plugin.eventsArraysList){
            if (oth.inRangePrice(amount)){
                newEvents.add(oth);
            }
        }

        if (!newEvents.isEmpty()) {
            int randInt = getRandInt(newEvents.size());
            plugin.getLogger().info("Кількість доступних івентів:
" + newEvents.size());
            if (!DonationEvents.useTargetName) {
                for (Player play : Bukkit.getOnlinePlayers()){
                    commandUse(newEvents.get(randInt), play);
                }
            } else {
                Player play = Bukkit.getOnlinePlayers().stream()
                        .filter(player ->
player.getName().equalsIgnoreCase(DonationEvents.targetNa
me))
                        .findFirst()
                        .orElse(null);
                commandUse(newEvents.get(randInt), play);
            }
        } else {
            plugin.getLogger().info("Відсутні доступні івенти");
        }
    }

    private void commandUse(EventsArrays event, Player
player){
        for (EventsArrays.Events.Commands command :
event.events.commands) {
            switch (command.type){
                case STRING_CORDS_STRING ->
execute.commandUseStringCordsString(command, player);
                case STRING_PLAYER_STRING ->
execute.commandUseStringPlayerString(command, player);
                case STRING_PLAYER_CORDS_STRING ->
execute.commandUseStringPlayerCordsString(command,
player);
                case CUSTOM_EVENT ->
execute.commandUseCustomEvent(command, player);
            }
        }
    }


    public int getRandInt(int endInt){
        return new Random().nextInt(endInt);
    }



}


//=========================================
package org.dominator.donationEvents.events;

import org.bukkit.entity.Player;
import org.bukkit.inventory.ItemStack;
import org.dominator.donationEvents.DonationEvents;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
```

```
public class CustomEvents {
   private final DonationEvents plugin;

   public CustomEvents(DonationEvents plugin) {
      this.plugin = plugin;
   }

   public void shuffleInventory(Player player) {
      plugin.getLogger().info("Перемішую інвентар");

      ItemStack[] inventoryContents = player.getInventory().getContents();

      List<ItemStack> inventoryItems = new ArrayList<>(Arrays.asList(inventoryContents));

      Collections.shuffle(inventoryItems);

      player.getInventory().clear();

      for (int i = 0; i < inventoryItems.size(); i++) {
         player.getInventory().setItem(i, inventoryItems.get(i));
      }

      for (int i = inventoryItems.size(); i < inventoryContents.length; i++) {
         player.getInventory().setItem(i, null);
      }
   }

   public void manyRandDonations(float sum){
      plugin.getLogger().info("Перемішую інвентар");
      plugin.customEventHandler.eventsCrossroads(10);
//TODO: rand sum

   }
}



//=========================================
package org.dominator.donationEvents.events;

import org.bukkit.Bukkit;
import org.bukkit.Location;
import org.bukkit.entity.Player;
import org.dominator.donationEvents.DonationEvents;

public class EventExecute {
   private final DonationEvents plugin;
   public final CustomEvents evList;

   public EventExecute(DonationEvents plugin) {
      this.plugin = plugin;
      this.evList = new CustomEvents(plugin);
   }

   public void commandUseStringCordsString(EventsArrays.Events.Commands command, Player play){
      Location location = play.getLocation();
      int x = location.getBlockX(), y = location.getBlockY(), z = location.getBlockZ();

      String newCommand = String.format(command.command, x, y, z, command.addition);

      plugin.getLogger().info("Викликав команду " + newCommand);

plugin.getServer().dispatchCommand(plugin.getServer().getConsoleSender(), newCommand);
   }

   public void commandUseStringPlayerString(EventsArrays.Events.Commands command, Player play){
      String newCommand = String.format(command.command, play.getName(), command.addition);
      plugin.getLogger().info("Викликав команду " + newCommand);

plugin.getServer().dispatchCommand(plugin.getServer().getConsoleSender(), newCommand);
   }

   public void commandUseStringPlayerCordsString(EventsArrays.Events.Commands command, Player play){
      Location location = play.getLocation();
      int x = location.getBlockX(), y = location.getBlockY(), z = location.getBlockZ();

      String newCommand = String.format(command.command, play, x, y, z, command.addition);
      plugin.getLogger().info("Викликав команду " + newCommand);

plugin.getServer().dispatchCommand(plugin.getServer().getConsoleSender(), newCommand);
   }

   public void commandUseCustomEvent(EventsArrays.Events.Commands command, Player play){
      // TODO: тут тре буде ще щось
      switch (command.command){
         case "inventoryShake" -> evList.shuffleInventory(play);
         case "manyEvents" -> {}
      }
   }
}



//=========================================
package org.dominator.donationEvents.events;

import org.apache.commons.lang3.tuple.Pair;

import java.util.ArrayList;
import java.util.List;

public class EventsArrays {
   public Events events;

   public boolean inRangePrice(double priceOth){
      return events.price.startSum <= priceOth && priceOth <= events.price.endSum;
   }

   public static  class Events {
      public String eventName;
      public String icon;
      public Price price;
      public List<Commands> commands = new ArrayList<>();

      public  Events(List<Commands> commands, Price price,
```

```
String eventName, String icon) {
        this.commands = commands;
        this.price = price;
        this.eventName = eventName;
        this.icon = icon;
    }

    public Events(Price price) {
        this.price = price;
    }

    public static class Commands {
        public Type type;
        public String command;
        public String addition;

        public Commands(String command, String addition,
Type type) {
            this.command = command;
            this.addition = addition;
            this.type = type;
        }

        public Commands(String command) {
            this(command,                                    "",
Type.STRING_CORDS_STRING);
        }
    }


    public static class Price {
        double startSum;
        double endSum;

        public Price(double startSum, double endSum) {
            this.startSum = startSum;
            this.endSum = endSum;
        }

        public boolean changeMaxSum(double endSum){
            if (this.startSum > endSum){
                return false;
            }
            this.endSum = endSum;
            return true;
        }

        public boolean changeMinSum(double startSum){
            if (this.endSum < startSum){
                return false;
            }
            this.startSum = startSum;
            return true;
        }

        public double getStartSum() {
            return startSum;
        }

        public double getEndSum() {
            return endSum;
        }
    }

    public static enum Type{
        STRING_CORDS_STRING,
        STRING_PLAYER_STRING,
        STRING_PLAYER_CORDS_STRING,
        CUSTOM_EVENT
    }
}
```

```
//========================================
package org.dominator.donationEvents.lastDonators;

import com.google.gson.JsonObject;
import org.json.JSONObject;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.format.DateTimeFormatter;
import java.util.Objects;

public class DonationsInformation {
    private final Double amount;
    private final String name;
    private final String message;
    private final DateTime dateTime;

    public DonationsInformation(double amount, String name,
String message, String dateTime) {
        this.amount = amount;
        this.name = name;
        this.message = message;
        this.dateTime = new DateTime(dateTime);
    }

    public DonationsInformation(JsonObject json) {
        this(
                json.get("amount").getAsDouble(),
                json.get("clientName").getAsString(),
                json.has("message")                              ?
json.get("message").getAsString() : "",
                json.get("createdAt").getAsString()
        );
    }

    public double getAmount() {
        return amount;
    }

    public String getName() {
        return name;
    }

    public String getMessage() {
        return message;
    }

    public DateTime getDateTime() {
        return dateTime;
    }

    public String getDateTimeString(){
        return dateTime.toString();
    }

    public boolean isOlder(DonationsInformation other){
        return this.dateTime.isOlder(other.dateTime);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        DonationsInformation that = (DonationsInformation) o;
```

```
        return Objects.equals(amount, that.amount) && Objects.equals(name, that.name) && Objects.equals(message, that.message) && Objects.equals(dateTime, that.dateTime);
    }

    @Override
    public int hashCode() {
        return Objects.hash(amount, name, message, dateTime);
    }

    public static class DateTime {
        int year;
        int month;
        int day;

        int hour;
        int minute;
        int seconds;

        public DateTime(int year, int month, int day, int hour, int minute, int seconds) {
            setDayTime(year, month, day, hour, minute, seconds);
        }

        public DateTime(String dateTimeString){
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
            LocalDateTime dateTime = LocalDateTime.parse(dateTimeString, formatter);
            setDayTime(dateTime.getYear(), dateTime.getMonthValue(), dateTime.getDayOfMonth(), dateTime.getHour(), dateTime.getMinute(), dateTime.getSecond());
        }

        public void setDayTime(int year, int month, int day, int hour, int minute, int seconds) {
            this.year = year;
            this.month = month;
            this.day = day;
            this.hour = hour;
            this.minute = minute;
            this.seconds = seconds;
        }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            DateTime dateTime = (DateTime) o;
            return year == dateTime.year && month == dateTime.month && day == dateTime.day && hour == dateTime.hour && minute == dateTime.minute && seconds == dateTime.seconds;
        }

        @Override
        public int hashCode() {
            return Objects.hash(year, month, day, hour, minute, seconds);
        }

        public boolean isOlder(DateTime other){
            if (this.year > other.year) {
                return true;
            } else if (this.year < other.year) {
                return false;
            }

            if (this.month > other.month) {
                return true;
            } else if (this.month < other.month) {
                return false;
            }

            if (this.day > other.day) {
                return true;
            } else if (this.day < other.day) {
                return false;
            }

            if (this.hour > other.hour) {
                return true;
            } else if (this.hour < other.hour) {
                return false;
            }

            if (this.minute > other.minute) {
                return true;
            } else if (this.minute < other.minute) {
                return false;
            }

            return this.seconds > other.seconds;
        }

        public boolean isEqual(DateTime other){
            return this.toString().equals(other.toString());
        }

        public String toString() {
            return String.format("%04d-%02d-%02d %02d:%02d:%02d", year, month, day, hour, minute, seconds);
        }
    }
}
```

```
//=========================================
package org.dominator.donationEvents.menu;

import org.bukkit.Bukkit;
import org.bukkit.ChatColor;
import org.bukkit.Material;
import org.bukkit.entity.Player;
import org.bukkit.event.EventHandler;
import org.bukkit.event.HandlerList;
import org.bukkit.event.Listener;
import org.bukkit.event.inventory.InventoryClickEvent;
import org.bukkit.event.player.AsyncPlayerChatEvent;
import org.bukkit.event.player.PlayerJoinEvent;
import org.bukkit.inventory.Inventory;
import org.bukkit.inventory.ItemStack;
import org.bukkit.inventory.meta.ItemMeta;
import org.dominator.donationEvents.DonationEvents;
import org.dominator.donationEvents.events.EventsArrays;

public class CustomMenu implements Listener {
    private Inventory inventory;
    private DonationEvents plugin;
    private EventsArrays editingEvent;

    public CustomMenu(DonationEvents plugin) {
        this.plugin = plugin;
        initializeMenu();
    }

    private void initializeMenu() {
        inventory = null;
```

```
        this.inventory = Bukkit.createInventory(null, 54,
ChatColor.DARK_PURPLE + "Редагування івентів");
        int count = 1;
        for (EventsArrays event : plugin.eventsArraysList){
            ItemStack item = createItem(event, count);
            inventory.setItem(count - 1, item);
            count++;
        }

        ItemStack iventsState;
        if (DonationEvents.acceptEvents){
            iventsState = new
ItemStack(Material.LIME_STAINED_GLASS_PANE);
            ItemMeta metaIventsState = iventsState.getItemMeta();
            metaIventsState.setDisplayName(ChatColor.GREEN +
"Вимкнути / Івенти ввімкнено");
            iventsState.setItemMeta(metaIventsState);
        } else {
            iventsState = new
ItemStack(Material.RED_STAINED_GLASS_PANE);
            ItemMeta metaIventsState = iventsState.getItemMeta();

metaIventsState.setDisplayName(ChatColor.DARK_RED +
"Ввімкнути / Івенти вимкнено");
            iventsState.setItemMeta(metaIventsState);
        }
        inventory.setItem(49, new ItemStack(iventsState));
    }


    private ItemStack createItem(EventsArrays event, int count) {
        String name = event.events.eventName;
        String materialName = event.events.icon;

        Material material =
Material.matchMaterial(materialName);

        if (material == null) {
            material = Material.BARRIER;
        }

        ItemStack item = new ItemStack(material, count);
        ItemMeta meta = item.getItemMeta();
        if (meta != null) {

            meta.setDisplayName(name);
            item.setItemMeta(meta);
        }

        return item;
    }



    public Inventory getInventory() {
        return inventory;
    }

    public static Material getMaterial(String name) {
        Material material = Material.matchMaterial(name);
        if (material != null) return material;
        return Material.BARRIER;
    }

    @EventHandler
    public void onMenuClick(InventoryClickEvent event) {
        if
(event.getView().getTitle().equals(ChatColor.DARK_PURPLE
+ "Редагування івентів")) {
            event.setCancelled(true);

            ItemStack clickedItem = event.getCurrentItem();
            if (clickedItem != null && clickedItem.hasItemMeta())
{
                String itemName =
clickedItem.getItemMeta().getDisplayName();
                if (itemName.equals(ChatColor.DARK_RED +
"Ввімкнути / Івенти вимкнено") ||
itemName.equals(ChatColor.GREEN + "Вимкнути / Івенти
ввімкнено")){
                    DonationEvents.acceptEvents =
!DonationEvents.acceptEvents;
                    initializeMenu();
                    event.getWhoClicked().openInventory(inventory);
                    plugin.updateStartTime();
                    return;
                }

                for (EventsArrays eventArray :
plugin.eventsArraysList) {
                    if
(eventArray.events.eventName.equals(itemName)) {
                        editingEvent = eventArray;
                        break;
                    }
                }

                if (editingEvent != null) {
                    openEditMenu((Player) event.getWhoClicked());
                }
            }
        }
    }

    private void openEditMenu(Player player) {
        Inventory editMenu = Bukkit.createInventory(null, 9,
ChatColor.DARK_PURPLE + "Редагування: " +
editingEvent.events.eventName);

        //ТУТ ТІПА ЗМІНА МАКСИМАЛЬНОЇ ЦІНИ
        ItemStack editButtonMax = new
ItemStack(Material.LIME_TERRACOTTA);
        ItemMeta metaMax = editButtonMax.getItemMeta();
        metaMax.setDisplayName(ChatColor.LIGHT_PURPLE +
"Максимальна ціна: " +
editingEvent.events.price.getEndSum());
        editButtonMax.setItemMeta(metaMax);
        editMenu.setItem(6, editButtonMax);

        //ТУТ ТІПА ЗМІНА МІНІМАЛЬНОЇ ЦІНИ
        ItemStack editButtonMin = new
ItemStack(Material.RED_TERRACOTTA);
        ItemMeta metaMin = editButtonMin.getItemMeta();
        metaMin.setDisplayName(ChatColor.LIGHT_PURPLE +
"Мінімальна ціна: " +
editingEvent.events.price.getStartSum());
        editButtonMin.setItemMeta(metaMin);
        editMenu.setItem(2, editButtonMin);

        //ТУТ ТІПА ВИМКНЕННЯ ІВЕНТУ
        ItemStack iventState;
        if (editingEvent.events.price.getStartSum() ==
editingEvent.events.price.getEndSum() &&
editingEvent.events.price.getStartSum() == 0f){
            iventState = new
ItemStack(Material.RED_STAINED_GLASS_PANE);
            ItemMeta metaIventState = iventState.getItemMeta();

metaIventState.setDisplayName(ChatColor.LIGHT_PURPLE
+ "Вимкнути / Івент вже вимкнено");
```

```
            iventState.setItemMeta(metaIventState);
        } else {
            iventState = new
ItemStack(Material.LIME_STAINED_GLASS_PANE);
            ItemMeta metaIventState = iventState.getItemMeta();

metaIventState.setDisplayName(ChatColor.LIGHT_PURPLE
+ "Вимкнути / Івент ввімкнено");
            iventState.setItemMeta(metaIventState);
        }
        editMenu.setItem(4, iventState);


        //ТУТ ТІПА ПОВЕРНУТИСЯ В ГОЛОВНЕ МЕНЮ
        ItemStack back = new ItemStack(Material.CHEST);
        ItemMeta backMeta = back.getItemMeta();
        backMeta.setDisplayName(ChatColor.LIGHT_PURPLE +
"Повернутися до івентів");
        back.setItemMeta(backMeta);
        editMenu.setItem(0, back);

        initializeMenu();
        player.openInventory(editMenu);
    }

    @EventHandler
    public void onEditMenuClick(InventoryClickEvent event) {
        if
(event.getView().getTitle().startsWith(ChatColor.DARK_PUR
PLE + "Редагування: ")) {
            event.setCancelled(true);

            //ТУТ ТІПА ПОВЕРНЕННЯ В ГОЛОВНЕ МЕНЮ
            if (event.getCurrentItem() != null &&
event.getCurrentItem().getType() == Material.CHEST){
                ((Player)
event.getWhoClicked()).openInventory(inventory);
            }

            //ТУТ ТІПА ВИМКНЕННЯ ІВЕНТУ
            if (event.getCurrentItem() != null &&
event.getCurrentItem().getType()
Material.LIME_STAINED_GLASS_PANE){
                editingEvent.events.price.changeMinSum(0);
                editingEvent.events.price.changeMaxSum(0);
                plugin.saveJsonFile();
                plugin.loadJsonFile();
                openEditMenu((Player) event.getWhoClicked());
            }

            //ТУТ ТІПА ЗМІНА МАКСИМАЛЬНОЇ ЦІНИ
            if (event.getCurrentItem() != null &&
event.getCurrentItem().getType()
Material.LIME_TERRACOTTA) {
                Player player = (Player) event.getWhoClicked();

                player.sendMessage(ChatColor.YELLOW +
"Введіть максимальну ціну:");
                player.closeInventory();


plugin.getServer().getPluginManager().registerEvents(new
Listener() {
                    @EventHandler
                    public void
onChatMessage(AsyncPlayerChatEvent chatEvent) {
                        if (chatEvent.getPlayer().equals(player)) {
                            chatEvent.setCancelled(true);

                            try {
                                double enteredValue =
Double.parseDouble(chatEvent.getMessage());
                                boolean success =
editingEvent.events.price.changeMaxSum(enteredValue);

                                if (success) {
                                    player.sendMessage(ChatColor.GREEN
+ "Максимальну ціну змінено на: " + enteredValue);
                                } else {
                                    player.sendMessage(ChatColor.RED +
"Введене значення не є дійсним.");
                                }

                            } catch (NumberFormatException e) {
                                player.sendMessage(ChatColor.RED +
"Будь ласка, введіть дійсне число.");
                            }

                            plugin.saveJsonFile();
                            plugin.loadJsonFile();

                            Bukkit.getScheduler().runTask(plugin, () ->
openEditMenu(player));

                            HandlerList.unregisterAll(this);
                        }
                    }
                }, plugin);
            }

            //ТУТ ТІПА ЗМІНА МІНІМАЛЬНОЇ ЦІНИ
            if (event.getCurrentItem() != null &&
event.getCurrentItem().getType()
Material.RED_TERRACOTTA) {
                Player player = (Player) event.getWhoClicked();

                player.sendMessage(ChatColor.YELLOW +
"Введіть мінімальну ціну:");
                player.closeInventory();


plugin.getServer().getPluginManager().registerEvents(new
Listener() {
                    @EventHandler
                    public void
onChatMessage(AsyncPlayerChatEvent chatEvent) {
                        if (chatEvent.getPlayer().equals(player)) {
                            chatEvent.setCancelled(true);

                            try {
                                double enteredValue =
Double.parseDouble(chatEvent.getMessage());

                                boolean success =
editingEvent.events.price.changeMinSum(enteredValue);

                                if (success) {
                                    player.sendMessage(ChatColor.GREEN
+ "Мінімальну ціну змінено на: " + enteredValue);
                                } else {
                                    player.sendMessage(ChatColor.RED +
"Введене значення не є дійсним.");
                                }

                            } catch (NumberFormatException e) {
                                player.sendMessage(ChatColor.RED +
"Будь ласка, введіть дійсне число.");
                            }

                            plugin.saveJsonFile();
```

```
                plugin.loadJsonFile();

                Bukkit.getScheduler().runTask(plugin, () -> openEditMenu(player));

                HandlerList.unregisterAll(this);
              }
            }
          }, plugin);
        }
      }
    }

    //TODO: move to other class
    @EventHandler
    public void onPlayerJoin(PlayerJoinEvent event) {
      if (plugin.api.donatelloAPIkey.equals("change me") || plugin.api.donatelloAPIkey.equals("")){
        event.getPlayer().sendMessage("§l§8Встанови АПІ ключ §3Donatello §4/setAPIKey §ствійAPI§4 2§r");
        event.getPlayer().sendMessage("§7Або якщо ти мамкін §l§8хацкер§7 заміни його в конфігу: §9../plugins/DonationEvents/config.yml§7 заміни §4change me§7 на свій АПІ ключ");
      }
    }
}
```

```
//=========================================
package org.dominator.donationEvents.workWithApi.APIusageType;

import org.dominator.donationEvents.DonationEvents;

import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.util.concurrent.CompletableFuture;

public class DonatelloAPI {

    private final DonationEvents plugin;

    private static final int MAX_DONATIONS = 5;

    public DonatelloAPI(DonationEvents plugin) {
        this.plugin = plugin;
    }

    /**
     *
     * @param token
     * @return
     */

    public CompletableFuture<String> getJsonDonators(String token) {
        String baseUrl = "https://donatello.to/api/v1/donates";
        int size = 10;

        String url = String.format("%s?page=%d&size=%d", baseUrl, 0, size);

        HttpClient client = HttpClient.newBuilder()
                .followRedirects(HttpClient.Redirect.NORMAL)
                .build();

        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create(url))
                .header("X-Token", token)
                .header("User-Agent", "Mozilla/5.0")
                .header("Content-Type", "application/json")
                .GET()
                .build();

        return client.sendAsync(request, HttpResponse.BodyHandlers.ofString())
                .thenApply(response -> {
                    if (response.statusCode() == 401) {
                        plugin.getLogger().severe("Помилка авторизації. Неправильний токен.");
                        return null;
                    }

                    plugin.getLogger().info("Донати успішно отримані");
                    return response.body();
                })
                .handle((responseBody, throwable) -> {
                    if (throwable != null) {
                        plugin.getLogger().severe("Помилка при отриманні донатів: " + throwable.getMessage());
                    }
                    return responseBody;
                });
    }

}
```

```
//=========================================
package org.dominator.donationEvents.workWithApi;

public class APIManager {
    public String dyakaAPIkey;
    public String donatelloAPIkey;
    public String monoAPIkey;;

    public APIManager(){
        dyakaAPIkey = "";
        donatelloAPIkey = "";
        monoAPIkey = "";
    }

    public APIManager(String APIkey, int keyType) {
        setAPIkey(APIkey, keyType);
    }

    public void setAPIkey(String APIkey, int keyType) {
        switch (keyType){
            case 1: dyakaAPIkey = APIkey; break;
            case 2: donatelloAPIkey = APIkey; break;
            case 3: monoAPIkey = APIkey; break;
            default: break;
        }
    }

    public String getAPIkey(int keyType) {
        return switch (keyType) {
            case 1 -> dyakaAPIkey;
            case 2 -> donatelloAPIkey;
            case 3 -> monoAPIkey;
            default -> "ПІТН";
        };
    }
```

```
    public String getAPIkey() {
        return "ПІНХ";
    }
}



//=========================================
package org.dominator.donationEvents;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonArray;
import com.google.gson.JsonObject;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.scheduler.BukkitRunnable;
import org.bukkit.Bukkit;
import org.dominator.donationEvents.commands.*;
import
org.dominator.donationEvents.events.CustomEventHandler;
import org.dominator.donationEvents.events.EventsArrays;
import
org.dominator.donationEvents.lastDonators.DonationsInformat
ion;
import org.dominator.donationEvents.menu.CustomMenu;
import
org.dominator.donationEvents.workWithApi.APIManager;
import
org.dominator.donationEvents.workWithApi.APIusageType.D
onatelloAPI;
import
org.dominator.donationEvents.workWithApi.APIusageType.D
yakaAPI;
import
org.dominator.donationEvents.workWithApi.APIusageType.M
onobankAPI;
import com.google.gson.reflect.TypeToken;
import java.io.File;
import java.io.FileReader;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;


public final class DonationEvents extends JavaPlugin {
    public APIManager api = new APIManager();
    public DyakaAPI dyakaAPI;
    public DonatelloAPI donatelloAPI;
    public MonobankAPI monobankAPI;

    private File jsonFile;
    private Gson gson;

    public OpenSettingsMenu openSettingsMenu = new
OpenSettingsMenu(this);

    public CustomEventHandler customEventHandler;
    public List<DonationsInformation> donationsInformation =
new ArrayList<>(10);
    public List<EventsArrays> eventsArraysList;

    public static boolean acceptEvents = true;
    public static boolean useTargetName = false;
    public static DonationsInformation.DateTime startTime;
    public static String targetName = "";



    @Override
    public void onEnable() {
        getLogger().info("DonationEvents плагін увімкнено!");

        updateStartTime(); //Set zero point

        gson = new GsonBuilder().setPrettyPrinting().create();

        saveResource("usage.txt", true);

        saveResource("events.json", false);
        jsonFile = new File(getDataFolder(), "events.json");
        if (!jsonFile.exists()) {
            try {
                jsonFile.getParentFile().mkdirs();
                jsonFile.createNewFile();
                eventsArraysList = new ArrayList<>();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            loadJsonFile();
        }

        saveDefaultConfig();

        targetName                                      =
getConfig().getString("settings.targetName"); //Use event only
for this player
        useTargetName                                   =
getConfig().getBoolean("settings.useTargetName"); //Use event
only for one player


api.setAPIkey(getConfig().getString("settings.dyakaAPI"), 1);

api.setAPIkey(getConfig().getString("settings.donatelloAPI"),
2);
        api.setAPIkey(getConfig().getString("settings.monoAPI"),
3);

        dyakaAPI = new DyakaAPI(this);
        donatelloAPI = new DonatelloAPI(this);
        monobankAPI = new MonobankAPI(this);

        customEventHandler = new CustomEventHandler(this);

        startDonationCheckTask();

        this.getCommand("setAPIkey").setExecutor(new
SetApiKeyCommand(this));
        this.getCommand("lastDonations").setExecutor(new
GetLastDonationsCommand(this));
        this.getCommand("openSettingsMenu").setExecutor(new
OpenSettingsMenu(this));
        this.getCommand("randEventSum").setExecutor(new
StartRandEvent(this));
        this.getCommand("reloadEvents").setExecutor(new
ReLoadEvents(this));
        getServer().getPluginManager().registerEvents(new
CustomMenu(this), this);
    }

    public void updateStartTime(){
        startTime                       =                   new
DonationsInformation.DateTime(LocalDateTime.now().plusHo
```

```
urs(1)./*TODO: hotfixed change local time for -1 h from Kyiv*/format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));

        DonationEvents.this.getLogger().info("Час запуску серверу: " + startTime.toString());
    }

    public void startDonationCheckTask() {
        new BukkitRunnable() {
            @Override
            public void run() {
                if (acceptEvents) {

donatelloAPI.getJsonDonators(api.getAPIkey(2)).thenAccept(response -> {
                        if (response != null) {
                            addDonate(response);
                        }
                    }).exceptionally(throwable -> {
                        DonationEvents.this.getLogger().info("Виникла помилка при отриманні донатів: " + throwable.getMessage());
                        return null;
                    });
                } else {

DonationEvents.this.getLogger().info("Отримання донатів ВИМКНЕННО");
                }
            }
        }.runTaskTimer(this, 0L, 20L * 20);

    }


    public void addDonate(String response){
        Gson gson = new Gson();
        JsonObject json = gson.fromJson(response, JsonObject.class);
        JsonArray contentArray = json.getAsJsonArray("content");

        for (int i = 0; i < contentArray.size(); i++) {
            JsonObject donationJson = contentArray.get(i).getAsJsonObject();
            if (!donationsInformation.contains(new DonationsInformation(donationJson)) &&
                    new DonationsInformation.DateTime(donationJson.get("createdAt").getAsString()).isOlder(startTime)
                    ) {
                if (donationsInformation.size() == 10 ) {
                    donationsInformation.removeFirst();
                }
                donationsInformation.add(new DonationsInformation(donationJson));
                DonationEvents.this.getLogger().info("Додано донат: Ім'я: `"+ donationsInformation.getLast().getName() + "` Сума: `" + donationsInformation.getLast().getAmount() + "` Повідовлення: `" + donationsInformation.getLast().getMessage() + "` Час створення: `" + donationsInformation.getLast().getDateTimeString() + "`");

                String createdAt = donationJson.get("createdAt").getAsString();
                DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
                LocalDateTime createdAtTime = LocalDateTime.parse(createdAt, formatter);
                LocalDateTime currentTime = LocalDateTime.now().plusHours(1); //TODO: hotfixed change local time for -1 h from Kyiv

                // Вираховуємо різницю в мілісекундах між поточним часом і часом створення донату
                long delay = ChronoUnit.MILLIS.between(currentTime, createdAtTime.plusSeconds(20));
                this.getLogger().info("Затримка до обробки" + delay / 1000.0 + " с.");
                if (delay < 0 || delay > 20000) {
                    delay = new Random().nextInt(19999);
                }

                this.getLogger().info("Затримка після обробки " + delay / 1000.0 + " с.");

                Bukkit.getScheduler().runTaskLater(this, () -> {
                    this.getLogger().info("Викликано івенти");

customEventHandler.eventsCrossroads(donationJson.get("amount").getAsDouble());
                }, delay / 50);
            }
        }
    }

    public void saveJsonFile() {
        try (FileWriter writer = new FileWriter(jsonFile)) {
            JsonObject jsonObject = new JsonObject();
            JsonArray jsonArray = gson.toJsonTree(eventsArraysList).getAsJsonArray();

            jsonObject.add("eventsArrays", jsonArray);
            gson.toJson(jsonObject, writer);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void loadJsonFile() {
        try (FileReader reader = new FileReader(jsonFile)) {
            JsonObject jsonObject = gson.fromJson(reader, JsonObject.class);
            eventsArraysList = gson.fromJson(jsonObject.getAsJsonArray("eventsArrays"), new TypeToken<List<EventsArrays>>() {}.getType());
            if (eventsArraysList == null) {
                eventsArraysList = new ArrayList<>();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onDisable() {
        saveJsonFile();
        getLogger().info("DonationEvents плагін вимкнено.");
    }

}
```