Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ МОДЕЛЮВАННЯ ІНДУСТРІАЛЬНИХ ПРОЦЕСІВ
SOFTWARE IMPLEMENTATION OF THE INDUSTRIAL PROCESS MODELING SYSTEM

Виконав(ла): студент(ка) 4 курсу, групи КНТ-122
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення
БАРАБАШ А.В.
(ПРІЗВИЩЕ та ініціали)
Керівник ГОФМАН Є.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент ДЬЯЧУК Т.С.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н., проф.
Сергій СУББОТІН
“____” _____2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

БАРАБАША Антона Володимировича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація системи моделювання індустріальних процесів.
Software Implementation of the Industrial Process Modeling System

керівник проєкту (роботи) к.т.н., доцент, ГОФМАН Євгеній Олександрович ,
(науковий ступінь, вчене звання ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування архітектури та вибір інструментальних засобів розробки. 3. Програмна реалізація та алгоритми функціонування системи. 4. Експлуатація та експериментальне дослідження програмної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Прийняв виконане завдання
1-4 Основна частина	ГОФМАН Є. О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 20 ” квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5 тиждень	Розділ 3
5	Тестування та експериментальне дослідження	6 тиждень	Розділ 4
	програми.		
6	Оформлення пояснювальної записки та документів	7 тиждень	Додатки
	до неї. Нормоконтроль та рецензування		
7	Захист роботи.	8 тиждень	

Студент(ка) _____ Антон БАРАБАШ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Євгеній ГОФМАН
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 111 с., 2 табл., 30 рис., 3 дод., 13 джерел.

МОДЕЛЮВАННЯ ПРОЦЕСІВ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, MINECRAFT FORGE, JAVA, PARCHMENT MAPPINGS, ЕНЕРГЕТИЧНІ МЕРЕЖІ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

Об’єкт дослідження – процес інженерії програмного забезпечення для моделювання та візуалізації індустріальних і виробничих систем у воксельних середовищах.

Предмет дослідження – методи, інструменти та архітектурні рішення для побудови модульних систем генерації, транспортування енергії та обробки ресурсів на базі ігрового рушія.

Мета роботи – розширення функціоналу та підвищення ступеню достовірності моделювання технологічних процесів в середовищі Minecraft.

Матеріали, методи та технічні засоби: використано принципи об’єктно-орієнтованого програмування та патерни проєктування клієнт-серверних систем. Програмну реалізацію виконано мовою програмування Java (JDK 21.0.8) у середовищі розробки IntelliJ IDEA. Як базову платформу використано рушій гри Minecraft (версія 1.20.1) та відкритий інтерфейс програмування додатків Forge API (MDK 47.4.10). Для забезпечення зрозумілої структури коду, деобфускації та стандартизації назв методів рушія використано Parchment mappings. Збирання проєкту налаштовано за допомогою системи автоматизації Gradle. Для реалізації енергомереж використано підсистему Forge Energy (FE).

Результати. Розроблено програмну систему моделювання індустріальних процесів (модифікацію «IndustrialCore»), яка реалізує функціонал генерації, збереження, транспортування енергії та

автоматизованої переробки ресурсів. Створено архітектурні абстракції для вузлів енергомережі (кабелі з ієрархією ізоляції), генераторів (тепловий, вітровий), машин-споживачів (дробарка, електропіч, металоформувальна машина) та інструментів. Налаштовано оптимізовану синхронізацію стану сутностей (BlockEntity) між клієнтом та сервером, реалізовано графічні інтерфейси користувача (GUI) за допомогою дата-орієнтованого підходу (Data-Driven) та систему користувацьких рецептів.

Висновки. Досягнуто мети роботи: створено надійну програмну архітектуру, здатну обробляти та візуалізувати індустріальні процеси в реальному часі без перевантаження сервера. Розроблена система демонструє ефективне використання об'єктно-орієнтованого підходу, чіткий архітектурний розподіл між клієнтською і серверною логікою (Separation of Concerns) та високий рівень модульності.

Галузь використання – освітні проєкти для інтерактивної візуалізації інженерних і фізичних процесів, ігрова індустрія (розробка модифікацій), а також використання як бази для вивчення принципів побудови клієнт-серверної архітектури.

Економічна ефективність. Використання готового воксельного середовища (Minecraft) як платформи для моделювання суттєво знижує витрати та час на розробку спеціалізованих 3D-рушіїв. Відкрита архітектура проєкту (Forge) дозволяє швидке масштабування системи та інтеграцію нових виробничих ланцюгів без значних фінансових вкладень.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 111 pages, 2 tables, 30 figures, 3 appendixes, 13 sources.

PROCESS MODELING, CLIENT-SERVER ARCHITECTURE, MINECRAFT FORGE, JAVA, PARCHMENT MAPPINGS, ENERGY NETWORKS, OBJECT-ORIENTED PROGRAMMING.

Object of study – the software engineering process for modeling and visualization of industrial and manufacturing systems in voxel environments.

Subject of study – methods, tools, and architectural solutions for building modular systems for energy generation, transportation, and resource processing based on a game engine.

Purpose of the work – Expansion of functionality and increasing the degree of reliability in modeling technological processes within the Minecraft environment.

Materials, methods, and technical tools: the principles of object-oriented programming and design patterns of client-server systems were utilized. The software implementation was executed in the Java programming language (JDK 21.0.8) within the IntelliJ IDEA integrated development environment. The Minecraft game engine (version 1.20.1) and the open application programming interface Forge API (MDK 47.4.10) were used as the base platform. Parchment mappings were utilized to ensure a clear code structure, deobfuscation, and standardization of the engine's method names. Project building was configured using the Gradle automation system. The Forge Energy (FE) subsystem was used to implement energy networks.

Results. A software system for modeling industrial processes (the "IndustrialCore" modification) was developed, which implements the functionality of energy generation, storage, transportation, and automated resource processing. Architectural abstractions were created for energy network nodes (cables with an

insulation hierarchy), generators (thermal, wind), consumer machines (macerator, electric furnace, metal former), and tools. Optimized synchronization of entity states (BlockEntity) between the client and server was configured, graphical user interfaces (GUI) were implemented using a Data-Driven approach, alongside a custom recipe system.

Conclusions. The purpose of the work was achieved: a reliable software architecture capable of processing and visualizing industrial processes in real time without overloading the server was created. The developed system demonstrates the effective use of the object-oriented approach, a clear architectural division between client and server logic (Separation of Concerns), and a high level of modularity.

Field of application – educational projects for interactive visualization of engineering and physical processes, the gaming industry (mod development), as well as serving as a base for studying the principles of building client-server architectures.

Economic efficiency. Using a ready-made voxel environment (Minecraft) as a modeling platform significantly reduces the cost and time required to develop specialized 3D engines. The open architecture of the project (Forge) allows for rapid system scaling and the integration of new production chains without significant financial investments.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AABB – Axis-Aligned Bounding Box;

API – Application Programming Interface;

FE – Forge Energy;

GUI – Graphical User Interface;

IDE – Integrated Development Environment;

JDK – Java Development Kit;

JSON – JavaScript Object Notation;

JVM – Java Virtual Machine;

MDK – Mod Development Kit;

NBT – Named Binary Tag;

OOP – Object-Oriented Programming;

UI – User Interface;

UV – UV mapping (текстурні координати).

ВСТУП

Стрімкий розвиток інтерактивних технологій та концепцій гейміфікації відкриває нові підходи до моделювання складних інженерних та логістичних систем. У сучасному освітньому та дослідницькому середовищі особливої ваги набуває наочна візуалізація процесів: від створення замкнутих циклів виробництва до відстеження розподілу енергії в мережі. Попри існування потужних професійних CAD-систем та симуляторів, чимало користувачів та дослідників стикаються з високим порогом входження та браком інтерактивності. Актуальність розробки зумовлена необхідністю створення доступного, гнучкого та модульного програмного середовища на базі воксельної платформи, де проєктування, тестування та візуалізація індустріальних процесів відбуваються в режимі реального часу з можливістю безпосередньої взаємодії з кожним елементом системи.

Глобальні тенденції в галузі розробки програмного забезпечення підтверджують, що використання ігрових рушіїв як платформ для прототипування є високоефективним рішенням. Дослідження впровадження інтерактивних середовищ наголошують: візуальний зворотний зв’язок та можливість експериментувати з фізичними та логічними законами в ізольованому просторі (sandbox) суттєво підвищують розуміння складних концепцій. Хоча використання Minecraft для створення логічних схем не є новим явищем, існує потреба в розробці оптимізованих архітектурних рішень, що дозволяють симулювати багатокрокові промислові процеси без критичного падіння продуктивності сервера. Існуючі масштабні модифікації часто виявляються перевантаженими або мають закриту архітектуру, що ускладнює їх вивчення та модифікацію.

Проблема дослідження полягає в необхідності створення оптимізованої, швидкодійної та легко масштабованої програмної архітектури, яка забезпечує коректну передачу енергії, обробку ресурсів та миттєву синхронізацію станів між серверною та клієнтською частинами гри. Огляд

технічних рішень показує, що реалізація подібних систем вимагає суворого дотримання принципів об'єктно-орієнтованого програмування та патернів проєктування для уникнення розсинхронізації даних під час обчислення ігрових тіків. Офіційна документація інтерфейсу програмування додатків Forge API вказує на високу перспективність використання системи "Можливостей" (Capabilities) [1]-[2] та дата-орієнтованого підходу (Data-Driven) [3] для створення надійних клієнт-серверних модифікацій.

Об’єктом дослідження виступають процеси програмного моделювання та візуалізації індустріальних і виробничих систем у воксельних середовищах.

Предметом дослідження є методи, інструменти та архітектурні рішення для побудови модульних клієнт-серверних систем генерації, транспортування енергії та автоматизованої обробки ресурсів на базі ігрового рушія.

Мета роботи полягає у розробці спеціалізованої програмної системи (модифікації) в середовищі Minecraft для розширення базового функціоналу та забезпечення достовірного й оптимізованого моделювання індустріальних технологічних процесів, гнучкого управління енергомережами та переробки ресурсів.

Для досягнення цієї мети було окреслено та виконано такі етапи дослідження:

- системний аналіз можливостей рушія Minecraft та інтерфейсу Forge API для визначення оптимальних шляхів збереження даних у сутностях блоків (BlockEntity);
- проєктування модульної архітектури енергомережі на базі стандарту Forge Energy (FE) для реалізації плавного розподілу струму між генераторами, кабелями та споживачами;
- розробка та впровадження функціональних вузлів системи, зокрема генераторів, машин для багатоетапної обробки ресурсів, а також динамічних інструментів і складних агрокультур;

- реалізація клієнт-серверної синхронізації для графічних інтерфейсів (GUI) та впровадження системи користувацьких рецептів на основі JSON;
- проведення тестування системи на предмет оптимізації обчислень у головному потоці сервера та коректності візуалізації на стороні клієнта.

Методологічну базу дослідження складають принципи об’єктно-орієнтованого програмування, розділення відповідальності (Separation of Concerns) та дата-орієнтований (Data-Driven) підхід. Технічна реалізація базується на використанні мови Java (JDK 21.0.8) [4] у середовищі IntelliJ IDEA [5]. Як платформу застосовано Minecraft 1.20.1 у зв'язці з Forge MDK 47.4.10 [1]. Для стандартизації мапінгів методів рушія використано Parchment mappings [6], а процес збирання проєкту автоматизовано за допомогою Gradle [7].

Розроблений програмний комплекс (IndustrialCore) має високу практичну цінність як інструмент для інтерактивної візуалізації інженерних мереж, а також як еталонна база для вивчення принципів створення клієнт-серверних додатків. Результати роботи свідчать про те, що спроєктована архітектура забезпечує високу стабільність симуляції, мінімізує мережеве навантаження під час оновлення станів механізмів та легко піддається подальшому масштабуванню.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз існуючих запитів цільової аудиторії на індустріальні модифікації

Середовище гри Minecraft історично сформувалося не лише як розважальна платформа, але й як потужний інструмент для моделювання. З моменту появи перших інструментів для створення модифікацій (Modding API), запит аудиторії на впровадження індустріальних та логістичних систем залишається одним із найвищих. Гравці та адміністратори багатокористувацьких серверів прагнуть розширити базовий ігровий процес механіками автоматизації, видобутку, обробки ресурсів та управління енергетичними мережами.

Аналіз сучасних форумів, платформ дистрибуції (CurseForge, Modrinth) та спільнот розробників [8] дозволяє виділити ключові запити сучасної цільової аудиторії щодо індустріальних модифікацій:

- стандартизація енергомереж. У минулому розробники створювали власні типи енергії (наприклад, EU, MJ), що призводило до несумісності модифікацій між собою. Сучасний запит вимагає використання єдиного стандарту, яким на платформі Forge є Forge Energy (FE);
- висока продуктивність та оптимізація серверної частини. Головною проблемою індустріальних збірок є падіння частоти оновлення сервера (TPS – Ticks Per Second) через велику кількість механізмів та труб/кабелів, які постійно обмінюються даними. Аудиторія потребує систем з оптимізованим алгоритмом передачі ресурсів;
- підтримка актуальних версій ігрового рушія. З виходом нових версій гри (зокрема 1.20.1) внутрішня архітектура Minecraft зазнає значних змін. Багато класичних проєктів припиняють підтримку, створюючи вакуум на ринку для сучасних індустріальних систем;
- зрозумілий поріг входження. Перевантаженість інтерфейсів та надмірна складність виробничих ланцюгів відштовхує нових користувачів.

Існує попит на модифікації, які пропонують інтуїтивно зрозумілі процеси (наприклад, переробка руди в пил, а пилу – у злитки) з поступовим масштабуванням складності.

Діаграма розподілу популярності категорій модифікацій на платформі CurseForge зображена на рисунку 1.1.

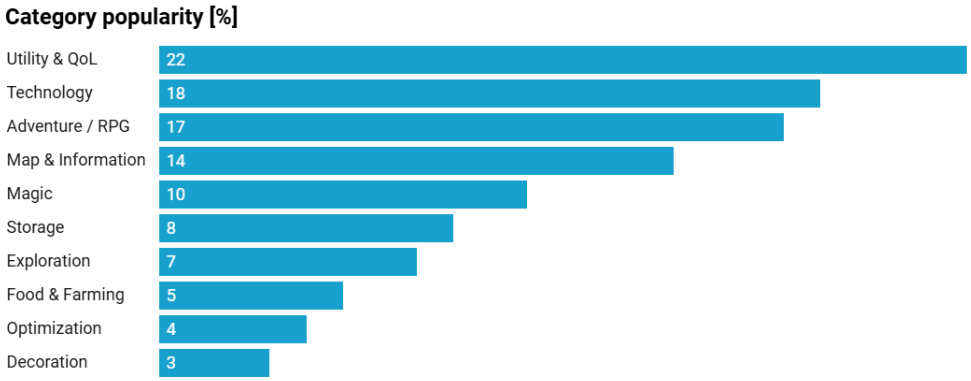

Рисунок 1.1 – Діаграма розподілу популярності категорій модифікацій на платформі CurseForge

Відповідно до виявлених запитів, створення нової системи моделювання повинно базуватися на принципах модульності, сумісності зі стандартами екосистеми Forge та мінімізації навантаження на обчислювальні потоки сервера.

1.2 Моніторинг та порівняльний аналіз існуючих рішень

Для визначення архітектурних переваг та недоліків існуючих індустріальних систем було проведено моніторинг провідних рішень у цьому сегменті. На сучасному ринку програмних додатків для Minecraft виділяються кілька ключових конкурентів, що пропонують схожий функціонал.

1.2.1 Industrial Craft 2

Це один із найстаріших та найвпливовіших проєктів, який заклав основи індустріального моделювання [9].

Переваги:

- глибоко пропрацьована багаторівнева система механізмів;
- класична та впізнавана концепція (дробарка, електропіч, екстрактор).

Недоліки:

- використання закритої власної енергетичної системи (Energy Units – EU), що унеможливлює інтеграцію з сучасними модами;
- застаріла кодова база ускладнює портування на версії 1.20+ і створює проблеми зі швидкістю рендерингу графічних інтерфейсів.

Зовнішній вигляд базових механізмів модифікації Industrial Craft 2 зображено на рисунку 1.2.

Рисунок 1.2 – Базові механізми модифікації Industrial Craft 2 [9]

1.2.2 Mekanism

Сучасний і надзвичайно популярний комплексний мод, що використовує стандартизовану енергію (FE) [10].

Переваги:

- величезна кількість контенту;
- можливість п'ятикратного збільшення виходу ресурсів з руди;
- високоякісні моделі.

Недоліки:

- система є гіпертрофованою (overpowered), що руйнує баланс базової гри;
- логіка універсальних труб (Universal Cables) створює надмірне навантаження на мережевий трафік сервера через постійне оновлення NBT-даних при перетіканні газів, рідин та енергії.

Зовнішній вигляд багаторівневої системи обробки ресурсів та мережі труб у Mekanism зображено на рисунку 1.3.

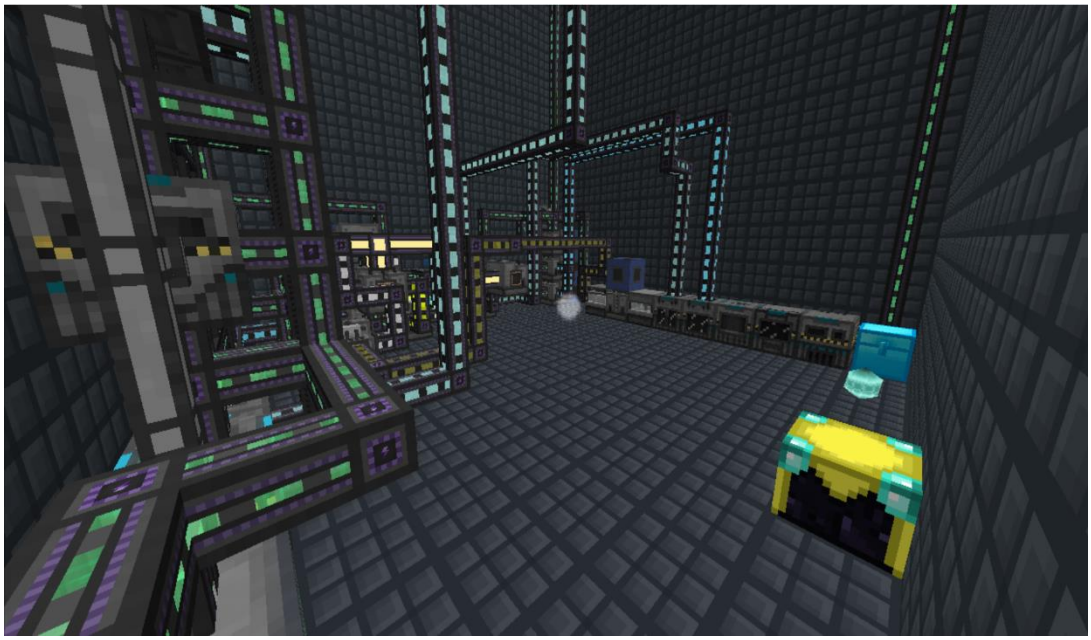

Рисунок 1.3 – Багаторівневі системи обробки ресурсів та мережі труб у Mekanism [10]

1.2.3 Thermal Series (Thermal Expansion)

Модульна система, яка свого часу стандартизувала поняття RF-енергії (попередника сучасного FE) [11].

Переваги:

- висока модульність (система розділена на базову частину, машини, динамо-генератори);
- відмінна оптимізація обробки даних (Data-Driven підхід).

Недоліки:

- в останніх версіях проєкт змінив парадигму обробки ресурсів, відмовившись від деяких класичних ланцюгів, що звузило варіативність для гравців, які шукають класичний індустріальний досвід.

Зовнішній вигляд модульної структури механізмів та генераторів у Thermal Expansion зображено на рисунку 1.4.

Рисунок 1.4 – Модульна структура механізмів та генераторів у Thermal Expansion [11]

1.2.4 Create

Модифікація нового покоління, що пропонує принципово інший підхід [12].

Переваги:

– вражаюча візуалізація процесів на базі кінетичної енергії (обертання валів, шестерень).

Недоліки:

– повна відсутність концепції електричного струму (FE);

– симуляція величезної кількості рухомих елементів вимагає значних ресурсів від відеокарти клієнта (GPU rendering), що робить мод важкодоступним для користувачів зі слабким апаратним забезпеченням.

Зовнішній вигляд кінетичних механізмів та динамічної візуалізації процесів у модифікації Create зображено на рисунку 1.5.

Рисунок 1.5 – Кінетичні механізми та динамічна візуалізація процесів у модифікації Create [12]

1.2.5 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

Для систематизації зібраних даних було складено порівняльну таблицю характеристик існуючих рішень та розроблюваної системи IndustrialCore (Таблиця 1.1).

Таблиця 1.1 – Порівняльний аналіз існуючих індустріальних модифікацій

Назва модифікації	Енергетична система	Універсальна сумісність (FE)	Вплив на продуктивність сервера (TPS)	Поріг входження для користувача
Industrial Craft 2	Власна (EU)	Низька (потребує конвертерів)	Середній	Середній
Mekanism	Гібридна (FE, Joules)	Висока	Високий (через складну логіку труб)	Високий
Thermal Series	Стандартна (FE/RF)	Висока	Низький (відмінна оптимізація)	Низький
Create	Кінетична (SU)	Відсутня	Високий вплив на клієнт (FPS)	Високий
IndustrialCore (Розроблювана)	Стандартна (FE)	Висока	Низький (оптимізація BlockEntity)	Низький

1.3 Концептуальна привабливість та унікальність модифікації IndustrialCore

Враховуючи проаналізовані недоліки існуючих рішень, програмна система «IndustrialCore» проєктується як оптимальна ланка (так звана «золота середина») між класичним ігровим процесом (vanilla) та гіпертрофованими індустріальними комплексами. Концептуальна привабливість розроблюваної

модифікації для кінцевого користувача базується на кількох ключових принципах:

- інтуїтивна зрозумілість та класичний баланс. Модифікація пропонує перевірену часом парадигму обробки ресурсів, яка є інтуїтивно зрозумілою: видобуток руди, її подрібнення у дробарці (Macerator) для збільшення виходу матеріалу (створення пилу), та подальше переплавлення в електропечі (Electric Furnace). Це знижує поріг входження для нових гравців. Крім того, додано унікальні гібридні інструменти, такі як Бронзовий бур (Bronze Drill), який копає зону 3x3 блоки, поєднуючи високу ефективність із залежністю від заряду енергії (FE), що зберігає баланс гри;
- синергія механіки та агрокультури. Проєкт виходить за рамки виключно машинної обробки, поєднуючи індустрію з сільським господарством. Наприклад, для створення ізольованих мідних кабелів (Insulated Copper Cable), які убезпечують гравця від ураження струмом, необхідно вирощувати спеціальну двоблочну культуру – льон (Flax). Це створює більш глибокий і різноманітний ігровий досвід;
- універсальна сумісність (Forge Energy). Усі механізми та генератори розроблені з використанням єдиного сучасного стандарту Forge Energy (FE). Це означає, що IndustrialCore працюватиме «з коробки» з будь-якими іншими сучасними модифікаціями, виступаючи як надійним джерелом, так і споживачем енергії у великих збірках (модпаках);
- сучасна візуалізація та оптимізація відображення. Попри оптимізовану серверну логіку, клієнтська частина мода використовує сучасні підходи до рендерингу. Яскравим прикладом є Вітряк (Windmill), для якого написано кастомний рендерер (BlockEntityRenderer). Візуальне обертання лопатей обчислюється виключно на стороні клієнта за допомогою математичної лінійної інтерполяції (Lerp) між кадрами (partialTick), що створює максимально плавну анімацію без жодного навантаження на мережевий трафік чи логіку сервера.

Завдяки цим факторам, IndustrialCore є привабливим рішенням як для індивідуальних гравців, так і для авторів багатокористувацьких збірок, яким потрібен надійний, красивий та оптимізований індустріальний «двигун».

1.4 Стратегія дистрибуції та монетизації програмного продукту

Життєвий цикл програмного забезпечення не закінчується на етапі написання коду; критично важливим аспектом є стратегія його розповсюдження (дистрибуції) та монетизації. Оскільки ринок модифікацій Minecraft має власну унікальну екосистему, розповсюдження системи «IndustrialCore» орієнтоване на спеціалізовані платформи з вбудованими механізмами фінансового заохочення розробників.

Стратегія монетизації проєкту включає три основні напрямки (Business-to-Consumer та Business-to-Business моделі):

а) платформна монетизація (CurseForge Reward Program). Основним майданчиком для дистрибуції обрано платформу CurseForge (під управлінням компанії Overwolf), яка є світовим лідером у сегменті ігрових модифікацій [8], а також альтернативну відкриту платформу Modrinth. CurseForge пропонує розробникам програму винагород (Reward Program), в рамках якої автор модифікації отримує спеціальні бали (CurseForge Points) за кожне унікальне завантаження та щоденне активне використання мода гравцями через офіційний лаунчер. Накопичені бали легально конвертуються у реальні грошові кошти або подарункові сертифікати (через платіжні системи PayPal/Payoneer), забезпечуючи розробнику стабільний пасивний дохід, прямо пропорційний популярності продукту;

б) краудфандинг та підтримка спільноти. Для формування ядра лояльної аудиторії передбачено створення сторінок на платформах краудфандингу (Patreon, Buy Me a Coffee або Ko-fi). Гравці, які бажають підтримати розробку, можуть оформлювати щомісячні підписки. Натомість вони отримують ексклюзивні переваги:

1) ранній доступ до тестових (Beta) версій нових оновлень;

2) можливість голосувати за впровадження нових машин;

3) спеціальні ролі на сервері спільноти у Discord. Ця модель стимулює прямий зв'язок між автором та цільовою аудиторією;

в) B2B інтеграції та партнерства з ігровими хостингами. Популярні модифікації генерують значний трафік, що робить їх привабливими для комерційних багатокористувацьких серверів та компаній, які надають послуги ігрового хостингу (наприклад, BisectHosting, Apex Hosting). Монетизація у цьому напрямку може відбуватися шляхом:

1) надання промокодів від хостинг-партнерів на сторінці завантаження модифікації (отримання відсотка від продажів за реферальною програмою);

2) інтеграції рекламних банерів партнерського хостингу безпосередньо в конфігураційні файли або головне меню модифікації;

3) додавання попередньо налаштованих спонсорських серверів у список багатокористувацької гри (Multiplayer Server List) за замовчуванням для всіх користувачів, які встановили мод.

Завантажену модифікацію IndustrialCore на платформу CurseForge зображено на рисунку 1.6.

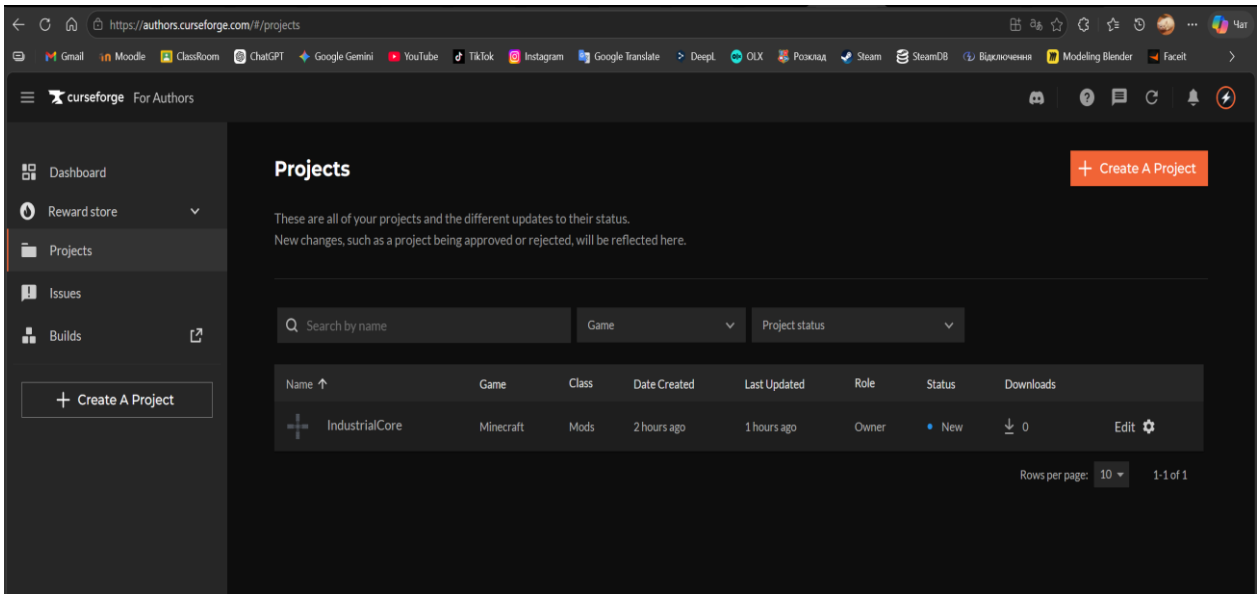

Рисунок 1.6 – Завантажена модифікація IndustrialCore на платформу CurseForge

Отже, розробка «IndustrialCore» має не лише академічну та дослідницьку цінність, але й чітко визначений комерційний потенціал. Використання сучасних майданчиків дистрибуції дозволяє трансформувати програмний продукт у джерело доходу, забезпечуючи подальший розвиток та підтримку проєкту.

1.5 Постановка задачі

Головною метою даної дипломної кваліфікаційної роботи є проєктування та програмна реалізація модульної клієнт-серверної системи моделювання індустріальних процесів «IndustrialCore» у середовищі Minecraft.

Для досягнення цієї мети необхідно розв'язати комплекс інженерних та алгоритмічних задач:

- розробити децентралізовану систему управління енергетичними потоками на базі стандарту Forge Energy (FE), де кожен вузол мережі (генератор, кабель, споживач) діє як автономна сутність (BlockEntity) із власною логікою прийому та віддачі струму;
- спроєктувати алгоритми оптимізації обчислень у головному потоці сервера для забезпечення стабільного показника TPS (Ticks Per Second) під час передачі енергії розгалуженими кабельними мережами;
- реалізувати набір взаємопов'язаних індустріальних машин для повного циклу переробки ресурсів: від подрібнення (Macerator) до переплавлення (Electric Furnace) та формування сплавів (Metal Former);
- розробити графічні інтерфейси користувача (GUI) з архітектурою, що підтримує двосторонню синхронізацію даних між клієнтом та сервером у режимі реального часу (заряд батареї, прогрес роботи);
- інтегрувати систему у базовий ігровий процес через генерацію нових руд (олово, свинець), агрокультур (льон) та створення гібридних енергозалежних інструментів (бронзовий бур).

1.6 Функціональні та нефункціональні вимоги

Для забезпечення високої якості кінцевого продукту було сформовано перелік вимог до системи.

Функціональні вимоги:

- система повинна генерувати енергію (FE) шляхом спалювання палива (враховуючи час горіння) або використання сили вітру (залежно від погодних умов: дощ, гроза);
- система повинна забезпечувати рівномірне та автоматичне перетікання енергії через блоки кабелів до найближчих машин-споживачів;
- система повинна наносити шкоду ігровим сутностям (Entity), які контактують з неізольованим кабелем під напругою;
- машини повинні обробляти вхідні ресурси (руду, пил, злитки) відповідно до зареєстрованих рецептів лише за умови наявності достатнього заряду енергії у їхньому внутрішньому буфері;
- інтерфейси (Menu/Screen) повинні відображати актуальний рівень енергії та прогрес виконання операції (у відсотках або пропорційних графічних елементах).

Нефункціональні вимоги:

- продуктивність: Алгоритм балансування енергії в кабелях не повинен викликати рекурсивних циклів (infinite loops), що призводять до зупинки сервера;
- сумісність: Внутрішні буфери енергії повинні приймати та віддавати струм іншим модифікаціям, які використовують стандарт IEnergyStorage;
- масштабованість: Додавання нових рецептів для машин повинно здійснюватися через дата-орієнтований (Data-Driven) підхід (JSON файли), без необхідності внесення змін до Java-коду;

- кодова база: Проєкт має компілюватися на Java Development Kit (JDK) версії 21 з використанням мапінгів Parchment для забезпечення чистоти та зрозумілості назв методів ігрового рушія.

1.7 Висновки за розділом 1

У першому розділі було проведено комплексний аналіз предметної області, який підтвердив високий попит аудиторії на індустріальні модифікації з інтуїтивним класичним ігровим процесом та сучасною оптимізованою архітектурою. Дослідження існуючих аналогів (Industrial Craft 2, Mekanism, Thermal Expansion, Create) дозволило виявити їхні слабкі місця: від закритої системи енергії до надмірного навантаження на обчислювальні потужності клієнта та сервера.

Спроєктована система «IndustrialCore» позиціонується як збалансоване рішення, що використовує єдиний стандарт Forge Energy (FE), поєднує логіку машинобудування з агрокультурою та забезпечує плавну візуалізацію процесів. Було розроблено стратегію дистрибуції та монетизації продукту через платформи CurseForge, Patreon та B2B інтеграції, що доводить комерційний потенціал проєкту.

Також у розділі було чітко сформульовано постановку задачі розробки, окреслено архітектурні межі системи та визначено перелік функціональних і нефункціональних вимог, які стануть основою для подальшого програмного проєктування у наступних розділах роботи.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РОЗРОБКИ

2.1 Специфіка платформи: Minecraft Java Edition та архітектурні відмінності від Bedrock Edition

Екосистема Minecraft концептуально поділяється на дві основні гілки, кожна з яких має фундаментально різну архітектуру та підходи до розширення функціоналу: Java Edition та Bedrock Edition. Для реалізації комплексної програмної системи моделювання індустріальних процесів вибір правильної базової платформи є критично важливим етапом.

Bedrock Edition, написана мовою C++, архітектурно орієнтована на максимальну кросплатформність та продуктивність на мобільних пристроях і консолях. Розширення ігрового процесу в цій версії відбувається через офіційний прикладний програмний інтерфейс (API) – систему Add-ons [13]. Вона використовує дата-орієнтований підхід (файли JSON) для опису властивостей об'єктів та мову JavaScript для написання базової поведінкової логіки. Попри безпеку та простоту розповсюдження, ця архітектура є суворо ізольованою (sandboxed). Вона категорично обмежує розробнику доступ до низькорівневих процесів ігрового рушія. У Bedrock Edition неможливо створити власну систему передачі енергії між сусідніми блоками з високою частотою оновлення (TPS), реалізувати кастомний рендеринг (наприклад, плавне обертання лопатей вітряка без прив'язки до ігрових тіків) або створити глибоку інвентарну логіку для машин-споживачів.

Натомість, Java Edition надає розробникам безпрецедентний рівень доступу до внутрішньої архітектури рушія. Попри відсутність офіційного Modding API від компанії-розробника Mojang, спільнота створила потужні інструменти на базі маніпуляції байт-кодом (Bytecode Manipulation), рефлексії (Reflection) та систем ін'єкції коду (наприклад, Mixins). Це дозволяє модифікаціям інтегруватися безпосередньо в ядро гри, змінювати методи рушія та створювати власні підсистеми.

Для проєкту «IndustrialCore» було обрано саме Java Edition, оскільки лише ця архітектура підтримує концепцію BlockEntity – «розумних» блоків, здатних виконувати складні розрахунки, зберігати нестандартні типи даних (через Named Binary Tag – NBT) та взаємодіяти з іншими блоками у реальному часі через систему Capabilities.

2.2 Вибір мови програмування та інструментарію розробки

Для забезпечення стабільності, швидкодії та легкості підтримки кодової бази розроблюваної системи було сформовано сучасний стек технологій.

2.2.1 Програмна платформа та фреймворк

Основою розробки є мова програмування Java (JDK 21.0.8) [4]. Вибір 21-ї версії зумовлений її статусом довгострокової підтримки (LTS), високою продуктивністю збирача сміття (Garbage Collector) та наявністю сучасних синтаксичних конструкцій, що полегшують написання безпечного коду.

Як фреймворк (завантажувач) для інтеграції з грою (версії 1.20.1) обрано Forge API (Mod Development Kit 47.4.10) [1]. Forge є де-факто галузевим стандартом для великих технічних та індустріальних модів. Він надає стандартизовану систему подій (Event Bus), вбудовану систему енергії Forge Energy (FE) та розширену систему реєстрації об'єктів (DeferredRegister), що гарантує уникнення конфліктів ідентифікаторів з іншими модулями.

2.2.2 Система мапінгів та деобфускація

Вихідний код Minecraft є обфускованим – усі назви класів, методів та змінних замінені на беззмістовні символи (наприклад, func_12345_a()). Для забезпечення читабельності коду рушія під час розробки використовуються мапінги (Mappings) – словники перекладу обфускованого коду у зрозумілий.

У проєкті застосовано Parchment mappings [6]. На відміну від стандартних мапінгів Mojang (Official Mappings), які перекладають лише назви класів та методів, Parchment додатково надає зрозумілі назви параметрів методів та детальну документацію (Javadocs). Це критично важливо під час проєктування складних систем, таких як кастомний рендеринг вітряка (partialTick, poseStack) або генерація світу, оскільки дозволяє розробнику точно розуміти призначення кожної змінної рушія.

2.2.3 Середовище розробки та система збирання

Процес написання коду вимагає використання надійного інтегрованого середовища розробки (IDE), яке здатне ефективно обробляти великі масиви даних деобфускованого рушія Minecraft. Для обґрунтування вибору програмного інструментарію було проведено порівняльний аналіз трьох найпопулярніших рішень, що використовуються у Java-розробці: IntelliJ IDEA (JetBrains), Eclipse (Eclipse Foundation) та Visual Studio Code (Microsoft). Результати порівняння наведено у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз інтегрованих середовищ розробки (IDE) для створення модифікацій Minecraft.

Критерій порівняння	IntelliJ IDEA (Community/Ultimate)	Eclipse IDE	Visual Studio Code
1	2	3	4
Глибина статичного аналізу Java-коду	Висока (найкраща в класі)	Висока	Середня (залежить від плагінів)

Продовження таблиці 2.1

1	2	3	4
Інтеграція з системою збирання Gradle	Нативна, безшовна	Потребує додаткових налаштувань (Buildship)	Часткова (через розширення)
Швидкість індексації великих проєктів (ядро гри)	Висока (асинхронна)	Середня	Висока
Вбудований декомпілятор Java-класів	Присутній (Fernflower)	Відсутній (або базовий)	Потребує сторонніх розширень
Зручність налаштування середовища (ForgeGradle)	Максимальна (автоматичне розгортання)	Задовільна (часті конфлікти шляхів)	Низька (складне налаштування launch.json)

Як видно з наведеного аналізу, IntelliJ IDEA [5] є беззаперечним лідером для завдань, пов'язаних із модингом Minecraft. Це середовище забезпечує глибокий статичний аналіз коду, підтримує інструменти декомпіляції класів Minecraft "на льоту" та має розширену вбудовану підтримку системи Gradle, що критично важливо для інтеграції Forge MDK. На відміну від VS Code, IDEA не потребує ручного конфігурування десятків розширень, а на відміну від Eclipse — пропонує значно сучасніший інтерфейс та швидшу систему індексації файлів.

Управління залежностями, компіляція проєкту та запуск тестових клієнтів і серверів автоматизовано за допомогою системи збирання Gradle [7]. Завдяки інтеграції плагіна ForgeGradle, процес розгортання робочого

середовища (завантаження деобфускованого рушія гри, налаштування бібліотек, підключення мапінгів Parchment) виконується автоматично на основі конфігураційного файлу build.gradle. Це гарантує ідентичність середовища розробки на будь-якій робочій станції та нівелює проблеми з несумісністю версій локальних бібліотек.

2.3 Архітектурна структура клієнт-серверної системи

Базова архітектура Minecraft, навіть у режимі однокористувацької гри (Singleplayer), побудована на суворій клієнт-серверній парадигмі. Локальний клієнт (Client) відповідає за рендеринг графіки, відтворення звуків та обробку вводу користувача. Водночас інтегрований локальний сервер (Logical Server) відповідає за всю ігрову логіку: генерацію світу, розрахунок фізики, оновлення інвентарів та обробку енергетичних мереж.

Проєктування модулів для «IndustrialCore» суворо дотримується цього принципу розділення відповідальності (Separation of Concerns), що гарантує відсутність розсинхронізації (desync) та захист від потенційних вразливостей (наприклад, дублювання предметів).

2.3.1 Базові компоненти ігрового світу: Block та BlockEntity

Для оптимізації використання оперативної пам’яті рушій гри використовує патерн проєктування «Одинак» (Singleton) для базових блоків. Класи, що наслідують Block (наприклад, ThermalGeneratorBlock або CableBlock), існують в єдиному екземплярі на всю гру. Вони не можуть зберігати індивідуальні дані (рівень енергії чи предмети) для конкретної координати у світі. Блок відповідає лише за статичні властивості: хітбокс, швидкість руйнування, модель та базові стани (BlockState), такі як напрямок (FACING) або стан горіння (LIT).

Для наділення блоків унікальними характеристиками та здатністю до обчислень застосовується концепція BlockEntity (сутність блоку). Кожен

встановлений у світі механізм створює власний об’єкт BlockEntity, який прив’язується до конкретних координат.

У розробленій системі класи сутностей (наприклад, MaceratorBlockEntity, WindmillBlockEntity) виконують такі функції:

– зберігання даних (серіалізація та десеріалізація через NBT-теги), таких як кількість енергії (FE) та вміст інвентарю;

– виконання циклічних розрахунків. Метод tick(), що викликається 20 разів на секунду виключно на серверній стороні, обробляє логіку генерації енергії, перевірку рецептів та переплавлення ресурсів;

– оновлення стану. Коли механізм отримує достатньо енергії, BlockEntity змінює змінну прогресу та надсилає сигнал серверу про необхідність оновлення (setChanged()), щоб зберегти дані на жорсткий диск.

Взаємодію класу Block та BlockEntity на прикладі механізмів зображено на рисунку 2.1.

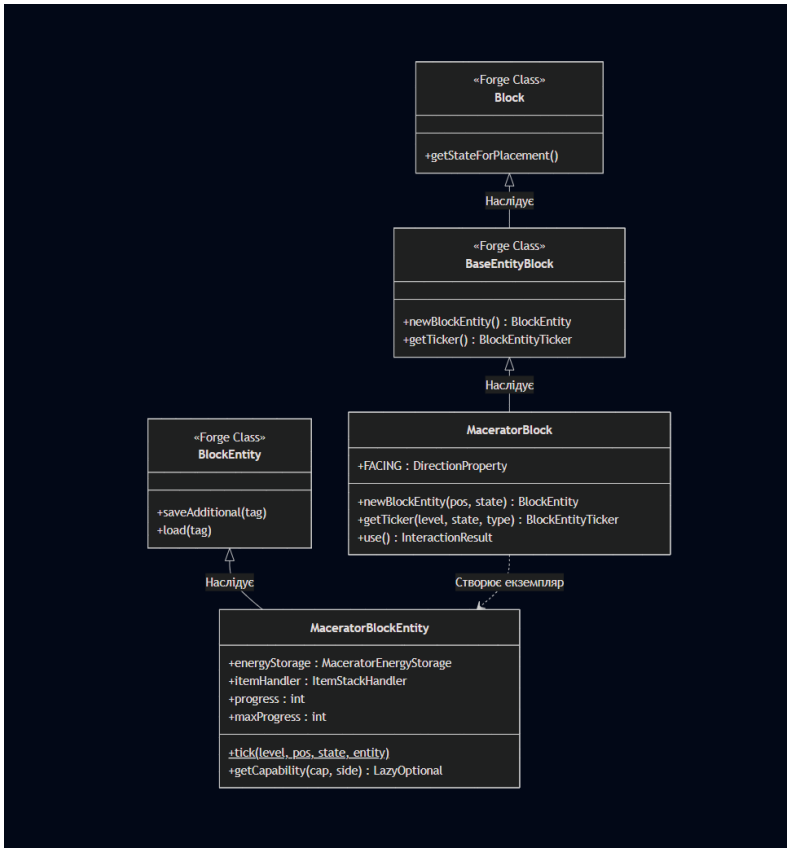

Рисунок 2.1 – UML-діаграма взаємодії класу Block та BlockEntity на прикладі механізмів IndustrialCore

Спроєктована архітектура також враховує оптимізацію мережевого трафіку. Наприклад, логіка переміщення енергії між кабелями (CableBlockEntity) працює виключно на сервері. Клієнту не надсилаються пакети з оновленням рівня енергії всередині кожного кабелю, що мінімізує затримки (lag) при розбудові великих промислових мереж.

2.3.2 Система графічних інтерфейсів (Menu та Screen)

Оскільки логіка обробки ресурсів відбувається на сервері, а гравець взаємодіє з візуальним відображенням на клієнті, виникає необхідність безпечної та синхронної передачі даних між цими двома вузлами. Ця проблема вирішується розбиттям графічного інтерфейсу користувача (GUI) на два взаємопов'язані компоненти: Menu (серверний контейнер) та Screen (клієнтський екран).

Компонент Menu (Контейнер):

- клас AbstractContainerMenu (наприклад, ElectricFurnaceMenu) існує одночасно на клієнті та на сервері. Його головне завдання – синхронізація даних (Data Synchronization) та управління слотами інвентарю гравця й механізму;
- для передачі цілочисельних значень (рівень енергії, прогрес крафту) без створення власних мережевих пакетів використовується об'єкт ContainerData. Наприклад, в EnergyStorageMenu цей об'єкт постійно зчитує поточний та максимальний рівень енергії з EnergyStorageBlockEntity на сервері та миттєво оновлює ці значення у своїй клієнтській копії. Це унеможливлює ситуацію, коли гравець забирає предмет, який фактично ще не створився на сервері.

Компонент Screen (Екран):

- клас AbstractContainerScreen (наприклад, ElectricFurnaceScreen) обробляється виключно на боці клієнта та відповідає за візуалізацію (рендер). Використовуючи дані, отримані з відповідного Menu, метод renderBg() динамічно відмальовує текстури;

– у розробленій модифікації графічні елементи (стрілки прогресу, індикатори напруги, полум'я генератора) розраховуються математично. Наприклад, ширина індикатора енергії у сховищі обчислюється пропорційно за формулою: (поточна енергія / максимальна енергія) * максимальна ширина текстури.

Синхронізацію даних між сервером та клієнтом при відкритті графічного інтерфейсу механізму зображено на рисунку 2.2.

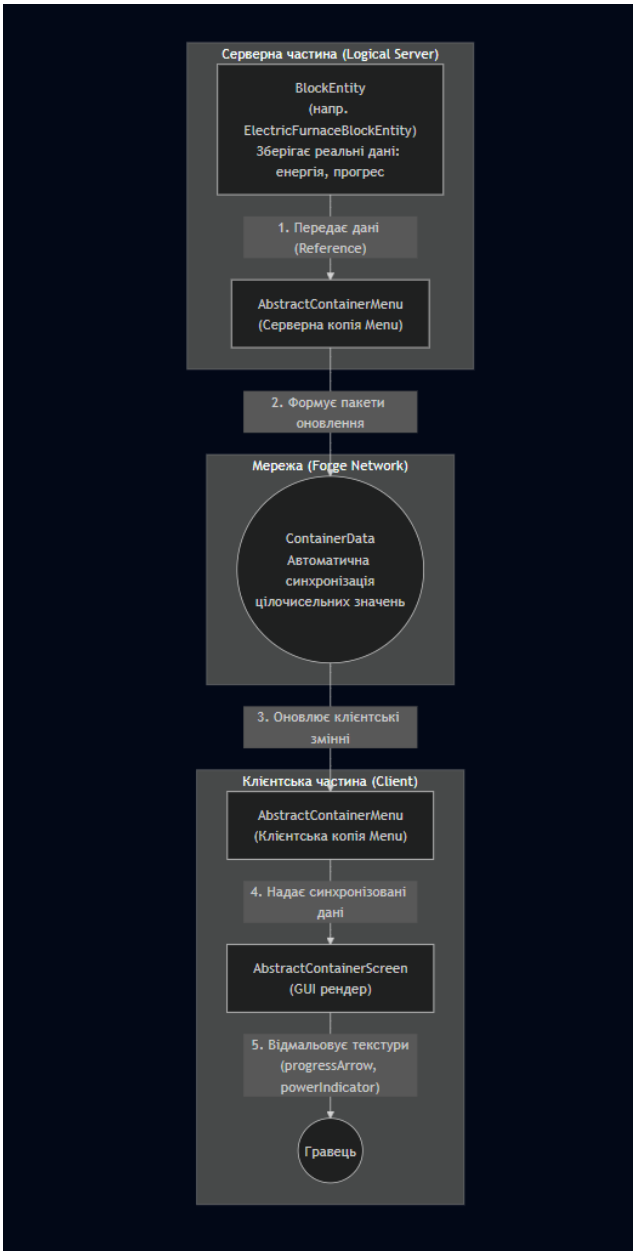

Рисунок 2.2 – Схема синхронізації даних між сервером та клієнтом при відкритті графічного інтерфейсу механізму

Завдяки розділенню на Menu та Screen, система інтерфейсів гарантує захист від чіт-модифікацій (наприклад, підробки пакетів для дублювання предметів у слотах), оскільки остаточне рішення щодо переміщення предметів чи витрати енергії завжди приймається захищеною серверною частиною.

2.4 Логіка взаємодії компонентів системи

Спроєктована архітектура модифікації «IndustrialCore» складається з десятків різнотипних об'єктів: генераторів, споживачів, кабелів, предметів та інструментів. Для забезпечення їхньої ефективної взаємодії без виникнення жорсткої зв'язності коду (Tight Coupling) застосовано передові архітектурні патерни, надані інтерфейсом Forge API.

2.4.1 Стандартизація енергообміну (Система Capabilities та Forge Energy)

Традиційний підхід у об'єктно-орієнтованому програмуванні для надання об'єктам нових властивостей передбачає використання множинного успадкування або реалізацію великої кількості інтерфейсів (наприклад, IEnergyReceiver, IInventory). Проте у масштабних системах це призводить до перевантаження кодової бази та несумісності між різними модифікаціями.

Для вирішення цієї проблеми у розробленій системі використано систему «Можливостей» (Capabilities) від Forge [2]. Це патерн проєктування, подібний до компонентного підходу (Entity-Component System). Замість того, щоб перевіряти, чи є сусідній блок конкретно «Тепловим генератором», кабель робить універсальний запит: «Чи має цей блок можливість приймати енергію з певного боку?».

Енергетична система проєкту повністю побудована на стандарті Forge Energy (FE). Клас ModEnergyStorage розширює базовий функціонал EnergyStorage, додаючи метод onEnergyChanged(), який автоматично

сигналізує серверу про необхідність збереження даних при будь-якій зміні балансу.

Логіка маршрутизації енергії у кабелях (CableBlockEntity) розроблена за принципом децентралізованого балансування (Fluid-like transfer). На кожному ігровому тіку кабель виконує два послідовні кроки:

- пріоритетна віддача машинам: Кабель перевіряє сусідні блоки. Якщо сусід має інтерфейс енергообміну (Capability) ForgeCapabilities.ENERGY і не є іншим кабелем (тобто це механізм), кабель намагається передати йому весь свій заряд струму. Безпека доступу до пам'яті забезпечується обгорткою LazyOptional, яка запобігає помилкам NullPointerException у разі раптового руйнування механізму гравцем;
- плавне перетікання мережею: Якщо сусідніми блоками є інші кабелі, система обчислює різницю їхніх зарядів. Енергія передається лише в тому випадку, якщо поточний кабель має більший заряд, ніж сусідній. Передається рівно половина від різниці зарядів, що створює ефект плавного перетікання струму без ефекту «пінг-понгу» (постійного перекидання однієї й тієї ж порції енергії туди-назад), який часто перевантажує сервери в інших модифікаціях.

Ця система також масштабована на предмети. Наприклад, акумуляторний блок (EnergyStorageBlockEntity) через слот інвентарю робить запит до предмета «Бронзовий бур» на наявність інтерфейсу енергообміну (Capability) і, у разі підтвердження, передає йому заряд.

Алгоритм запиту та передачі енергії через систему Forge Capabilities зображено на рисунку 2.3.

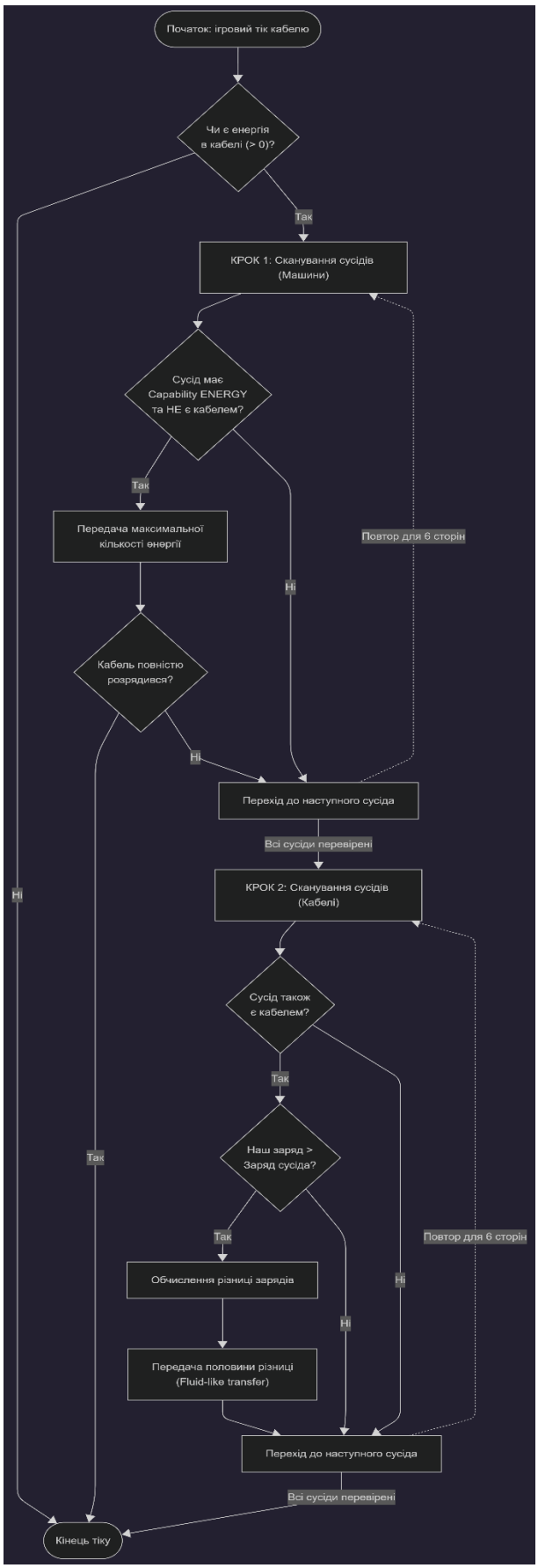

Рисунок 2.3 – Алгоритм запиту та передачі енергії через систему Forge Capabilities

2.4.2 Дата-орієнтований підхід (Data-Driven)

Сучасна архітектура Minecraft відходить від жорсткого програмування (hardcoding) логіки на користь дата-орієнтованого підходу. Це означає, що більшість ігрових правил, конфігурацій та рецептів винесено за межі Java-коду у зовнішні JSON-файли. Така структура дозволяє адміністраторам серверів легко балансувати модифікацію без необхідності втручання у вихідний код проєкту.

У системі «IndustrialCore» дата-орієнтований (Data-Driven) [3] підхід реалізовано у трьох ключових напрямках:

- користувацькі типи рецептів (Custom Recipes). Замість того, щоб жорстко прописувати в коді, що «з руди виходить злиток», було розроблено власний тип рецептів electric_furnace_smelting та відповідний модуль перетворення даних ElectricFurnaceRecipeSerializer. Він зчитує JSON-файли з папки recipes. Це дозволяє електропечі обробляти як стандартні ванільні ресурси (їжу, залізо), так і специфічний індустріальний пил (наприклад, lead_dust_electric_smelting.json);
- таблиці видобутку (Loot Tables). Логіка випадіння предметів при руйнуванні блоків повністю контролюється JSON-файлами. Наприклад, для агрокультури льону (flax_crop.json) прописано строгі умови (conditions): льоноволокно та насіння випадають виключно тоді, коли значення властивості блоку age дорівнює 4 (максимальна стадія росту), а властивість half дорівнює lower (щоб уникнути подвійного випадіння при руйнуванні двоблочної рослини). Окремі умови прописані для олов'яної і свинцевої руди з урахуванням зачарувань «Шовковий дотик» (Silk Touch) та «Фортуна» (Fortune) ;
- динамічні моделі та стани блоків (Blockstates / Multipart). Візуальна мережа кабелів базується на складних просторових моделях. JSON-файл copper_cable.json використовує директиву multipart, яка динамічно об'єднує візуальні елементи. Блок постійно перевіряє свої булеві стани (up,

down, north тощо). Якщо до кабелю під'єднується інший блок з інтерфейсом енергообміну (Capability), булевий стан змінюється на true, і система рендерингу автоматично домальовує «відросток» кабелю у потрібному напрямку.

Зв'язок Java-модулей перетворення даних із зовнішніми JSON-ресурсами зображено на рисунку 2.4.

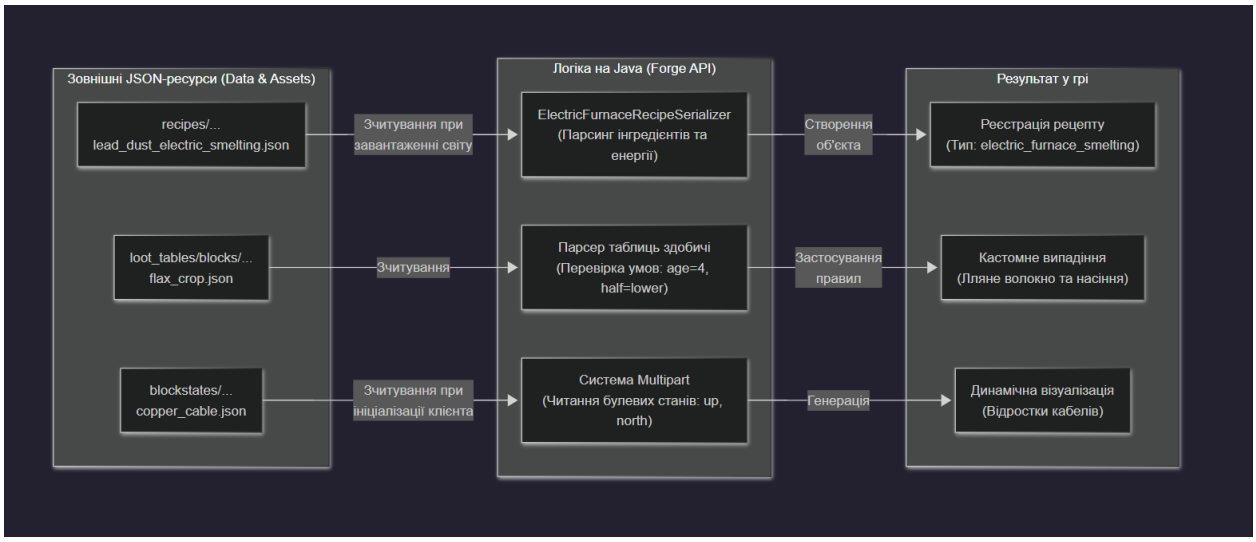

Рисунок 2.4 – Структура дата-орієнтованої (Data-Driven) архітектури: зв'язок Java-модулей перетворення даних із зовнішніми JSON-ресурсами

Використання дата-орієнтованої архітектури суттєво підвищило модульність проєкту, забезпечивши його сумісність із глобальними змінами ігрового світу за допомогою стандартизованих пакетів даних (Data Packs).

2.5 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної роботи розробленої системи «IndustrialCore» та уникнення падіння частоти кадрів на клієнті (FPS) чи затримок обробки логіки на сервері (TPS), було визначено мінімальні та рекомендовані вимоги до апаратного і програмного забезпечення.

Оскільки архітектура Minecraft 1.20.1 використовує оновлені бібліотеки рушія (зокрема оновлений рендер OpenGL та систему управління

пам'яттю), системні вимоги розділені на програмні та апаратні, а також на клієнтську та серверну частини.

Вимоги до програмного забезпечення (спільні для клієнта та сервера):

- операційна система: Windows 10/11 (64-bit), macOS 11.0+ або дистрибутиви Linux (Ubuntu 20.04+);
- середовище виконання Java: Java Runtime Environment (JRE) або Java Development Kit (JDK) версії не нижче 21.0. Використання застарілих версій Java (наприклад, Java 8 або 17) призведе до помилки запуску гри;
- базова платформа: Minecraft Java Edition версії 1.20.1;
- програмний завантажувач: Forge версії 47.4.10 або новішої.

Клієнтська частина відповідає за рендеринг складних просторових моделей (кабелів) та кастомних анімацій (вітряк), що створює навантаження на відеокарту та оперативну пам'ять.

Вимоги до апаратного забезпечення клієнта (Player Client):

- процесор (CPU): Чотириядерний процесор із тактовою частотою від 3.0 ГГц (наприклад, Intel Core i5 8-го покоління або AMD Ryzen 5 серії 2000 та вище);
- оперативна пам'ять (RAM): Мінімум 4 ГБ виділеної пам'яті (Allocated RAM) для процесу Java. Рекомендовано 6-8 ГБ для стабільної роботи зі встановленими пакетами ресурсів (Resource Packs) та уникнення мікрофрізів під час роботи Garbage Collector;
- відеокарта (GPU): Дискретна відеокарта з підтримкою OpenGL 4.4+ (наприклад, серія NVIDIA GeForce GTX 1050 або AMD Radeon RX 560);
- накопичувач: SSD-накопичувач для швидкого завантаження ресурсів модифікації.

Серверна частина не обробляє графіку, але виконує сотні математичних розрахунків щосекунди (передача енергії по кабелях, обробка рецептів у механізмах). Оскільки ігровий цикл (Tick Loop) Minecraft є переважно однопотоковим (Single-threaded), ключовим параметром сервера є не кількість ядер, а продуктивність на одне ядро (Single-core performance).

Вимоги до апаратного забезпечення сервера (Dedicated Server):

- процесор (CPU): Процесор із високою базовою частотою (3.5 ГГц+) та потужним одним ядром (Intel Core i7/i9 або AMD Ryzen 7/9 останніх поколінь) ;
- оперативна пам'ять (RAM): 4 ГБ для малих груп (до 5 гравців), 8 ГБ і більше для масштабних індустріальних серверів.

2.6 Висновки за розділом 2

У другому розділі було проведено комплексне проєктування архітектури розроблюваної системи. Здійснено порівняльний аналіз платформ, який обґрунтував вибір Minecraft Java Edition як єдиного середовища, що надає необхідний інструментарій (байт-код маніпуляції, систему BlockEntity) для створення складних індустріальних мереж, на відміну від обмеженого API версії Bedrock.

Визначено та затверджено технологічний стек: розробка ведеться мовою Java (JDK 21) у середовищі IntelliJ IDEA за допомогою системи збирання Gradle та фреймворку Forge (MDK 47.4.10). Особливу увагу приділено використанню мапінгів Parchment, які забезпечили деобфускацію коду рушія та полегшили проєктування кастомного рендерингу.

Спроєктована архітектура базується на суворому розділенні відповідальності (Separation of Concerns). Розрахунки процесів обробки ресурсів винесено у серверні сутності (BlockEntity), тоді як візуалізація та синхронізація з користувачем делеговані графічним обгорткам (AbstractContainerMenu та AbstractContainerScreen). Такий підхід гарантує захист від розсинхронізації даних та читерства.

Для розв'язання проблеми надмірного навантаження на сервер (TPS drops) під час передачі енергії застосовано патерн «Можливостей» (Forge Capabilities) у поєднанні зі стандартом Forge Energy (FE). Це дозволило створити оптимізований алгоритм «плавного перетікання» (Fluid-like transfer) струму по кабелях без безкінечних рекурсивних циклів. Крім того, систему

адаптовано до сучасних дата-орієнтованих (Data-Driven) стандартів: логіка рецептів, випадіння предметів (Loot Tables) та побудова динамічних моделей (Multipart) винесені у зовнішні JSON-ресурси, що забезпечує максимальну масштабованість проєкту. Сформовані вимоги до апаратного та програмного забезпечення гарантують стабільну експлуатацію системи на клієнтських та серверних станціях.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА АЛГОРИТМИ ФУНКЦІОНУВАННЯ СИСТЕМИ

3.1 Реалізація ключових алгоритмів функціонування механізмів

В основі програмної реалізації модифікації «IndustrialCore» лежить обробка ігрових подій (Ticks), яка ініціює виклик методів логіки для кожної активної сутності блоку (BlockEntity). Програмний код поділено на класи-генератори, класи-провідники та класи-споживачі.

Структурну схему взаємодії цих програмних компонентів під час обробки ігрового такту наведено на рисунку 3.1.

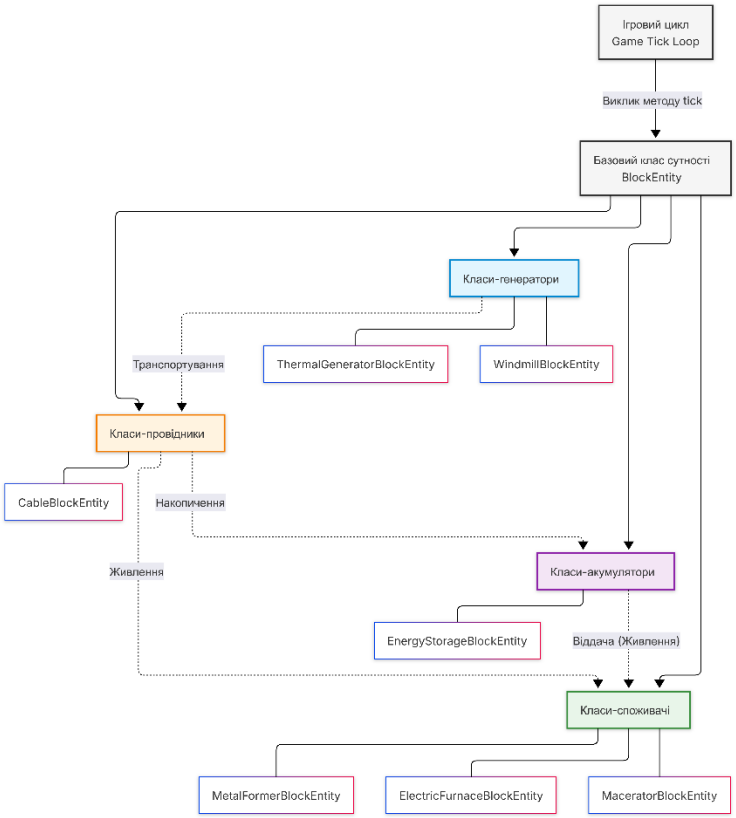

Рисунок 3.1 – Структурна схема програмних модулів обробки ігрових подій

3.1.1 Алгоритм генерації та балансування енергомережі

Генерація енергії реалізована на прикладі класу ThermalGeneratorBlockEntity. Алгоритм його роботи у методі tick() розбитий на дві стадії: поглинання палива та перетворення його на енергію. Коли паливо

потрапляє у вхідний слот, система зчитує його ванільний час горіння (Burn Time) через ForgeHooks.getBurnTime() і зменшує цей показник за допомогою константи прискорення BURN_SPEED_MULTIPLIER, оптимізуючи процес для промислових потреб. Під час горіння генератор додає визначену кількість енергії (ENERGY_PER_TICK) до внутрішнього сховища ModEnergyStorage, перевіряючи, щоб загальний заряд не перевищив максимальну ємність (10000 FE).

Транспортування енергії здійснюється класом CableBlockEntity. Зважаючи на високий ризик зниження продуктивності сервера при розрахунках великих кабельних мереж, було реалізовано двоетапний алгоритм балансування струму:

- етап пріоритетної передачі машинам: На кожному ігровому тіку кабель сканує 6 суміжних блоків (напрямки Direction.values()). Якщо сусідній блок має підтримку ForgeCapabilities.ENERGY і не є іншим кабелем, система викликає метод receiveEnergy(), передаючи машині максимально можливу кількість енергії;
- етап мережевого балансування (Fluid-like transfer): Якщо сусідній блок також є кабелем (neighbor instanceof CableBlockEntity), алгоритм порівнює їхні рівні заряду. Якщо поточний кабель має більше енергії, обчислюється різниця, яка потім ділиться навпіл: transferAmount = (currentEnergy - neighborStored) / 2. Ця порція енергії безпечно передається сусіду, що створює плавний ефект розподілу струму без виникнення рекурсивних циклів і перевантаження обчислювального потоку.

3.1.2 Логіка автоматизованої обробки ресурсів

Обробка ресурсів реалізується класами MaceratorBlockEntity (Дробарка) та MetalFormerBlockEntity (Металоформувальна машина). Алгоритм функціонування цих машин базується на суворій перевірці станів (State Check).

У кожному ігровому тіку машина виконує метод hasRecipe(), який перевіряє наявність валідних інгредієнтів у вхідних слотах інвентарю (об'єкт ItemStackHandler) та достатню кількість вільного місця у вихідному слоті. У разі успішної перевірки рецепту, алгоритм переходить до перевірки енергії.

Лише за умови наявності необхідної порції заряду (energyStorage.getEnergyStored() >= ENERGY_REQ), машина списує енергію за допомогою кастомного методу consumeEnergy() та збільшує інкремент змінної progress.

Коли змінна progress досягає значення maxProgress, викликається метод craftItem(), який зменшує кількість вхідного ресурсу на 1 (shrink(1)) та створює новий об'єкт ItemStack у вихідному слоті. Важливою архітектурною деталлю є виклик методу setChanged() при кожній зміні інвентарю або енергії, що примусово маркує «чанк» (chunk) як змінений, гарантуючи збереження даних при зупинці сервера.

3.2 Реалізація клієнт-серверного обміну даними та візуалізації

Оскільки логіка генерації та переробки обчислюється на сервері, візуалізація цих процесів для гравця потребує синхронізації даних із мінімальними затримками мережі (Latency).

3.2.1 Синхронізація графічних інтерфейсів (GUI)

Для реалізації графічних інтерфейсів застосовано архітектуру зв'язки класів AbstractContainerMenu та AbstractContainerScreen.

Проблема мережевої синхронізації цілочисельних змінних (рівень заряду, прогрес переплавлення, загальний час горіння) вирішена за допомогою об'єкта ContainerData. Наприклад, у класі ElectricFurnaceBlockEntity ініціалізується анонімний клас ContainerData, де метод get(int index) повертає поточні значення змінних progress (індекс 0), maxProgress (індекс 1) та статус наявності енергії (індекс 2).

При відкритті інтерфейсу серверну копію ElectricFurnaceMenu прив'язано до клієнтської. Система Forge автоматично виявляє зміни значень у ContainerData і генерує легкотермінові мережеві пакети для оновлення цих цифр на боці клієнта. Далі клас візуалізації ElectricFurnaceScreen у методі renderBg() математично перетворює ці цифри на координати текстур, відмальовуючи стрілку прогресу (renderProgressArrow) та індикатор напруги (renderPowerIndicator) через вбудований інструментарій GuiGraphics.blit().

Синхронізований графічний інтерфейс Електропечі зображено на рисунку 3.2.

Рисунок 3.2 – Візуальне відображення графічного інтерфейсу Електропечі з синхронізованими індикаторами прогресу та енергії

3.2.2 Клієнтський рендеринг та плавна анімація

Для об'єктів із динамічною геометрією, таких як Вітряк (WindmillBlockEntity), стандартної моделі відображення через файли JSON недостатньо [3]. Обертання лопатей реалізовано через клас WindmillRenderer, який наслідує BlockEntityRenderer.

Для забезпечення ідеально плавної анімації, що не залежить від частоти оновлення сервера (20 TPS) і прив'язана до частоти кадрів монітора гравця (FPS), використано алгоритм лінійної інтерполяції (Lerp). У методі рендерингу поточний кут повороту обчислюється методом Mth.lerp(partialTick, blockEntity.prevRotation, blockEntity.rotation). Тут partialTick вказує на частку часу, що минула між двома ігровими тіками. Далі система матричних перетворень PoseStack виконує трансляцію (translate) лопатей до центру блоку та їх поворот (mulPose). Для коректного відображення тіней алгоритм динамічно зчитує рівень освітлення середовища перед вітряком (LevelRenderer.getLightColor()) і застосовує його до кожного вертекса за допомогою матриці нормалей (Matrix3f). Такий підхід повністю розвантажує сервер від обчислення графіки, делегуючи ці завдання GPU клієнтського комп'ютера.

3.3 Програмна реалізація сценаріїв взаємодії з користувачем

Взаємодія користувача з віртуальним середовищем у розробленій системі не обмежується лише пасивним спостереженням за роботою механізмів. Вона включає безпосередній вплив на світ через динамічні інструменти, взаємодію з небезпечними елементами мережі та ведення сільського господарства для індустріальних потреб.

3.3.1 Механіка енергозалежних інструментів

Класична механіка інструментів у Minecraft базується на системі міцності (Durability), де кожна дія зменшує цілочисельний показник до повного руйнування об'єкта. У системі «IndustrialCore» реалізовано концепцію інструментів, що перезаряджаються, на прикладі Бронзового бура (BronzeDrillItem).

Клас бура наслідує стандартний PickaxeItem, проте його логіка докорінно змінена шляхом використання системи Capabilities. Через

перевизначений метод initCapabilities до предмета прив'язується провайдер IEnergyStorage із ємністю 40 000 FE. Енергія записується безпосередньо у NBT-теги об'єкта (stack.getOrCreateTag().putInt), що дозволяє предмету зберігати свій стан у процесі переміщення між інвентарями гравця та механізмів (наприклад, під час зарядки в EnergyStorageBlockEntity). Візуалізація заряду реалізована шляхом перевизначення методів відмальовування смужки міцності (isBarVisible, getBarWidth, getBarColor), завдяки чому бур має зелений індикатор, довжина якого обчислюється пропорційно поточному заряду.

Алгоритм взаємодії бура зі світом (копання зони 3x3 блоки) розроблено у методі mineBlock. Для точного визначення зони руйнування алгоритм використовує технологію трасування променів (Raytracing) через getPlayerPOVHitResult. Це дозволяє отримати нормаль площини, по якій клікнув гравець (Direction). На основі цієї нормалі розраховуються осі X, Y, Z для циклічного обходу сусідніх 8 блоків.

Усередині циклу система перевіряє твердість блоків та їхню сумісність з інструментом. За кожен успішно зруйнований блок (включно із сусідніми) система списує 100 FE. Якщо рівень енергії падає нижче 100 FE, ефективність інструмента штучно знижується до рівня базового видобутку вручну (метод getDestroySpeed повертає 1.0f).

Роботу Бронзового бура та віджети заряду в інвентарі зображено на рисунку 3.3.

Рисунок 3.3 – Демонстрація роботи Бронзового бура: зона руйнування 3x3 та віджети заряду предмета в інвентарі

3.3.2 Взаємодія зі складними агрокультурами та ізоляцією

Для створення ефекту глибини виробничих ланцюгів у модифікації реалізовано взаємозв'язок між біологічними ресурсами та промисловими компонентами. Яскравим прикладом є вирощування льону (FlaxCropBlock) для отримання матеріалів ізоляції кабелів.

Льон спроєктовано як складну двоблочну агрокультуру, що вимагає специфічного алгоритмічного підходу до обробки ігрових подій (Random Ticks). Для оптимізації розрахунків на сервері, метод isRandomlyTicking повертає true виключно для нижньої половини рослини (DoubleBlockHalf.LOWER), що унеможливлює баг подвійної швидкості росту.

Під час переходу на наступну стадію (метод growFlax), алгоритм перевіряє наявність вільного простору або блоку льону над поточною координатою та синхронно оновлює стан (BlockState) обох половин. Руйнування однієї частини рослини ініціює ланцюгову реакцію через метод updateShape, який замінює іншу частину на блок повітря (Blocks.AIR), гарантуючи цілісність сутності у світі.

Здобуті з льону матеріали використовуються для створення ізольованих кабелів (InsulatedCableBlock). Цей клас є прямою ілюстрацією застосування принципу поліморфізму в об'єктно-орієнтованому програмуванні. Базовий клас CableBlock містить метод entityInside, який постійно сканує свій хітбокс. Якщо всередину потрапляє гравець або моб (LivingEntity), а рівень енергії в кабелі перевищує 0, система завдає суб'єкту шкоди типу "удар струмом" (damageSources().lightningBolt()).

Дочірній клас InsulatedCableBlock просто перевизначає цей метод, залишаючи його тіло порожнім. Це блокує виконання батьківського коду, елегантно та з нульовими затратами продуктивності сервера реалізуючи механіку безпечної ізольованої мережі.

3.4 Висновки за розділом 3

У третьому розділі було детально висвітлено програмну реалізацію ключових алгоритмів розробленої системи моделювання «IndustrialCore». Застосовані архітектурні рішення довели свою ефективність при вирішенні нетривіальних інженерних завдань у середовищі Minecraft.

Реалізація децентралізованого алгоритму балансування енергомережі через класи CableBlockEntity та механізм LazyOptional дозволила забезпечити безперервну доставку Forge Energy (FE) до споживачів. Завдяки алгоритму «плавного перетікання» (передача половини різниці зарядів) вдалося повністю усунути ризик рекурсивного зациклення, яке часто стає причиною падіння частоти оновлення (TPS) на серверах.

Автоматизована обробка ресурсів була надійно відокремлена від клієнтської візуалізації. Логіка машин (Дробарки, Електропечі) суворо дотримується паттерну перевірки стану перед споживанням енергії, що унеможливлює втрату ресурсів у разі програмних збоїв. Двосторонню взаємодію з користувачем забезпечено оптимізованими інтерфейсами, де об'єкт ContainerData відповідає за цілісність передачі цілочисельних

параметрів, а класи типу Screen забезпечують їхнє математичне перетворення у зрозумілі графічні індикатори.

Завдяки застосуванню лінійної інтерполяції (Lerp) та partialTick у кастомному рендерері вітряка було досягнуто ідеально плавної анімації механізмів виключно силами клієнтського графічного процесора, що зняло будь-яке навантаження з логічного сервера. Додатково було імплементовано інтерактивні елементи ігрового світу: гібридні енергозалежні інструменти на базі NBT-даних (Бронзовий бур) та складні двоблочні агрокультури з синхронізованими станами руйнування.

У підсумку, програмна реалізація проєкту підтвердила гіпотезу про те, що використання дата-орієнтованого підходу у поєднанні зі стандартом Capabilities та грамотним об'єктно-орієнтованим дизайном дозволяє створювати стабільні, розширювані та високопродуктивні клієнт-серверні модулі симуляції у середовищі ігрового рушія.

4 ЕКСПЛУАТАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 Опис умов експлуатації та розгортання системи

Розгортання (deployment) програмної системи «IndustrialCore» відбувається у два етапи: компіляція вихідного коду та безпосередня інтеграція артефакту в середовище гри.

Оскільки проєкт розроблено на базі платформи Forge, він використовує систему збирання Gradle. Для створення готового до експлуатації продукту виконується консольна команда gradlew build, яка компілює Java-класи, мініфікує та пакує зовнішні JSON-ресурси (рецепти, моделі, текстури, лут-таблиці) і формує єдиний виконуваний файл формату .jar у директорії build/libs.

Експлуатація програми вимагає виконання наступних умов:

- наявність ліцензійного або альтернативного клієнта гри Minecraft версії 1.20.1;
- попередньо ініціалізоване середовище завантажувача Forge (версії 47.4.10 або новішої);
- наявність інстальованої Java версії 21 у системі користувача.

Встановлення модифікації здійснюється шляхом переміщення зкомпільованого файлу до директорії mods у кореневій папці гри. Завдяки універсальній архітектурі (Universal Mod), створений файл є сумісним як із клієнтською частиною (для режиму Singleplayer), так і з виділеними серверами (Dedicated Server), автоматично розмежовуючи логіку рендерингу та серверних розрахунків під час ініціалізації.

4.2 Демонстрація роботи та користувацький досвід

Користувацький досвід (User Experience) у системі спроєктовано за принципом поступового масштабування складності. Для демонстрації роботи алгоритмів розглянемо базовий сценарій взаємодії користувача з

модифікацією: від початкового етапу до побудови повністю автоматизованої індустріальної мережі.

Етап 1: Збір первинних ресурсів та розвиток агрокультури.

На початковому етапі користувач досліджує ігровий світ для видобутку базових ресурсів: олов'яної (tin_ore) та свинцевої (lead_ore) руд, які генеруються у підземних біомах згідно з прописаними правилами Worldgen. Паралельно ініціюється сільськогосподарський процес: гравець висаджує насіння льону. Рослина має складну двоблочну структуру та проходить через 5 стадій росту. Після досягнення фінальної стадії гравець збирає врожай – лляну солому (flax_straw), яка в Дробарці перетворюється на лляне волокно (flax_fiber), що є критично важливим для безпеки мережі.

Процес дозрівання агрокультури льону зображено на рисунку 4.1.

Рисунок 4.1 – Процес дозрівання двоблочної агрокультури льону в ігровому середовищі

Етап 2: Первинна генерація енергії.

Для забезпечення роботи майбутніх механізмів гравець конструює Тепловий генератор (ThermalGeneratorBlock). Розмістивши генератор та забезпечивши його твердим паливом (вугіллям), користувач запускає процес генерації Forge Energy (FE). Графічний інтерфейс чітко сигналізує про роботу через анімацію спалювання полум'я та поступове заповнення шкали

внутрішнього акумулятора до 10 000 FE. Наявність енергії також змінює візуальний стан самого блоку у світі (властивість lit=true).

Етап 3: Побудова мережі та ізоляція провідників.

Транспортування енергії здійснюється мідними кабелями (copper_cable). Алгоритм їхньої роботи передбачає динамічну зміну моделі (Multipart): при розміщенні поруч із генератором кабель автоматично створює візуальне з'єднання. Оскільки базові кабелі завдають шкоди при контакті з ігровою сутністю гравця (удар струмом), користувач комбінує їх із лляним волокном для створення безпечних ізольованих мідних кабелів (insulated_copper_cable).

Побудову енергетичної мережі зображено на рисунку 4.2.

Рисунок 4.2 – Побудова енергетичної мережі: підключення ізольованих кабелів від теплового генератора до споживачів

Етап 4: Автоматизація виробництва та переробка ресурсів.

До підготовленої мережі підключаються машини-споживачі: Дробарка (Macerator) та Електропіч (Electric Furnace).

Гравець поміщає необроблену руду вхідний слот Дробарки. За наявності струму машина подрібнює 1 одиницю руди на 2 одиниці пилу (наприклад, tin_dust), забезпечуючи 200% вихід матеріалу.

Отриманий пил переміщується до Електропечі, де переплавляється на готові злитки.

Отримані мідні та олов'яні злитки об'єднуються у Металоформувальній машині (Metal Former) для створення міцних бронзових пластин (bronze_plate).

Етап 5: Експлуатація енергозалежних інструментів.

Фінальним етапом базового розвитку є створення гібридного інструмента – Бронзового бура (BronzeDrillItem). Гравець поміщає його у спеціальний слот Енергосховища (Energy StorageBlock), де через систему IEnergyStorage відбувається зарядка предмета.

Отримавши заряджений інструмент, гравець використовує його для руйнування породи площею 3x3 блоки. Динамічний зелений індикатор під іконкою інструмента відображає поточний заряд у NBT-тегах, забезпечуючи зручний візуальний контроль.

4.3 Експериментальне тестування програми

Для підтвердження надійності, коректності роботи алгоритмів та відповідності розробленої системи заявленим вимогам було проведено комплексне експериментальне тестування. Процес тестування поділено на два етапи: перевірка функціональної логіки (на відповідність ігровим правилам) та аналіз продуктивності (навантажувальне тестування сервера).

4.3.1 Функціональне тестування

Функціональне тестування (Functional Testing) здійснювалося методом «чорного ящика» (Black-box testing) з боку кінцевого користувача, а також шляхом перевірки внутрішніх станів (NBT-даних) через консоль розробника.

Під час тестування було перевірено такі ключові сценарії (Test Cases):

- генерація енергії: Перевірено роботу Теплового генератора. Підтверджено, що вугілля згорає швидше, ніж у ванільній печі (відповідно до коефіцієнта BURN_SPEED_MULTIPLIER), а внутрішній буфер коректно

акумулює по 50 FE за тік. Робота Вітряка тестувалася зміною погодних умов (команди /weather rain, /weather thunder): зафіксовано коректне математичне збільшення генерації струму в 1.5 та 3 рази відповідно;

- безпека мережі: Штучно змодельовано контакт ігрової сутності (моба та гравця) з оголеним мідним кабелем, по якому проходить струм. Зафіксовано коректне нанесення шкоди. Після заміни блоку на ізольований мідний кабель (InsulatedCableBlock), нанесення шкоди повністю припинилося, що підтверджує правильність перевизначення методу entityInside;
- обробка рецептів: У Дробарку було поміщено 64 одиниці необробленого олова (raw_tin). Перевірка підтвердила, що за наявності енергії машина коректно видає 128 одиниць олов'яного пилу (tin_dust), автоматично зупиняючись, коли енергія вичерпується або вихідний слот заповнюється;
- енергозалежні інструменти: Бронзовий бур було заряджено до максимального значення (40 000 FE). Під час видобутку породи зафіксовано стабільне списання рівно 100 FE за кожен зруйнований блок із зони 3x3. При падінні заряду нижче 100 FE бур коректно втратив швидкість копання, що відповідає логіці методу getDestroySpeed().

Процес тестування взаємодії неізольованих кабелів із сутностями зображено на рисунку 4.3.

Рисунок 4.3 – Процес функціонального тестування взаємодії ізольованих та неізольованих кабелів із сутностями

4.3.2 Тестування продуктивності та стабільності

Найбільш вразливим місцем клієнт-серверних модифікацій є падіння частоти оновлення сервера (TPS – Ticks Per Second) при розбудові великих мереж. Для тестування стабільності «IndustrialCore» було змодельовано екстремальне навантаження (Stress Testing): у світі побудовано мережу з 500 з'єднаних кабелів, 20 генераторів та 50 працюючих машин-споживачів.

Аналіз продуктивності проводився за допомогою вбудованого профайлера Minecraft та інструменту Spark.

Результати тестування:

- стабільність TPS: Навіть при максимальному навантаженні сервер стабільно утримував еталонний показник у 20.0 TPS;
- алгоритм балансування: Децентралізований алгоритм «плавного перетікання» (передача половини різниці заряду між кабелями) довів свою ефективність. Профайлер не зафіксував жодних безкінечних рекурсивних циклів або так званого ефекту «пінг-понгу» (коли енергія безперервно перекидається між двома блоками в межах одного тіку). Метод tick() у CableBlockEntity займав менше 0.5 мілісекунд процесорного часу сервера;
- витоки пам'яті (Memory Leaks): Під час тривалої сесії гри (понад 4 години) та постійного завантаження/вивантаження чанків (Chunk loading/unloading) витоків оперативної пам'яті не виявлено. Обгортки LazyOptional коректно інвалідували свої значення (метод invalidateCaps()) при руйнуванні механізмів, запобігаючи виникненню помилок NullPointerException (Crash).

4.4 Плани подальшого розвитку системи

Попри те, що розроблена система виконує всі поставлені задачі та забезпечує повноцінний цикл обробки твердих ресурсів, її архітектура спроєктована з урахуванням можливості подальшого масштабування.

Перспективні напрямки розвитку модифікації «IndustrialCore» включають:

- інтеграція системи рідин та газів (Fluid Mechanics). Розробка мережі труб для транспортування води, лави та промислового охолоджувача на основі стандарту Forge Fluids. Це дозволить впровадити нові механізми, такі як Помпа (Pump) та Геотермальний генератор;
- багатоклітинні структури (Multiblock Structures). Для пізніх етапів гри (End-game) планується реалізація масштабних об'єктів, таких як Доменна піч (Blast Furnace) або Ядерний реактор, які складатимуться з десятків окремих блоків, об'єднаних єдиним віртуальним контролером (Master BlockEntity);
- система модернізації механізмів (Upgrades). Впровадження слотів розширення для встановлення модулів (Overclockers), які дозволять гравцеві балансувати машини, підвищуючи швидкість їх роботи ціною експоненційного зростання споживання енергії;
- сумісність із системами перегляду рецептів. Розробка інтеграційних модулів (API Plugins) для найпопулярніших інформаційних модифікацій: JEI (Just Enough Items) та EMI. Це дозволить гравцям бачити кастомні рецепти (наприклад, типи electric_furnace_smelting) безпосередньо в ігровому інтерфейсі без використання сторонніх веб-енциклопедій (Wiki).

4.5 Висновки за розділом 4

У четвертому розділі було розглянуто процеси компіляції, розгортання та експлуатації створеної програмної системи. Описано типовий сценарій користувацького досвіду (Use Case), який демонструє логічну послідовність розвитку гравця: від ручного вирощування льону та видобутку ресурсів до побудови замкненого циклу автоматизованої переробки руди та експлуатації енергозалежних інструментів.

Експериментальне тестування підтвердило високу надійність архітектури. Функціональні тести довели коректність обробки рецептів, роботи графічних інтерфейсів та захисної механіки ізольованих кабелів.

Навантажувальне тестування засвідчило, що оптимізований алгоритм балансування Forge Energy (FE) ефективно запобігає перевантаженню процесорних потоків, зберігаючи стабільні 20 TPS навіть при розрахунках розгалужених мереж із сотнями активних вузлів.

Окреслений план подальшого розвитку підтверджує життєздатність проєкту та його готовність до інтеграції нових механік, таких як системи рідин або підтримка плагінів перегляду рецептів (JEI/EMI), що відкриває широкі перспективи для подальшої технічної підтримки та розширення цільової аудиторії модифікації.

ВИСНОВКИ

У дипломній кваліфікаційній роботі бакалавра успішно розв’язано актуальну задачу проєктування та програмної реалізації модульної клієнт-серверної системи моделювання індустріальних процесів «IndustrialCore» у воксельному середовищі Minecraft. Досягнуто головної мети дослідження - створено стабільну, оптимізовану та масштабовану програмну архітектуру, що забезпечує автоматизацію віртуального виробництва.

За результатами виконання роботи можна зробити такі висновки:

- аналіз предметної області та ринку: Проведено ґрунтовний моніторинг сучасних існуючих рішень (Industrial Craft 2, Mekanism, Thermal Expansion, Create). Виявлено потребу аудиторії у продуктах, що поєднують класичний інтуїтивний ігровий процес із сучасною оптимізованою кодовою базою. Розроблено стратегію дистрибуції та монетизації продукту за моделлю B2C та B2B (через платформу CurseForge, краудфандинг та спонсорські інтеграції), що доводить комерційну життєздатність та практичну цінність створеного програмного забезпечення;
- проєктування архітектури та вибір інструментарію: Обґрунтовано вибір Minecraft Java Edition як єдиної платформи з необхідним рівнем доступу до низькорівневих обчислень рушія. Сформовано сучасний технологічний стек: мова програмування Java (JDK 21), фреймворк Forge (MDK 47.4.10) та система мапінгів Parchment для деобфускації коду. Проєктування відбувалося на засадах строгого розділення відповідальності (Separation of Concerns), де розрахунки делеговано серверним сутностям (BlockEntity), а візуалізацію — клієнтським графічним екранам;
- реалізація оптимізованої енергомережі: Розв'язано критичну проблему падіння продуктивності сервера (TPS drops) при передачі енергії. Завдяки застосуванню патерну Forge Capabilities та стандарту Forge Energy (FE) було розроблено децентралізований алгоритм «плавного перетікання»

(передача половини різниці заряду). Це повністю усунуло ризик виникнення безкінечних рекурсивних циклів у розгалужених кабельних мережах;

- програмна реалізація механік взаємодії: Створено цілісну екосистему, яка включає машини для обробки ресурсів (Дробарка, Електропіч, Металоформувальна машина) та генератори (Тепловий, Вітровий). Успішно впроваджено дата-орієнтований підхід (Data-Driven): логіка рецептів, динамічних моделей та випадіння предметів винесена у зовнішні JSON-файли. Розроблено складні інтерактивні елементи: двоблочну агрокультуру льону для ізоляції кабелів та гібридний енергозалежний Бронзовий бур, що динамічно зчитує заряд із NBT-даних інструмента;
- розробка візуальної та мережевої синхронізації: Забезпечено надійний двосторонній зв'язок між сервером та клієнтом для графічних інтерфейсів за допомогою об'єкта ContainerData, що захищає систему від розсинхронізації та вразливостей. Для динамічних об'єктів (лопаті вітряка) імплементовано кастомний рендерер із використанням лінійної інтерполяції (Mth.lerp), який забезпечує ідеально плавну анімацію силами клієнтського GPU, не завантажуючи сервер;
- експериментальне дослідження: Комплексне тестування підтвердило високу надійність створеної архітектури. Функціональні тести довели коректність обробки рецептів та механік захисту. Навантажувальне стрес-тестування зафіксувало стабільну частоту оновлення сервера на рівні 20 TPS при розбудові промислової мережі з понад 500 активними вузлами, без жодних витоків оперативної пам'яті.

Розроблений програмний комплекс повністю відповідає висунутим функціональним та нефункціональним вимогам. Архітектура проєкту має високий рівень модульності, що відкриває широкі перспективи для подальшого масштабування: інтеграції систем транспортування рідин, підтримки інформаційних плагінів (JEI) та розробки масштабних багатоклітинних механізмів. Результати роботи можуть бути використані як

готовий програмний продукт, а також як еталонна база для навчання принципам побудови клієнт-серверних архітектур.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Forge Documentation. Official Minecraft Forge API documentation. Minecraft Forge. URL: https://docs.minecraftforge.net (date of access: 18.04.2026).

2. Forge Capabilities System (1.20.x). Minecraft Forge. URL: https://docs.minecraftforge.net/en/1.20.x/datastorage/capabilities (date of access: 18.04.2026).

3. Introducing JSON (JavaScript Object Notation). JSON. URL: https://www.json.org (date of access: 20.04.2026).

4. Java Platform, Standard Edition 21 API Specification. Oracle. URL: https://docs.oracle.com/en/java/javase/21/docs/api (date of access: 18.04.2026).

5. IntelliJ IDEA: The Capable & Ergonomic Java IDE. JetBrains. URL: https://www.jetbrains.com/idea (date of access: 20.04.2026).

6. ParchmentMC – Modern, open, and community-driven Minecraft mappings. ParchmentMC. URL: https://parchmentmc.org (date of access: 18.04.2026).

7. Gradle Build Tool User Manual. Gradle. URL: https://docs.gradle.org (date of access: 20.04.2026).

8. CurseForge – The Home for the Best Minecraft Mods. CurseForge. URL: https://www.curseforge.com/minecraft (date of access: 03.05.2026).

9. Industrial Craft 2. CurseForge. URL: https://www.curseforge.com/minecraft/mc-mods/industrial-craft (date of access: 03.05.2026).

10. Mekanism. CurseForge. URL: https://www.curseforge.com/minecraft/mc-mods/mekanism (date of access: 03.05.2026).

11. Thermal Expansion. CurseForge. URL: https://www.curseforge.com/minecraft/mc-mods/thermal-expansion (date of access: 03.05.2026).

12. Create. CurseForge. URL: https://www.curseforge.com/minecraft/mc-mods/create (date of access: 03.05.2026).

13. Introduction to Minecraft Bedrock Add-Ons. Microsoft Learn. URL: https://learn.microsoft.com/en-us/minecraft/creator/documents/gettingstarted (date of access: 10.05.2026).

ДОДАТОК А

Технічне завдання

Вступ

Програмна модифікація «IndustrialCore» призначена для впровадження комплексних систем автоматизації, генерації енергії та обробки ресурсів у воксельне середовище гри Minecraft. Вона розроблена для використання гравцями, адміністраторами ігрових серверів та розробниками збірок (модпаків), які потребують збалансованого та оптимізованого інструментарію для моделювання індустріальних процесів. Область застосування охоплює ігрову індустрію та освітні симуляції інженерних мереж, а об’єктом використання є клієнт-серверне середовище Minecraft Java Edition.

А.1 Підстави до розробки

Підставою для розробки є завдання на дипломну роботу зі спеціальності «Інженерія програмного забезпечення», видане у квітні 2026 року. Найменування теми розробки – «Програмна реалізація системи моделювання індустріальних процесів».

А.2 Призначення розробки

Система розроблена для симуляції повного життєвого циклу індустріального виробництва: від видобутку та подрібнення руди до побудови розгалужених енергетичних мереж. Проєкт забезпечує створення децентралізованої архітектури передачі енергії, гарантує захист даних (запобігання дублюванню предметів) через суворе клієнт-серверне розділення та надає інструменти для автоматизації рутинних завдань користувача. Експлуатаційне призначення продукту полягає у розширенні базового ігрового процесу механіками інженерного проєктування, що стимулює розвиток логічного мислення та навичок планування складних систем.

А.3 Вимоги до програми чи програмного виробу

А.3.1 Вимоги до функціональних характеристик

Програмна модифікація «IndustrialCore» розроблена для вирішення широкого спектра завдань, пов’язаних із розбудовою віртуального виробництва. Система забезпечує генерацію універсальної енергії (Forge Energy) за допомогою теплових та кінетичних (вітрових) генераторів, обсяг якої динамічно залежить від зовнішніх умов (наявність палива, погодні явища).

Керування ресурсами є фундаментальною функцією системи. Користувачі мають можливість розгортати машини для обробки матеріалів (Дробарки, Електропечі, Металоформувальні машини), які взаємодіють з енергомережею та автоматизують створення сплавів і технічного пилу. Для побудови мереж розроблено багаторівневу систему кабелів з підтримкою динамічної зміни візуальних моделей (Multipart) та механікою безпеки (необхідність ізоляції кабелів лляним волокном для уникнення шкоди ігровим сутностям).

Застосунок пропонує інструменти для безпосередньої взаємодії зі світом, зокрема енергозалежні гібридні інструменти (Бронзовий бур), які здатні руйнувати породу площею 3x3 блоки та заряджатися від енергомережі. Вхідними даними для системи є дії користувача в графічному інтерфейсі (розміщення ресурсів) та зміни стану ігрового світу. Вихідними результатами є динамічна обробка матеріалів, візуалізація процесів (анімація обертання механізмів, зміна індикаторів заряду) та математичний розподіл енергії по кабельних вузлах.

А.3.2 Вимоги до надійності

Для забезпечення безперебійного функціонування модифікації необхідно дотримуватися суворих технічних вимог щодо оптимізації. Оскільки «IndustrialCore» є клієнт-серверним модулем, архітектура передачі

струму по кабелях має унеможливлювати виникнення рекурсивних зациклень. Використання алгоритму «плавного перетікання» гарантує стабільну частоту оновлення логічного сервера на рівні 20 TPS (Ticks Per Second) навіть при розбудові мереж із сотнями елементів.

Система спроєктована для ефективного реагування на непередбачувані ситуації. Доступ до об'єктів у пам'яті (енергетичних буферів, інвентарів) здійснюється виключно через безпечні обгортки (LazyOptional), що запобігає виникненню критичних помилок та зупинці сервера при раптовому руйнуванні механізмів гравцем. Синхронізація графічних інтерфейсів реалізується через систему ContainerData, що мінімізує мережеві затримки та гарантує цілісність відображених параметрів.

А.3.3 Умови експлуатації

Для коректної роботи модифікації необхідно забезпечити стандартні умови експлуатації комп’ютерної техніки. Оскільки продукт є доповненням до гри Minecraft, ключовою вимогою є наявність встановленого середовища виконання Java (JRE) та ініціалізованого профілю гри з модифікатором Forge.

Рівень підготовки користувача передбачає володіння базовими навичками гри у Minecraft: просторова навігація, взаємодія з блоками та інвентарем. Прогресія в модифікації розроблена за принципами інтуїтивно зрозумілого дизайну. Технічний супровід зводиться до періодичного оновлення файлу модифікації у відповідній директорії.

А.3.4 Вимоги до складу параметрів технічних засобів

Для забезпечення стабільної та швидкої роботи системи, пристрій користувача (або виділений сервер) має відповідати наступним мінімальним технічним характеристикам:

- операційна система: Windows 10/11 (64-bit), macOS 11+, дистрибутиви Linux (Ubuntu 20.04+);

- програмне забезпечення: Minecraft Java Edition версії 1.20.1, завантажувач Forge API (версія 47.4.10 або новіша), Java Development Kit (JDK) або JRE версії 21.0+;
- процесор: чотириядерний процесор із базовою тактовою частотою від 3.0 ГГц;
- оперативна пам’ять: мінімум 4 ГБ виділеної пам’яті для процесу Java (рекомендовано 6–8 ГБ для стабільної роботи без мікрофрізів);
- графічна підсистема: дискретна відеокарта з підтримкою стандарту OpenGL 4.4+;
- вільне місце: до 50 МБ дискового простору для зберігання артефакту модифікації (без урахування розміру самої гри).

А.3.5 Вимоги до інформаційної й програмної сумісності

Для забезпечення високого рівня інформаційної сумісності, розробка базується на дата-орієнтованому підході (Data-Driven). Визначення рецептів обробки ресурсів, параметрів генерації руд (Worldgen), таблиць здобичі (Loot Tables) та станів блоків (Blockstates) здійснюється виключно у форматі JSON. Це гарантує легку інтеграцію з іншими модифікаціями та можливість редагування правил гри адміністраторами серверів без втручання у вихідний Java-код.

Програмна реалізація бекенду (логіки) та фронтенду (рендерингу) об'єднана в єдину кодову базу мовою Java, що використовує інструментарій мапінгів Parchment для взаємодії з деобфускованим рушієм гри. Збереження динамічних даних (рівень енергії в машинах, заряд у бурі) реалізовано через стандартизовані теги формату NBT (Named Binary Tag), що гарантує стійкість даних при перезапусках серверної станції.

А.3.6 Вимоги до маркування й упакування

Дистрибуція системи «IndustrialCore» здійснюється у форматі єдиного зкомпільованого .jar архіву. Продукт поширюється через спеціалізовані онлайн-платформи (наприклад, CurseForge або Modrinth) або шляхом прямого завантаження.

Оскільки система є виключно інтелектуальним цифровим продуктом, будь-які вимоги до фізичного пакування, матеріального маркування чи логістичного транспортування відсутні.

А.3.7 Вимоги до транспортування й збереження

З огляду на цифровий формат постачання модифікації, вимоги до фізичного транспортування чи спеціальних умов складського зберігання не застосовуються. Встановлення, оновлення та видалення програмного забезпечення здійснюється дистанційно шляхом маніпуляцій з файловою системою пристрою користувача.

А.4 Вимоги до програмної документації

Супровідна документація до проєкту включає це технічне завдання та розширену пояснювальну записку (основний текст дипломної роботи). Ці матеріали містять вичерпні відомості про архітектурні патерни, опис роботи ключових алгоритмів (балансування струму, перевірки станів механізмів), UML-діаграми взаємодії класів та вимоги до програмно-апаратного середовища.

А.5 Техніко–економічні показники

Впровадження модифікації має високий потенціал для розширення можливостей оригінальної гри, створюючи нову нішу для любителів інженерного моделювання. Завдяки високій оптимізації коду, використання

«IndustrialCore» дозволяє створювати великі багатокористувацькі сервери індустріальної тематики зі зниженими витратами на оренду обчислювальних потужностей хостинг-провайдерів (за рахунок низького навантаження на процесор). Продукт може розповсюджуватися безкоштовно для формування спільноти з подальшою монетизацією через добровільну підтримку (краудфандинг).

А.6 Стадії й етапи розробки

Розробка системи включає наступні етапи:

- постановка завдання (1 тиждень);
- аналіз предметної області та інструментів Forge (1–2 тижні);
- розробка архітектури енергомережі та реєстрації об'єктів (1 тиждень);
- програмна реалізація механізмів, інструментів та GUI (1–2 тижні);
- внутрішнє тестування та профілювання навантаження (1 тиждень);
- оформлення супровідної документації (1 тиждень).

А.7 Порядок контролю та приймання

Контроль працездатності модифікації базується на комплексному експериментальному тестуванні у локальному середовищі та на виділеному сервері. Функціональне тестування підтверджує коректність роботи алгоритмів обробки ресурсів, генерації енергії, нанесення шкоди ізольованими/неізольованими кабелями та логіки копання 3x3 для бура.

Тестування продуктивності (за допомогою профайлерів) проводиться у стресових умовах (розбудова мережі з понад 500 вузлів) для підтвердження стабільності показника TPS. Після успішного завершення перевірки на відсутність конфліктів ідентифікаторів та витоків пам'яті, артефакт .jar визнається придатним до повноцінної експлуатації та публічного релізу.

ДОДАТОК Б

Текст програми

Примітка

З метою оптимізації обсягу кваліфікаційної роботи та уникнення надмірного перевантаження документа технічною інформацією, у цьому додатку наведено не повний лістинг вихідного коду, а його фрагменти.

До додатку включено головний файл ініціалізації системи IndustrialCore.java, а також вихідний код у повному обсязі з ключових директорій: client.renderer, energy, init, item та recipe. Крім того, для демонстрації застосованих архітектурних та алгоритмічних рішень частково наведено базові класи з директорій block (2 файли), blockentity (2 файли) та screen (2 файли). Наведені фрагменти повною мірою ілюструють логіку обробки даних, клієнт-серверну взаємодію та загальну структуру розробленого програмного забезпечення.

Б.1 Файл CableBlock

```
package com.xqueryy.industrialcore.block;

import net.minecraft.core.BlockPos;
import net.minecraft.core.Direction;
import net.minecraft.world.entity.Entity;
import net.minecraft.world.entity.LivingEntity;
import net.minecraft.world.item.context.BlockPlaceContext;
import net.minecraft.world.level.BlockGetter;
import net.minecraft.world.level.Level;
import net.minecraft.world.level.LevelAccessor;
import net.minecraft.world.level.block.BaseEntityBlock;
import net.minecraft.world.level.block.Block;
import net.minecraft.world.level.block.RenderShape;
import net.minecraft.world.level.block.entity.BlockEntity;
import net.minecraft.world.level.block.state.BlockState;
import net.minecraft.world.level.block.state.StateDefinition;
import net.minecraft.world.level.block.state.properties.BlockStateProperties;
import net.minecraft.world.level.block.state.properties.BooleanProperty;
import net.minecraft.world.phys.shapes.CollisionContext;
import net.minecraft.world.phys.shapes.Shapes;
import net.minecraft.world.phys.shapes.VoxelShape;
import net.minecraftforge.common.capabilities.ForgeCapabilities;
import com.xqueryy.industrialcore.blockentity.CableBlockEntity;
import com.xqueryy.industrialcore.init.ModBlockEntityInit;
import net.minecraft.world.level.block.entity.BlockEntityTicker;
import net.minecraft.world.level.block.entity.BlockEntityType;
```

```
import org.jetbrains.annotations.Nullable;

public class CableBlock extends BaseEntityBlock {
   // Властивості для кожної з 6 сторін (чи є підключення?)
   public static final BooleanProperty UP = BlockStateProperties.UP;
   public static final BooleanProperty DOWN = BlockStateProperties.DOWN;
   public static final BooleanProperty NORTH = BlockStateProperties.NORTH;
   public static final BooleanProperty SOUTH = BlockStateProperties.SOUTH;
   public static final BooleanProperty WEST = BlockStateProperties.WEST;
   public static final BooleanProperty EAST = BlockStateProperties.EAST;

   public CableBlock(Properties pProperties) {
      super(pProperties);
      // За замовчуванням кабель нікуди не підключений
      this.registerDefaultState(this.stateDefinition.any()
            .setValue(UP, false).setValue(DOWN, false)
            .setValue(NORTH, false).setValue(SOUTH, false)
            .setValue(WEST, false).setValue(EAST, false));
   }

   // Цей метод викликається щоразу, коли хтось торкається хітбоксу блоку
   @Override
   public void entityInside(BlockState pState, Level pLevel, BlockPos pPos, Entity
pEntity) {
      if (!pLevel.isClientSide() && pEntity instanceof LivingEntity) {

         // Отримуємо BlockEntity нашого кабелю
         BlockEntity blockEntity = pLevel.getBlockEntity(pPos);

         if (blockEntity != null) {
            // Шукаємо в ньому підтримку енергії Forge

blockEntity.getCapability(ForgeCapabilities.ENERGY).ifPresent(energyStorage -> {

               // Якщо в кабелі прямо зараз є хоч якась енергія (більше 0)
               if (energyStorage.getEnergyStored() > 0) {

                  // Б'ємо струмом!
                  pEntity.hurt(pLevel.damageSources().lightningBolt(), 2.0F);
               }
            });
         }
      }

      super.entityInside(pState, pLevel, pPos, pEntity);
   }

   @Override
   public RenderShape getRenderShape(BlockState pState) {
      return RenderShape.MODEL;
   }
```

```
    @Override
    protected void createBlockStateDefinition(StateDefinition.Builder<Block, BlockState> pBuilder) {
        pBuilder.add(UP, DOWN, NORTH, SOUTH, WEST, EAST);
    }

    // Метод перевіряє, чи можна підключитися до сусіда
    private boolean canConnectTo(LevelAccessor level, BlockPos pos, Direction direction) {
        BlockEntity be = level.getBlockEntity(pos);
        if (be != null) {
            // Перевіряємо, чи має сусідній блок енергетичне Capability з потрібної сторони
            return be.getCapability(ForgeCapabilities.ENERGY, direction.getOpposite()).isPresent();
        }
        return false;
    }

    // Встановлення блоку гравцем: перевіряємо всіх сусідів одразу
    @Nullable
    @Override
    public BlockState getStateForPlacement(BlockPlaceContext pContext) {
        LevelAccessor level = pContext.getLevel();
        BlockPos pos = pContext.getClickedPos();
        return this.defaultBlockState()
                .setValue(UP, canConnectTo(level, pos.above(), Direction.UP))
                .setValue(DOWN, canConnectTo(level, pos.below(), Direction.DOWN))
                .setValue(NORTH, canConnectTo(level, pos.north(), Direction.NORTH))
                .setValue(SOUTH, canConnectTo(level, pos.south(), Direction.SOUTH))
                .setValue(WEST, canConnectTo(level, pos.west(), Direction.WEST))
                .setValue(EAST, canConnectTo(level, pos.east(), Direction.EAST));
    }

    // Цей метод викликається, коли сусідній блок змінюється (наприклад, поставили генератор поруч з існуючим кабелем)
    @Override
    public BlockState updateShape(BlockState pState, Direction pDirection, BlockState pNeighborState, LevelAccessor pLevel, BlockPos pCurrentPos, BlockPos pNeighborPos) {
        boolean canConnect = canConnectTo(pLevel, pNeighborPos, pDirection);
        return switch (pDirection) {
            case UP -> pState.setValue(UP, canConnect);
            case DOWN -> pState.setValue(DOWN, canConnect);
            case NORTH -> pState.setValue(NORTH, canConnect);
            case SOUTH -> pState.setValue(SOUTH, canConnect);
            case WEST -> pState.setValue(WEST, canConnect);
            case EAST -> pState.setValue(EAST, canConnect);
        };
    }

    // --- ФОРМА БЛОКУ (VOXEL SHAPE) ---
```

```
    // Базовий центральний кубик кабелю (від 6 до 10 пікселів у системі координат 16x16)
    private static final VoxelShape CENTER_SHAPE = Block.box(6, 6, 6, 10, 10, 10);
    // Відростки для кожної сторони
    private static final VoxelShape UP_SHAPE = Block.box(6, 10, 6, 10, 16, 10);
    private static final VoxelShape DOWN_SHAPE = Block.box(6, 0, 6, 10, 6, 10);
    private static final VoxelShape NORTH_SHAPE = Block.box(6, 6, 0, 10, 10, 6);
    private static final VoxelShape SOUTH_SHAPE = Block.box(6, 6, 10, 10, 10, 16);
    private static final VoxelShape WEST_SHAPE = Block.box(0, 6, 6, 6, 10, 10);
    private static final VoxelShape EAST_SHAPE = Block.box(10, 6, 6, 16, 10, 10);

    @Override
    public VoxelShape getShape(BlockState pState, BlockGetter pLevel, BlockPos pPos, CollisionContext pContext) {
        VoxelShape shape = CENTER_SHAPE;
        if (pState.getValue(UP)) shape = Shapes.or(shape, UP_SHAPE);
        if (pState.getValue(DOWN)) shape = Shapes.or(shape, DOWN_SHAPE);
        if (pState.getValue(NORTH)) shape = Shapes.or(shape, NORTH_SHAPE);
        if (pState.getValue(SOUTH)) shape = Shapes.or(shape, SOUTH_SHAPE);
        if (pState.getValue(WEST)) shape = Shapes.or(shape, WEST_SHAPE);
        if (pState.getValue(EAST)) shape = Shapes.or(shape, EAST_SHAPE);
        return shape;
    }

    @Nullable
    @Override
    public BlockEntity newBlockEntity(BlockPos pPos, BlockState pState) {
        return new CableBlockEntity(pPos, pState);
    }

    @Nullable
    @Override
    public <T extends BlockEntity> BlockEntityTicker<T> getTicker(Level pLevel, BlockState pState, BlockEntityType<T> pBlockEntityType) {
        if (pLevel.isClientSide()) {
            return null;
        }
        return createTickerHelper(pBlockEntityType, ModBlockEntityInit.CABLE.get(), CableBlockEntity::tick);
    }


}
```

Б.2 Файл ElectricFurnaceBlock

```
package com.xqueryy.industrialcore.block;

import com.xqueryy.industrialcore.blockentity.ElectricFurnaceBlockEntity;
import net.minecraft.core.BlockPos;
import net.minecraft.core.Direction;
```

```
import net.minecraft.server.level.ServerPlayer;
import net.minecraft.world.Containers;
import net.minecraft.world.InteractionHand;
import net.minecraft.world.InteractionResult;
import net.minecraft.world.entity.player.Player;
import net.minecraft.world.item.context.BlockPlaceContext;
import net.minecraft.world.level.Level;
import net.minecraft.world.level.block.BaseEntityBlock;
import net.minecraft.world.level.block.Block;
import net.minecraft.world.level.block.RenderShape;
import net.minecraft.world.level.block.entity.BlockEntity;
import net.minecraft.world.level.block.entity.BlockEntityTicker;
import net.minecraft.world.level.block.entity.BlockEntityType;
import net.minecraft.world.level.block.state.BlockState;
import net.minecraft.world.level.block.state.StateDefinition;
import net.minecraft.world.level.block.state.properties.BlockStateProperties;
import net.minecraft.world.level.block.state.properties.DirectionProperty;
import net.minecraft.world.phys.BlockHitResult;
import net.minecraftforge.common.capabilities.ForgeCapabilities;
import net.minecraftforge.network.NetworkHooks;

public class ElectricFurnaceBlock extends BaseEntityBlock {
   public static final DirectionProperty FACING =
BlockStateProperties.HORIZONTAL_FACING;

   public ElectricFurnaceBlock(Properties pProperties) {
      super(pProperties);
      this.registerDefaultState(this.stateDefinition.any().setValue(FACING,
Direction.NORTH));
   }

   @Override
   public BlockState getStateForPlacement(BlockPlaceContext pContext) {
      return this.defaultBlockState().setValue(FACING,
pContext.getHorizontalDirection().getOpposite());
   }

   @Override
   protected void createBlockStateDefinition(StateDefinition.Builder<Block,
BlockState> pBuilder) {
      pBuilder.add(FACING);
   }

   @Override
   public RenderShape getRenderShape(BlockState pState) {
      return RenderShape.MODEL;
   }

   @Override
   public BlockEntity newBlockEntity(BlockPos pPos, BlockState pState) {
      return new ElectricFurnaceBlockEntity(pPos, pState);
   }
```

```
    @Override
    public <T extends BlockEntity> BlockEntityTicker<T> getTicker(Level pLevel,
BlockState pState, BlockEntityType<T> pBlockEntityType) {
        if (pLevel.isClientSide()) return null;
        return (lvl, pos, state, be) -> {
            if (be instanceof ElectricFurnaceBlockEntity furnace) {
                ElectricFurnaceBlockEntity.tick(lvl, pos, state, furnace);
            }
        };
    }

    @Override
    public InteractionResult use(BlockState pState, Level pLevel, BlockPos pPos, Player
pPlayer, InteractionHand pHand, BlockHitResult pHit) {
        if (!pLevel.isClientSide()) {
            BlockEntity entity = pLevel.getBlockEntity(pPos);
            if(entity instanceof ElectricFurnaceBlockEntity) {
                NetworkHooks.openScreen(((ServerPlayer)pPlayer),
(ElectricFurnaceBlockEntity)entity, pPos);
            } else {
                throw new IllegalStateException("Our Container provider is missing!");
            }
        }
        return InteractionResult.sidedSuccess(pLevel.isClientSide());
    }

    @Override
    public void onRemove(BlockState pState, Level pLevel, BlockPos pPos, BlockState
pNewState, boolean pIsMoving) {
        // Найважливіша перевірка: чи ми дійсно ЗЛАМАЛИ блок, а не просто змінили
його стан
        // (наприклад, коли піч вмикається і змінює текстуру на "гарячу", речі випадати
НЕ повинні)
        if (pState.getBlock() != pNewState.getBlock()) {

            BlockEntity blockEntity = pLevel.getBlockEntity(pPos);
            if (blockEntity != null) {
                // Шукаємо інвентар (IItemHandler) у нашому блоці

blockEntity.getCapability(ForgeCapabilities.ITEM_HANDLER).ifPresent(itemHandler
-> {

                    // Проходимося по всіх слотах (вхідний, вихідний, слот для палива)
                    for (int i = 0; i < itemHandler.getSlots(); i++) {
                        var stack = itemHandler.getStackInSlot(i);

                        // Якщо слот не порожній, викидаємо предмет у світ
                        if (!stack.isEmpty()) {
                            Containers.dropItemStack(pLevel, pPos.getX(), pPos.getY(),
pPos.getZ(), stack);
                        }
```

```
                }
            });
        }
    }

    // Обов'язково викликаємо стандартний метод, щоб блок остаточно видалився зі світу
    super.onRemove(pState, pLevel, pPos, pNewState, pIsMoving);
  }
}
```

Б.3 Файл CableBlockEntity

```
package com.xqueryy.industrialcore.blockentity;

import com.xqueryy.industrialcore.energy.ModEnergyStorage;
import com.xqueryy.industrialcore.init.ModBlockEntityInit;
import net.minecraft.core.BlockPos;
import net.minecraft.core.Direction;
import net.minecraft.nbt.CompoundTag;
import net.minecraft.world.level.Level;
import net.minecraft.world.level.block.entity.BlockEntity;
import net.minecraft.world.level.block.state.BlockState;
import net.minecraftforge.common.capabilities.Capability;
import net.minecraftforge.common.capabilities.ForgeCapabilities;
import net.minecraftforge.common.util.LazyOptional;
import net.minecraftforge.energy.IEnergyStorage;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

public class CableBlockEntity extends BlockEntity {

  // Ємність 500 FE: достатньо для плавного потоку, але не створює ефекту "прихованої батареї"
  public final ModEnergyStorage energyStorage = new ModEnergyStorage(500, 500, 500) {
    @Override
    public void onEnergyChanged() {
      setChanged();
    }
  };

  private LazyOptional<IEnergyStorage> lazyEnergyHandler = LazyOptional.empty();

  public CableBlockEntity(BlockPos pPos, BlockState pBlockState) {
    super(ModBlockEntityInit.CABLE.get(), pPos, pBlockState);
  }

  @Override
  public @NotNull <T> LazyOptional<T> getCapability(@NotNull Capability<T> cap, @Nullable Direction side) {
```

```
        if (cap == ForgeCapabilities.ENERGY) {
            return lazyEnergyHandler.cast();
        }
        return super.getCapability(cap, side);
    }

    @Override
    public void onLoad() {
        super.onLoad();
        lazyEnergyHandler = LazyOptional.of(() -> energyStorage);
    }

    @Override
    public void invalidateCaps() {
        super.invalidateCaps();
        lazyEnergyHandler.invalidate();
    }

    @Override
    protected void saveAdditional(CompoundTag pTag) {
        pTag.putInt("energy", energyStorage.getEnergyStored());
        super.saveAdditional(pTag);
    }

    @Override
    public void load(CompoundTag pTag) {
        super.load(pTag);
        energyStorage.setEnergy(pTag.getInt("energy"));
    }

    public static void tick(Level pLevel, BlockPos pPos, BlockState pState,
CableBlockEntity pBlockEntity) {
        if (pLevel.isClientSide()) return;

        int currentEnergy = pBlockEntity.energyStorage.getEnergyStored();
        if (currentEnergy <= 0) return;

        for (Direction direction : Direction.values()) {
            BlockEntity neighbor = pLevel.getBlockEntity(pPos.relative(direction));

            if (neighbor != null && !(neighbor instanceof CableBlockEntity)) {
                neighbor.getCapability(ForgeCapabilities.ENERGY,
direction.getOpposite()).ifPresent(neighborEnergy -> {
                    if (neighborEnergy.canReceive()) {
                        int energyToGive = pBlockEntity.energyStorage.getEnergyStored();
                        int accepted = neighborEnergy.receiveEnergy(energyToGive, false);
                        if (accepted > 0) {
                            pBlockEntity.energyStorage.extractEnergy(accepted, false);
                        }
                    }
                });
            }
```

```
        }

        // Оновлюємо значення після того, як частина енергії могла піти в машини
        currentEnergy = pBlockEntity.energyStorage.getEnergyStored();
        if (currentEnergy <= 0) return;

        for (Direction direction : Direction.values()) {
            BlockEntity neighbor = pLevel.getBlockEntity(pPos.relative(direction));

            if (neighbor instanceof CableBlockEntity neighborCable) {
                int neighborStored = neighborCable.energyStorage.getEnergyStored();

                // Якщо в нас більше енергії, ніж у сусіда - ділимося.
                // Жорстка різниця навпіл: плавно, як вода, і без жодного пінг-понгу.
                if (currentEnergy > neighborStored) {
                    int transferAmount = (currentEnergy - neighborStored) / 2;

                    if (transferAmount > 0) {
                        int accepted =
neighborCable.energyStorage.receiveEnergy(transferAmount, false);
                        if (accepted > 0) {
                            pBlockEntity.energyStorage.extractEnergy(accepted, false);
                            currentEnergy -= accepted;
                        }
                    }
                }
            }
        }
    }
}
```

Б.4 Файл ElectricFurnaceBlockEntity

```
package com.xqueryy.industrialcore.blockentity;

import com.xqueryy.industrialcore.init.ModBlockEntityInit;
import com.xqueryy.industrialcore.init.ModRecipes;
import com.xqueryy.industrialcore.recipe.ElectricFurnaceRecipe;
import com.xqueryy.industrialcore.screen.ElectricFurnaceMenu;
import net.minecraft.core.BlockPos;
import net.minecraft.core.Direction;
import net.minecraft.nbt.CompoundTag;
import net.minecraft.network.chat.Component;
import net.minecraft.world.MenuProvider;
import net.minecraft.world.SimpleContainer;
import net.minecraft.world.entity.player.Inventory;
import net.minecraft.world.entity.player.Player;
import net.minecraft.world.inventory.AbstractContainerMenu;
import net.minecraft.world.inventory.ContainerData;
import net.minecraft.world.item.ItemStack;
import net.minecraft.world.item.crafting.RecipeType;
```

```
import net.minecraft.world.item.crafting.SmeltingRecipe;
import net.minecraft.world.level.Level;
import net.minecraft.world.level.block.entity.BlockEntity;
import net.minecraft.world.level.block.state.BlockState;
import net.minecraftforge.common.capabilities.Capability;
import net.minecraftforge.common.capabilities.ForgeCapabilities;
import net.minecraftforge.common.util.LazyOptional;
import net.minecraftforge.energy.EnergyStorage;
import net.minecraftforge.energy.IEnergyStorage;
import net.minecraftforge.items.IItemHandler;
import net.minecraftforge.items.ItemStackHandler;

import java.util.Optional;

public class ElectricFurnaceBlockEntity extends BlockEntity implements MenuProvider
{

    public final ItemStackHandler itemHandler = new ItemStackHandler(2) {
        @Override
        protected void onContentsChanged(int slot) {
            setChanged();
        }
    };

    public final ElectricFurnaceEnergyStorage energyStorage = new
ElectricFurnaceEnergyStorage(40, 40);

    private LazyOptional<IItemHandler> lazyItemHandler = LazyOptional.empty();
    private LazyOptional<IEnergyStorage> lazyEnergyHandler = LazyOptional.empty();

    protected final ContainerData data;
    public int progress = 0;
    public int maxProgress = 100; // Піч може працювати швидше або повільніше за
дробарку
    private static final int ENERGY_REQ = 20;

    public ElectricFurnaceBlockEntity(BlockPos pPos, BlockState pState) {
        super(ModBlockEntityInit.ELECTRIC_FURNACE.get(), pPos, pState);
        this.data = new ContainerData() {
            @Override
            public int get(int pIndex) {
                return switch (pIndex) {
                    case 0 -> ElectricFurnaceBlockEntity.this.progress;
                    case 1 -> ElectricFurnaceBlockEntity.this.maxProgress;
                    case 2 -> ElectricFurnaceBlockEntity.this.energyStorage.getEnergyStored()
>= ENERGY_REQ ? 1 : 0;
                    default -> 0;
                };
            }
            @Override
            public void set(int pIndex, int pValue) {
                switch (pIndex) {
```

```
                case 0 -> ElectricFurnaceBlockEntity.this.progress = pValue;
                case 1 -> ElectricFurnaceBlockEntity.this.maxProgress = pValue;
            }
        }
        @Override
        public int getCount() {
            return 3;
        }
    };
}

@Override
public <T> LazyOptional<T> getCapability(Capability<T> cap, Direction side) {
    if (cap == ForgeCapabilities.ENERGY) return lazyEnergyHandler.cast();
    if (cap == ForgeCapabilities.ITEM_HANDLER) return lazyItemHandler.cast();
    return super.getCapability(cap, side);
}

@Override
public void onLoad() {
    super.onLoad();
    lazyItemHandler = LazyOptional.of(() -> itemHandler);
    lazyEnergyHandler = LazyOptional.of(() -> energyStorage);
}

@Override
public void invalidateCaps() {
    super.invalidateCaps();
    lazyItemHandler.invalidate();
    lazyEnergyHandler.invalidate();
}

@Override
protected void saveAdditional(CompoundTag pTag) {
    pTag.put("inventory", itemHandler.serializeNBT());
    pTag.putInt("energy", energyStorage.getEnergyStored());
    pTag.putInt("progress", progress);
    super.saveAdditional(pTag);
}

@Override
public void load(CompoundTag pTag) {
    super.load(pTag);
    itemHandler.deserializeNBT(pTag.getCompound("inventory"));
    energyStorage.receiveEnergy(pTag.getInt("energy"), false);
    progress = pTag.getInt("progress");
}

public static void tick(Level pLevel, BlockPos pPos, BlockState pState,
ElectricFurnaceBlockEntity pBlockEntity) {
    if (pLevel.isClientSide()) return;
```

```
        Optional<SmeltingRecipe> recipe = pBlockEntity.getCurrentRecipe();
        // Перевіряємо ТІЛЬКИ наявність рецепту та місця для результату
        boolean hasRecipe = recipe.isPresent() && pBlockEntity.canInsertAmount(recipe.get()) && pBlockEntity.canInsertItem(recipe.get());

        if (hasRecipe) {
            // Якщо рецепт є, перевіряємо енергію
            boolean hasEnergy = pBlockEntity.energyStorage.getEnergyStored() >= ENERGY_REQ;

            if (hasEnergy) {
                pBlockEntity.energyStorage.consumeEnergy(ENERGY_REQ);
                pBlockEntity.progress++;

                if (pBlockEntity.progress >= pBlockEntity.maxProgress) {
                    pBlockEntity.craftItem(recipe.get());
                    pBlockEntity.progress = 0;
                }
            }

            setChanged(pLevel, pPos, pState);
        } else {
            // Скидаємо прогрес НА НУЛЬ тільки якщо гравець забрав сировину з печі
            pBlockEntity.progress = 0;
            setChanged(pLevel, pPos, pState);
        }
    }

    private Optional<SmeltingRecipe> getCurrentRecipe() {
        SimpleContainer inventory = new SimpleContainer(this.itemHandler.getSlots());
        for(int i = 0; i < itemHandler.getSlots(); i++) {
            inventory.setItem(i, this.itemHandler.getStackInSlot(i));
        }

        // 1. Спочатку шукаємо ексклюзивні рецепти Електропечі (наприклад, пил -> злиток)
        Optional<ElectricFurnaceRecipe> electricRecipe = this.level.getRecipeManager().getRecipeFor(ModRecipes.ELECTRIC_SMELTING_TYPE.get(), inventory, this.level);

        if (electricRecipe.isPresent()) {
            // Якщо знайшли ексклюзивний рецепт, віддаємо його
            // Оскільки ElectricFurnaceRecipe є спадкоємцем SmeltingRecipe, Java дозволить цю "підміну"
            return Optional.of(electricRecipe.get());
        }

        // 2. Якщо спеціального рецепту немає, шукаємо звичайні ванільні рецепти (залізо, їжа)
        return this.level.getRecipeManager().getRecipeFor(RecipeType.SMELTING, inventory, this.level);
```

```
    }

    private boolean canInsertItem(SmeltingRecipe recipe) {
        ItemStack outputStack = itemHandler.getStackInSlot(1);
        return outputStack.isEmpty() || outputStack.getItem() ==
recipe.getResultItem(level.registryAccess()).getItem();
    }

    private boolean canInsertAmount(SmeltingRecipe recipe) {
        ItemStack outputStack = itemHandler.getStackInSlot(1);
        return outputStack.getCount() +
recipe.getResultItem(level.registryAccess()).getCount() <=
outputStack.getMaxStackSize();
    }

    private void craftItem(SmeltingRecipe recipe) {
        ItemStack result = recipe.getResultItem(level.registryAccess());
        itemHandler.extractItem(0, 1, false);
        itemHandler.setStackInSlot(1, new ItemStack(result.getItem(),
itemHandler.getStackInSlot(1).getCount() + result.getCount()));
    }

    @Override
    public Component getDisplayName() {
        return Component.translatable("block.industrialcore.electric_furnace");
    }

    @Override
    public AbstractContainerMenu createMenu(int pContainerId, Inventory
pPlayerInventory, Player pPlayer) {
        return new ElectricFurnaceMenu(pContainerId, pPlayerInventory, this, this.data);
    }

    public static class ElectricFurnaceEnergyStorage extends EnergyStorage {
        public ElectricFurnaceEnergyStorage(int capacity, int maxReceive) {
            super(capacity, maxReceive, 0);
        }
        public void consumeEnergy(int amount) {
            this.energy = Math.max(0, this.energy - amount);
        }
    }

}
```

Б.5 Файл WindmillRenderer

```
package com.xqueryy.industrialcore.client.renderer;

import com.mojang.blaze3d.vertex.PoseStack;
import com.mojang.blaze3d.vertex.VertexConsumer;
import com.mojang.math.Axis;
```

```
import com.xqueryy.industrialcore.IndustrialCore;
import com.xqueryy.industrialcore.block.WindmillBlock;
import com.xqueryy.industrialcore.blockentity.WindmillBlockEntity;
import net.minecraft.client.renderer.MultiBufferSource;
import net.minecraft.client.renderer.RenderType;
import net.minecraft.client.renderer.blockentity.BlockEntityRenderer;
import net.minecraft.client.renderer.blockentity.BlockEntityRendererProvider;
import net.minecraft.core.Direction;
import net.minecraft.resources.ResourceLocation;
import net.minecraft.util.Mth;
import org.joml.Matrix3f;
import net.minecraft.client.renderer.LevelRenderer;
import org.joml.Matrix4f;

public class WindmillRenderer implements BlockEntityRenderer<WindmillBlockEntity> {
   private static final ResourceLocation BLADES_TEXTURE = new ResourceLocation(IndustrialCore.MODID, "textures/block/windmill_blades.png");

   public WindmillRenderer(BlockEntityRendererProvider.Context context) {
   }

   @Override
   public void render(WindmillBlockEntity blockEntity, float partialTick, PoseStack poseStack, MultiBufferSource bufferSource, int packedLight, int packedOverlay) {
      // Отримуємо напрямок блоку, щоб лопаті були спереду
      Direction direction = blockEntity.getBlockState().getValue(WindmillBlock.FACING);

      poseStack.pushPose(); // Починаємо трансформації

      // 1. Переміщуємося в центр блоку
      poseStack.translate(0.5, 0.5, 0.5);

      // 2. Обертаємо всю конструкцію туди, куди дивиться вітряк
      poseStack.mulPose(Axis.YP.rotationDegrees(-direction.toYRot()));

      // 3. Зміщуємося трохи вперед (щоб лопаті не врізалися в сам блок)
      poseStack.translate(0, 0, 0.55);

      // 4. ПЛАВНЕ ОБЕРТАННЯ! (Lerp згладжує рух між кадрами)
      float angle = Mth.lerp(partialTick, blockEntity.prevRotation, blockEntity.rotation);
      poseStack.mulPose(Axis.ZP.rotationDegrees(angle));

      // 5. Малюємо площину з лопатями
      VertexConsumer vertexConsumer = bufferSource.getBuffer(RenderType.entityCutout(BLADES_TEXTURE));
      Matrix4f matrix4f = poseStack.last().pose();

      // Отримуємо матрицю нормалей, яка враховує наші повороти лопатей
      Matrix3f matrix3f = poseStack.last().normal();
```

```
        // Беремо рівень світла не з центру блоку, а з вулиці (перед вітряком)
        int actualLight = LevelRenderer.getLightColor(blockEntity.getLevel(),
blockEntity.getBlockPos().relative(direction));

        float size = 5.0f;
        float thickness = 0.01f;

        // --- ПЕРЕДНЯ СТОРОНА ---
        vertexConsumer.vertex(matrix4f, -size, -size, thickness).color(255, 255, 255,
255).uv(0.0F, 1.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, 1).endVertex();
        vertexConsumer.vertex(matrix4f, size, -size, thickness).color(255, 255, 255,
255).uv(1.0F, 1.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, 1).endVertex();
        vertexConsumer.vertex(matrix4f, size, size, thickness).color(255, 255, 255,
255).uv(1.0F, 0.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, 1).endVertex();
        vertexConsumer.vertex(matrix4f, -size, size, thickness).color(255, 255, 255,
255).uv(0.0F, 0.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, 1).endVertex();

        // --- ЗАДНЯ СТОРОНА ---
        vertexConsumer.vertex(matrix4f, -size, size, -thickness).color(255, 255, 255,
255).uv(0.0F, 0.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, -1).endVertex();
        vertexConsumer.vertex(matrix4f, size, size, -thickness).color(255, 255, 255,
255).uv(1.0F, 0.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, -1).endVertex();
        vertexConsumer.vertex(matrix4f, size, -size, -thickness).color(255, 255, 255,
255).uv(1.0F, 1.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, -1).endVertex();
        vertexConsumer.vertex(matrix4f, -size, -size, -thickness).color(255, 255, 255,
255).uv(0.0F, 1.0F).overlayCoords(packedOverlay).uv2(actualLight).normal(matrix3f,
0, 0, -1).endVertex();

        poseStack.popPose();
    }
}
```

Б.6 Файл ModEnergyStorage

```
package com.xqueryy.industrialcore.energy;

import net.minecraftforge.energy.EnergyStorage;

public abstract class ModEnergyStorage extends EnergyStorage {

    public ModEnergyStorage(int capacity, int maxTransfer) {
        super(capacity, maxTransfer, maxTransfer);
    }
```

```
    public ModEnergyStorage(int capacity, int maxReceive, int maxExtract) {
        super(capacity, maxReceive, maxExtract);
    }

    public ModEnergyStorage(int capacity, int maxReceive, int maxExtract, int energy) {
        super(capacity, maxReceive, maxExtract, energy);
    }

    @Override
    public int receiveEnergy(int maxReceive, boolean simulate) {
        int receivedEnergy = super.receiveEnergy(maxReceive, simulate);
        if (receivedEnergy > 0 && !simulate) {
            onEnergyChanged();
        }
        return receivedEnergy;
    }

    @Override
    public int extractEnergy(int maxExtract, boolean simulate) {
        int extractedEnergy = super.extractEnergy(maxExtract, simulate);
        if (extractedEnergy > 0 && !simulate) {
            onEnergyChanged();
        }
        return extractedEnergy;
    }

    public void energy(int energy) {
        this.energy = energy;
        onEnergyChanged();
    }

    public void setEnergy(int energy) {
        this.energy = Math.max(0, Math.min(energy, this.capacity));
        onEnergyChanged();
    }

    public abstract void onEnergyChanged();
}
```

Б.7 Файл ModBlockEntityInit

```
package com.xqueryy.industrialcore.init;

import com.xqueryy.industrialcore.IndustrialCore;
import com.xqueryy.industrialcore.blockentity.*; // Переконайся, що імпорти правильні
import net.minecraft.world.level.block.entity.BlockEntityType;
import net.minecraftforge.registries.DeferredRegister;
import net.minecraftforge.registries.ForgeRegistries;
import net.minecraftforge.registries.RegistryObject;
```

```
public class ModBlockEntityInit {
   public static final DeferredRegister<BlockEntityType<?>> REGISTER =
DeferredRegister.create(ForgeRegistries.BLOCK_ENTITY_TYPES,
IndustrialCore.MODID);

   public static final RegistryObject<BlockEntityType<ThermalGeneratorBlockEntity>>
THERMAL_GENERATOR = REGISTER.register("thermal_generator",
         () -> BlockEntityType.Builder.of(ThermalGeneratorBlockEntity::new,
ModBlockInit.THERMAL_GENERATOR.get()).build(null));

   public static final RegistryObject<BlockEntityType<CableBlockEntity>> CABLE =
REGISTER.register("cable", () ->
               BlockEntityType.Builder.of(CableBlockEntity::new,
                     ModBlockInit.COPPER_CABLE.get(),
                     ModBlockInit.INSULATED_COPPER_CABLE.get()
               ).build(null));

   public static final RegistryObject<BlockEntityType<EnergyStorageBlockEntity>>
ENERGY_STORAGE = REGISTER.register("energy_storage",
         () -> BlockEntityType.Builder.of(EnergyStorageBlockEntity::new,
ModBlockInit.ENERGY_STORAGE.get()).build(null));

   public static final RegistryObject<BlockEntityType<WindmillBlockEntity>>
WINDMILL = REGISTER.register("windmill",
         () -> BlockEntityType.Builder.of(WindmillBlockEntity::new,
ModBlockInit.WINDMILL.get()).build(null));

   public static final RegistryObject<BlockEntityType<MaceratorBlockEntity>>
MACERATOR = REGISTER.register("macerator",
         () -> BlockEntityType.Builder.of(MaceratorBlockEntity::new,
ModBlockInit.MACERATOR.get()).build(null));

   public static final RegistryObject<BlockEntityType<ElectricFurnaceBlockEntity>>
ELECTRIC_FURNACE = REGISTER.register("electric_furnace",
         () -> BlockEntityType.Builder.of(ElectricFurnaceBlockEntity::new,
ModBlockInit.ELECTRIC_FURNACE.get()).build(null));

   public static final RegistryObject<BlockEntityType<MetalFormerBlockEntity>>
METAL_FORMER = REGISTER.register("metal_former",
         () -> BlockEntityType.Builder.of(MetalFormerBlockEntity::new,
ModBlockInit.METAL_FORMER.get()).build(null));

}
```

Б.8 Файл ModBlockInit

```
package com.xqueryy.industrialcore.init;

import com.xqueryy.industrialcore.IndustrialCore;
import com.xqueryy.industrialcore.block.*;
import net.minecraft.world.item.BlockItem;
```

```
import net.minecraft.world.item.Item;
import net.minecraft.world.level.block.Block;
import net.minecraft.world.level.block.Blocks;
import net.minecraft.world.level.block.state.BlockBehaviour;
import net.minecraft.world.level.material.MapColor;
import net.minecraftforge.registries.DeferredRegister;
import net.minecraftforge.registries.ForgeRegistries;
import net.minecraftforge.registries.RegistryObject;

import java.util.function.Supplier;

public class ModBlockInit {
   public static final DeferredRegister<Block> REGISTER = DeferredRegister.create(ForgeRegistries.BLOCKS, IndustrialCore.MODID);

   public static final RegistryObject<Block> THERMAL_GENERATOR = registerBlock("thermal_generator",
         () -> new ThermalGeneratorBlock(BlockBehaviour.Properties.of()
               .mapColor(MapColor.METAL)
               .strength(3.5f)
               .requiresCorrectToolForDrops()
               // Якщо LIT = true, рівень світла 13, інакше 0
               .lightLevel(state -> state.getValue(ThermalGeneratorBlock.LIT) ? 13 : 0)));

   public static final RegistryObject<Block> COPPER_CABLE = registerBlock("copper_cable",
         () -> new CableBlock(BlockBehaviour.Properties.of()
               .mapColor(MapColor.COLOR_ORANGE)
               .strength(1.0f)
               .requiresCorrectToolForDrops()
               // Кабель не блокує світло і не є повним кубом
               .noOcclusion()));

   public static final RegistryObject<Block> INSULATED_COPPER_CABLE = registerBlock("insulated_copper_cable",
         () -> new InsulatedCableBlock(BlockBehaviour.Properties.of()
               .mapColor(MapColor.COLOR_BLACK)
               .strength(1.5f)
               .requiresCorrectToolForDrops()
               .noOcclusion()));

   public static final RegistryObject<Block> ENERGY_STORAGE = registerBlock("energy_storage",
         () -> new EnergyStorageBlock(BlockBehaviour.Properties.of()
               .mapColor(MapColor.METAL)
               .strength(4.0f)
               .requiresCorrectToolForDrops()));

   public static final RegistryObject<Block> WINDMILL = registerBlock("windmill",
         () -> new WindmillBlock(BlockBehaviour.Properties.of()
               .mapColor(MapColor.WOOD)
               .strength(2.0f)
```

```
                .requiresCorrectToolForDrops()));

    public static final RegistryObject<Block> TIN_ORE = registerBlock("tin_ore",
        () -> new Block(BlockBehaviour.Properties.of()
                .mapColor(MapColor.STONE)
                .strength(3.0f, 3.0f)
                .requiresCorrectToolForDrops())); // Потребує кайла

    public static final RegistryObject<Block> LEAD_ORE = registerBlock("lead_ore",
        () -> new Block(BlockBehaviour.Properties.of()
                .mapColor(MapColor.STONE)
                .strength(3.0f, 3.0f)
                .requiresCorrectToolForDrops()));

    public static final RegistryObject<Block> MACERATOR =
registerBlock("macerator",
        () -> new MaceratorBlock(BlockBehaviour.Properties.of()
                .mapColor(MapColor.METAL)
                .strength(3.5f)
                .requiresCorrectToolForDrops()));

    public static final RegistryObject<Block> ELECTRIC_FURNACE =
registerBlock("electric_furnace",
        () -> new ElectricFurnaceBlock(BlockBehaviour.Properties.of()
                .mapColor(MapColor.METAL)
                .strength(3.5f)
                .requiresCorrectToolForDrops()));

    public static final RegistryObject<Block> METAL_FORMER =
registerBlock("metal_former",
        () -> new MetalFormerBlock(BlockBehaviour.Properties.of()
                .mapColor(MapColor.METAL)
                .strength(3.5f)
                .requiresCorrectToolForDrops()));

    public static final RegistryObject<Block> FLAX_CROP = registerBlock("flax_crop",
        () -> new
FlaxCropBlock(BlockBehaviour.Properties.copy(Blocks.WHEAT).noOcclusion().noCo
llission()));


    private static <T extends Block> RegistryObject<T> registerBlock (String name,
Supplier<T> properties){
        RegistryObject<T> block =  REGISTER.register(name, properties);
        registerItem(name, block);
        return block;
    }

    private static <T extends Block> RegistryObject<BlockItem> registerItem (String
name, RegistryObject<T> block){
        return ModItemInit.REGISTER.register(name, () -> new BlockItem(block.get(),
new Item.Properties()));
```

```
    }

}
```

Б.9 Файл ModItemInit

```
package com.xqueryy.industrialcore.init;

import com.xqueryy.industrialcore.IndustrialCore;
import com.xqueryy.industrialcore.item.BronzeDrillItem;
import net.minecraft.world.food.FoodProperties;
import net.minecraft.world.item.BlockItem;
import net.minecraft.world.item.Item;
import net.minecraft.world.item.ItemNameBlockItem;
import net.minecraftforge.registries.DeferredRegister;
import net.minecraftforge.registries.ForgeRegistries;
import net.minecraftforge.registries.RegistryObject;

public class ModItemInit {
    public static final DeferredRegister<Item> REGISTER =
DeferredRegister.create(ForgeRegistries.ITEMS, IndustrialCore.MODID);

    public static final RegistryObject<Item> RAW_TIN = REGISTER.register("raw_tin",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> TIN_DUST =
REGISTER.register("tin_dust",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> TIN_INGOT =
REGISTER.register("tin_ingot",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> RAW_LEAD =
REGISTER.register("raw_lead",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> LEAD_DUST =
REGISTER.register("lead_dust",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> LEAD_INGOT =
REGISTER.register("lead_ingot",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> BRONZE_PLATE =
REGISTER.register("bronze_plate",
            () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> FLAX_STRAW =
REGISTER.register("flax_straw",
```

```
        () -> new Item(new Item.Properties()));

    public static final RegistryObject<Item> FLAX_FIBER =
REGISTER.register("flax_fiber",
        () -> new Item(new Item.Properties()));

    // Насіння прив'язується до блоку FLAX_CROP
    public static final RegistryObject<Item> FLAX_SEEDS =
REGISTER.register("flax_seeds",
        () -> new ItemNameBlockItem(ModBlockInit.FLAX_CROP.get(), new
Item.Properties()));

    public static final RegistryObject<Item> BRONZE_DRILL =
REGISTER.register("bronze_drill",
        () -> new BronzeDrillItem(new Item.Properties().durability(2500)));

}
```

Б.10 Файл ModMenuTypesInit

```
package com.xqueryy.industrialcore.init;

import com.xqueryy.industrialcore.IndustrialCore;
import com.xqueryy.industrialcore.screen.*;
import net.minecraft.world.inventory.MenuType;
import net.minecraftforge.common.extensions.IForgeMenuType;
import net.minecraftforge.registries.DeferredRegister;
import net.minecraftforge.registries.ForgeRegistries;
import net.minecraftforge.registries.RegistryObject;

public class ModMenuTypesInit {
    public static final DeferredRegister<MenuType<?>> REGISTER =
DeferredRegister.create(ForgeRegistries.MENU_TYPES, IndustrialCore.MODID);

    public static final RegistryObject<MenuType<ThermalGeneratorMenu>>
THERMAL_GENERATOR_MENU = REGISTER.register("thermal_generator_menu",
        () -> IForgeMenuType.create(ThermalGeneratorMenu::new));

    public static final RegistryObject<MenuType<EnergyStorageMenu>>
ENERGY_STORAGE_MENU = REGISTER.register("energy_storage_menu",
        () -> IForgeMenuType.create(EnergyStorageMenu::new));

    public static final RegistryObject<MenuType<MaceratorMenu>>
MACERATOR_MENU = REGISTER.register("macerator_menu",
        () -> IForgeMenuType.create(MaceratorMenu::new));

    public static final RegistryObject<MenuType<ElectricFurnaceMenu>>
ELECTRIC_FURNACE_MENU = REGISTER.register("electric_furnace_menu",
        () -> IForgeMenuType.create(ElectricFurnaceMenu::new));
```

```
    public static final RegistryObject<MenuType<MetalFormerMenu>>
METAL_FORMER_MENU = REGISTER.register("metal_former_menu",
            () -> IForgeMenuType.create(MetalFormerMenu::new));


}
```

Б.11 Файл ModRecipes

```
package com.xqueryy.industrialcore.init;

import com.xqueryy.industrialcore.recipe.ElectricFurnaceRecipe;
import com.xqueryy.industrialcore.recipe.ElectricFurnaceRecipeSerializer;
import net.minecraft.core.registries.Registries;
import net.minecraft.world.item.crafting.RecipeSerializer;
import net.minecraft.world.item.crafting.RecipeType;
import net.minecraft.world.item.crafting.SimpleCookingSerializer;
import net.minecraftforge.registries.DeferredRegister;
import net.minecraftforge.registries.ForgeRegistries;
import net.minecraftforge.registries.RegistryObject;

public class ModRecipes {
    public static final DeferredRegister<RecipeSerializer<?>> SERIALIZERS =
            DeferredRegister.create(ForgeRegistries.RECIPE_SERIALIZERS,
"industrialcore");

    public static final DeferredRegister<RecipeType<?>> TYPES =
            DeferredRegister.create(Registries.RECIPE_TYPE, "industrialcore");

    public static final RegistryObject<RecipeType<ElectricFurnaceRecipe>>
ELECTRIC_SMELTING_TYPE =
            TYPES.register("electric_furnace_smelting", () -> new
RecipeType<ElectricFurnaceRecipe>() {
                @Override
                public String toString() {
                    return "electric_furnace_smelting";
                }
            });

    public static final RegistryObject<RecipeSerializer<ElectricFurnaceRecipe>>
ELECTRIC_SMELTING_SERIALIZER =
            SERIALIZERS.register("electric_furnace_smelting",
ElectricFurnaceRecipeSerializer::new);
}
```

Б.12 Файл ModTabInit

```
package com.xqueryy.industrialcore.init;

import com.xqueryy.industrialcore.IndustrialCore;
```

```
import net.minecraft.core.registries.Registries;
import net.minecraft.network.chat.Component;
import net.minecraft.world.item.CreativeModeTab;
import net.minecraft.world.item.CreativeModeTabs;
import net.minecraftforge.registries.DeferredRegister;
import net.minecraftforge.registries.RegistryObject;

public class ModTabInit {
   public static final DeferredRegister<CreativeModeTab> REGISTER =
DeferredRegister.create(Registries.CREATIVE_MODE_TAB,
IndustrialCore.MODID);

   public static final RegistryObject<CreativeModeTab> EXAMPLE_TAB =
REGISTER.register("example_tab", () -> CreativeModeTab.builder()
         .withTabsBefore(CreativeModeTabs.COMBAT)
         .icon(()                                                    ->
ModBlockInit.ELECTRIC_FURNACE.get().asItem().getDefaultInstance())
         .title(Component.translatable("IndustrialCore by Antonio"))
         .displayItems((parameters, output) -> {
            output.accept(ModBlockInit.THERMAL_GENERATOR.get());

            output.accept(ModBlockInit.COPPER_CABLE.get());
            output.accept(ModBlockInit.INSULATED_COPPER_CABLE.get());

            output.accept(ModBlockInit.ENERGY_STORAGE.get());
            output.accept(ModBlockInit.WINDMILL.get());
            output.accept(ModBlockInit.MACERATOR.get());
            output.accept(ModBlockInit.ELECTRIC_FURNACE.get());
            output.accept(ModBlockInit.METAL_FORMER.get());

            // Предмети
            output.accept(ModItemInit.RAW_TIN.get());
            output.accept(ModItemInit.TIN_DUST.get());
            output.accept(ModItemInit.TIN_INGOT.get());
            output.accept(ModItemInit.RAW_LEAD.get());
            output.accept(ModItemInit.LEAD_DUST.get());
            output.accept(ModItemInit.LEAD_INGOT.get());
            output.accept(ModItemInit.BRONZE_PLATE.get());
            output.accept(ModItemInit.FLAX_FIBER.get());
            output.accept(ModItemInit.FLAX_STRAW.get());
            output.accept(ModItemInit.FLAX_SEEDS.get());

            output.accept(ModItemInit.BRONZE_DRILL.get());


            // Блоки
            output.accept(ModBlockInit.TIN_ORE.get());
            output.accept(ModBlockInit.LEAD_ORE.get());

         }).build());
}
```

Б.13 Файл BronzeDrillItem

```
package com.xqueryy.industrialcore.item;

import net.minecraft.core.BlockPos;
import net.minecraft.core.Direction;
import net.minecraft.nbt.CompoundTag;
import net.minecraft.server.level.ServerLevel;
import net.minecraft.world.entity.EquipmentSlot;
import net.minecraft.world.entity.LivingEntity;
import net.minecraft.world.entity.player.Player;
import net.minecraft.world.item.ItemStack;
import net.minecraft.world.item.PickaxeItem;
import net.minecraft.world.item.Tiers;
import net.minecraft.world.level.ClipContext;
import net.minecraft.world.level.Level;
import net.minecraft.world.level.block.Block;
import net.minecraft.world.level.block.state.BlockState;
import net.minecraft.world.phys.BlockHitResult;
import net.minecraft.world.phys.HitResult;
import net.minecraftforge.common.capabilities.Capability;
import net.minecraftforge.common.capabilities.ForgeCapabilities;
import net.minecraftforge.common.capabilities.ICapabilityProvider;
import net.minecraftforge.common.util.LazyOptional;
import net.minecraftforge.energy.IEnergyStorage;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

public class BronzeDrillItem extends PickaxeItem {

   public BronzeDrillItem(Properties pProperties) {
      super(Tiers.DIAMOND, 1, -2.8F, pProperties);
   }

   @Override
   public ICapabilityProvider initCapabilities(ItemStack stack, @Nullable
CompoundTag nbt) {
      return new ICapabilityProvider() {
         private final IEnergyStorage storage = new IEnergyStorage() {
            private final int capacity = 40000; // 40k FE
            private final int maxTransfer = 1000;

            @Override
            public int receiveEnergy(int maxReceive, boolean simulate) {
               int energy = getEnergyStored();
               int energyReceived = Math.min(capacity - energy, Math.min(maxTransfer,
maxReceive));
               if (!simulate) stack.getOrCreateTag().putInt("energy", energy +
energyReceived);
               return energyReceived;
            }
```

```
            @Override
            public int extractEnergy(int maxExtract, boolean simulate) {
               int energy = getEnergyStored();
               int energyExtracted = Math.min(energy, Math.min(maxTransfer, maxExtract));
               if (!simulate) stack.getOrCreateTag().putInt("energy", energy - energyExtracted);
               return energyExtracted;
            }

            @Override
            public int getEnergyStored() { return stack.getOrCreateTag().getInt("energy"); }
            @Override
            public int getMaxEnergyStored() { return capacity; }
            @Override
            public boolean canExtract() { return false; } // Бур не віддає енергію
            @Override
            public boolean canReceive() { return true; }  // Але може заряджатися
         };
         private final LazyOptional<IEnergyStorage> lazyOptional = LazyOptional.of(() -> storage);

         @Override
         public <T> @NotNull LazyOptional<T> getCapability(@NotNull Capability<T> cap, @Nullable Direction side) {
            if (cap == ForgeCapabilities.ENERGY) return lazyOptional.cast();
            return LazyOptional.empty();
         }
      };
   }

   // Допоміжний метод для швидкого отримання заряду
   public int getEnergy(ItemStack stack) {
      return stack.getOrCreateTag().getInt("energy");
   }

   @Override
   public float getDestroySpeed(ItemStack pStack, BlockState pState) {
      if (getEnergy(pStack) < 100) return 1.0f; // Розряджений бур копає як рука
      float defaultSpeed = super.getDestroySpeed(pStack, pState);
      return defaultSpeed > 1.0f ? 34.0f : defaultSpeed;
   }

   @Override
   public boolean mineBlock(ItemStack pStack, Level pLevel, BlockState pState, BlockPos pPos, LivingEntity pEntityLiving) {
      if (pLevel.isClientSide || !(pEntityLiving instanceof Player player)) {
         return super.mineBlock(pStack, pLevel, pState, pPos, pEntityLiving);
      }
```

```
        // Перевіряємо, чи вистачає енергії хоча б на 1 блок
        if (getEnergy(pStack) < 100) return false;

        HitResult raytraceresult = getPlayerPOVHitResult(pLevel, player,
ClipContext.Fluid.NONE);
        if (raytraceresult.getType() == HitResult.Type.BLOCK) {
            BlockHitResult blockHitResult = (BlockHitResult) raytraceresult;
            Direction sideHit = blockHitResult.getDirection();

            int xRange = sideHit.getStepX() == 0 ? 1 : 0;
            int yRange = sideHit.getStepY() == 0 ? 1 : 0;
            int zRange = sideHit.getStepZ() == 0 ? 1 : 0;

            for (int x = -xRange; x <= xRange; x++) {
                for (int y = -yRange; y <= yRange; y++) {
                    for (int z = -zRange; z <= zRange; z++) {
                        BlockPos targetPos = pPos.offset(x, y, z);
                        if (targetPos.equals(pPos)) continue;

                        BlockState targetState = pLevel.getBlockState(targetPos);

                        if (!targetState.isAir() && targetState.getDestroySpeed(pLevel,
targetPos) >= 0
                                && pStack.isCorrectToolForDrops(targetState)) {

                            // Знову перевіряємо енергію всередині циклу (раптом вона
закінчилася на 5-му блоці)
                            if (getEnergy(pStack) >= 100) {
                                if (pLevel instanceof ServerLevel serverLevel) {
                                    Block.dropResources(targetState, serverLevel, targetPos,
serverLevel.getBlockEntity(targetPos), player, pStack);
                                    serverLevel.destroyBlock(targetPos, false);

                                    // Віднімаємо 100 енергії за зламаний сусідній блок

pStack.getCapability(ForgeCapabilities.ENERGY).ifPresent(energy ->
energy.extractEnergy(100, false));
                                }
                            }
                        }
                    }
                }
            }
        }

        // Віднімаємо 100 енергії за центральний блок (по якому клікнули)
        pStack.getCapability(ForgeCapabilities.ENERGY).ifPresent(energy ->
energy.extractEnergy(100, false));
        return true;
    }

    @Override
```

```
    public boolean isBarVisible(ItemStack pStack) {
        return true;
    }

    @Override
    public int getBarWidth(ItemStack pStack) {
        return Math.round(13.0F * getEnergy(pStack) / 40000f);
    }

    @Override
    public int getBarColor(ItemStack pStack) {
        return 0x00FF00; // Зелений колір
    }
}
```

Б.14 Файл ElectricFurnaceRecipe

```
package com.xqueryy.industrialcore.recipe;

import com.xqueryy.industrialcore.init.ModRecipes;
import net.minecraft.resources.ResourceLocation;
import net.minecraft.world.item.ItemStack;
import net.minecraft.world.item.crafting.CookingBookCategory;
import net.minecraft.world.item.crafting.Ingredient;
import net.minecraft.world.item.crafting.RecipeSerializer;
import net.minecraft.world.item.crafting.RecipeType;
import net.minecraft.world.item.crafting.SmeltingRecipe;

public class ElectricFurnaceRecipe extends SmeltingRecipe {

    public ElectricFurnaceRecipe(ResourceLocation pId, String pGroup,
CookingBookCategory pCategory, Ingredient pIngredient, ItemStack pResult, float
pExperience, int pCookingTime) {
        super(pId, pGroup, pCategory, pIngredient, pResult, pExperience, pCookingTime);
    }


    @Override
    public RecipeType<?> getType() {
        return ModRecipes.ELECTRIC_SMELTING_TYPE.get();
    }

    @Override
    public RecipeSerializer<?> getSerializer() {
        return ModRecipes.ELECTRIC_SMELTING_SERIALIZER.get();
    }
}
```

Б.15 Файл ElectricFurnaceRecipeSerializer

```
package com.xqueryy.industrialcore.recipe;

import com.google.gson.JsonObject;
import net.minecraft.core.RegistryAccess;
import net.minecraft.network.FriendlyByteBuf;
import net.minecraft.resources.ResourceLocation;
import net.minecraft.world.item.crafting.RecipeSerializer;
import net.minecraft.world.item.crafting.SmeltingRecipe;
import org.jetbrains.annotations.Nullable;

public class ElectricFurnaceRecipeSerializer implements RecipeSerializer<ElectricFurnaceRecipe> {

    @Override
    public ElectricFurnaceRecipe m_6729_(ResourceLocation p_44103_, JsonObject p_44104_) {
        SmeltingRecipe recipe = RecipeSerializer.SMELTING_RECIPE.m_6729_(p_44103_, p_44104_);

        return new ElectricFurnaceRecipe(
                p_44103_,
                recipe.getGroup(),
                recipe.category(),
                recipe.getIngredients().get(0),
                recipe.getResultItem(RegistryAccess.EMPTY),
                recipe.getExperience(),
                recipe.getCookingTime()
        );
    }

    @Override
    public @Nullable ElectricFurnaceRecipe fromNetwork(ResourceLocation p_44105_, FriendlyByteBuf pBuffer) {
        SmeltingRecipe recipe = RecipeSerializer.SMELTING_RECIPE.fromNetwork(p_44105_, pBuffer);

        return new ElectricFurnaceRecipe(
                p_44105_,
                recipe.getGroup(),
                recipe.category(),
                recipe.getIngredients().get(0),
                recipe.getResultItem(RegistryAccess.EMPTY),
                recipe.getExperience(),
                recipe.getCookingTime()
        );
    }

    @Override
    public void toNetwork(FriendlyByteBuf pBuffer, ElectricFurnaceRecipe pRecipe) {
```

```
        RecipeSerializer.SMELTING_RECIPE.toNetwork(pBuffer, pRecipe);
    }
}
```

Б.16 Файл EnergyStorageMenu

```
package com.xqueryy.industrialcore.screen;

import com.xqueryy.industrialcore.blockentity.EnergyStorageBlockEntity;
import com.xqueryy.industrialcore.init.ModBlockInit;
import com.xqueryy.industrialcore.init.ModMenuTypesInit;
import net.minecraft.network.FriendlyByteBuf;
import net.minecraft.world.entity.player.Inventory;
import net.minecraft.world.entity.player.Player;
import net.minecraft.world.inventory.*;
import net.minecraft.world.item.ItemStack;
import net.minecraft.world.level.Level;
import net.minecraft.world.level.block.entity.BlockEntity;
import net.minecraftforge.common.capabilities.ForgeCapabilities;
import net.minecraftforge.items.SlotItemHandler;

public class EnergyStorageMenu extends AbstractContainerMenu {
    public final EnergyStorageBlockEntity blockEntity;
    private final Level level;
    private final ContainerData data;

    public EnergyStorageMenu(int pContainerId, Inventory inv, FriendlyByteBuf
extraData) {
        this(pContainerId,                                                    inv,
inv.player.level().getBlockEntity(extraData.readBlockPos()),                 new
SimpleContainerData(2));
    }

    public EnergyStorageMenu(int pContainerId, Inventory inv, BlockEntity entity,
ContainerData data) {
        super(ModMenuTypesInit.ENERGY_STORAGE_MENU.get(), pContainerId);
        this.blockEntity = (EnergyStorageBlockEntity) entity;
        this.level = inv.player.level();
        this.data = data;

        this.addSlot(new SlotItemHandler(this.blockEntity.itemHandler, 0, 80, 53));

        addPlayerInventory(inv);
        addPlayerHotbar(inv);

        addDataSlots(data);
    }

    public int getEnergy() { return this.data.get(0); }
    public int getMaxEnergy() { return this.data.get(1); }
```

```
    // ЛОГІКА SHIFT-CLICK
    @Override
    public ItemStack quickMoveStack(Player playerIn, int pIndex) {
        Slot sourceSlot = this.slots.get(pIndex);
        if (sourceSlot == null || !sourceSlot.hasItem()) return ItemStack.EMPTY;

        ItemStack sourceStack = sourceSlot.getItem();
        ItemStack copyOfSourceStack = sourceStack.copy();

        // Якщо клікаємо по слоту зарядки (0) -> відправляємо в інвентар (1-36)
        if (pIndex == 0) {
            if (!this.moveItemStackTo(sourceStack, 1, 37, false)) {
                return ItemStack.EMPTY;
            }
        }
        // Якщо клікаємо в інвентарі (1-36) -> відправляємо в слот зарядки (0), але ТІЛЬКИ якщо предмет приймає енергію
        else {
            if (sourceStack.getCapability(ForgeCapabilities.ENERGY).isPresent()) {
                if (!this.moveItemStackTo(sourceStack, 0, 1, false)) {
                    return ItemStack.EMPTY;
                }
            } else {
                return ItemStack.EMPTY; // Якщо це не бур, Shift-click нічого не робить
            }
        }

        if (sourceStack.getCount() == 0) sourceSlot.set(ItemStack.EMPTY);
        else sourceSlot.setChanged();

        sourceSlot.onTake(playerIn, sourceStack);
        return copyOfSourceStack;
    }

    @Override
    public boolean stillValid(Player pPlayer) {
        return stillValid(ContainerLevelAccess.create(level, blockEntity.getBlockPos()), pPlayer, ModBlockInit.ENERGY_STORAGE.get());
    }

    private void addPlayerInventory(Inventory playerInventory) {
        for (int i = 0; i < 3; ++i) {
            for (int l = 0; l < 9; ++l) {
                this.addSlot(new Slot(playerInventory, l + i * 9 + 9, 8 + l * 18, 84 + i * 18));
            }
        }
    }

    private void addPlayerHotbar(Inventory playerInventory) {
        for (int i = 0; i < 9; ++i) {
            this.addSlot(new Slot(playerInventory, i, 8 + i * 18, 142));
        }
```

```
    }
}
```

Б.17 Файл EnergyStorageScreen

```
package com.xqueryy.industrialcore.screen;

import com.mojang.blaze3d.systems.RenderSystem;
import com.xqueryy.industrialcore.IndustrialCore;
import net.minecraft.client.gui.GuiGraphics;
import net.minecraft.client.gui.screens.inventory.AbstractContainerScreen;
import net.minecraft.network.chat.Component;
import net.minecraft.resources.ResourceLocation;
import net.minecraft.world.entity.player.Inventory;

public class EnergyStorageScreen extends AbstractContainerScreen<EnergyStorageMenu> {
    private static final ResourceLocation TEXTURE = new ResourceLocation(IndustrialCore.MODID, "textures/gui/energy_storage.png");

    public EnergyStorageScreen(EnergyStorageMenu pMenu, Inventory pPlayerInventory, Component pTitle) {
        super(pMenu, pPlayerInventory, pTitle);
    }

    @Override
    protected void init() {
        super.init();
        this.inventoryLabelY = 10000;
        this.titleLabelX = (this.imageWidth - this.font.width(this.title)) / 2;
    }

    @Override
    public void render(GuiGraphics guiGraphics, int mouseX, int mouseY, float partialTick) {
        renderBackground(guiGraphics);
        super.render(guiGraphics, mouseX, mouseY, partialTick);
        renderTooltip(guiGraphics, mouseX, mouseY);

        // Тултіп для енергії. Координати: X=59, Y=33, ширина=58, висота=14
        if (isHovering(59, 33, 58, 14, mouseX, mouseY)) {
            guiGraphics.renderTooltip(this.font, Component.literal(menu.getEnergy() + " / " + menu.getMaxEnergy() + " FE"), mouseX, mouseY);
        }
    }

    @Override
    protected void renderBg(GuiGraphics guiGraphics, float pPartialTick, int pMouseX, int pMouseY) {
        RenderSystem.setShaderColor(1.0F, 1.0F, 1.0F, 1.0F);
        RenderSystem.setShaderTexture(0, TEXTURE);
```

```
        int x = (width - imageWidth) / 2;
        int y = (height - imageHeight) / 2;

        guiGraphics.blit(TEXTURE, x, y, 0, 0, imageWidth, imageHeight);

        int energyWidth = getEnergyProgress();

        guiGraphics.blit(TEXTURE, x + 59, y + 33, 176, 15, energyWidth, 14);
    }

    private int getEnergyProgress() {
        int energy = menu.getEnergy();
        int maxEnergy = menu.getMaxEnergy();
        if (maxEnergy == 0) return 0;

        return (int) (((float) energy / maxEnergy) * 58);
    }
}
```

Б.18 Файл IndustrialCore

```
package com.xqueryy.industrialcore;

import com.mojang.logging.LogUtils;
import com.xqueryy.industrialcore.client.renderer.WindmillRenderer;
import com.xqueryy.industrialcore.init.*;
import com.xqueryy.industrialcore.screen.*;
import net.minecraft.client.renderer.blockentity.BlockEntityRenderers;
import net.minecraftforge.common.MinecraftForge;
import net.minecraftforge.eventbus.api.IEventBus;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.fml.javafmlmod.FMLJavaModLoadingContext;
import net.minecraft.client.gui.screens.MenuScreens;
import net.minecraftforge.fml.event.lifecycle.FMLClientSetupEvent;
import org.slf4j.Logger;

@Mod(IndustrialCore.MODID)
public class IndustrialCore
{
    public static final String MODID = "industrialcore";
    private static final Logger LOGGER = LogUtils.getLogger();

    public IndustrialCore() {
        IEventBus modEventBus =
FMLJavaModLoadingContext.get().getModEventBus();

        ModBlockInit.REGISTER.register(modEventBus);
        ModItemInit.REGISTER.register(modEventBus);
        ModTabInit.REGISTER.register(modEventBus);
        ModBlockEntityInit.REGISTER.register(modEventBus);
        ModMenuTypesInit.REGISTER.register(modEventBus);
```

```
        ModRecipes.SERIALIZERS.register(modEventBus);
        ModRecipes.TYPES.register(modEventBus);

        modEventBus.addListener(this::clientSetup);

        MinecraftForge.EVENT_BUS.register(this);
    }

    private void clientSetup(final FMLClientSetupEvent event) {
        event.enqueueWork(() -> {

MenuScreens.register(ModMenuTypesInit.THERMAL_GENERATOR_MENU.get(),
ThermalGeneratorScreen::new);
            MenuScreens.register(ModMenuTypesInit.ENERGY_STORAGE_MENU.get(),
EnergyStorageScreen::new);
            MenuScreens.register(ModMenuTypesInit.MACERATOR_MENU.get(),
MaceratorScreen::new);

MenuScreens.register(ModMenuTypesInit.ELECTRIC_FURNACE_MENU.get(),
ElectricFurnaceScreen::new);
            MenuScreens.register(ModMenuTypesInit.METAL_FORMER_MENU.get(),
MetalFormerScreen::new);
        });

        BlockEntityRenderers.register(ModBlockEntityInit.WINDMILL.get(),
WindmillRenderer::new);
    }
}
```

ДОДАТОК В

Слайди презентації

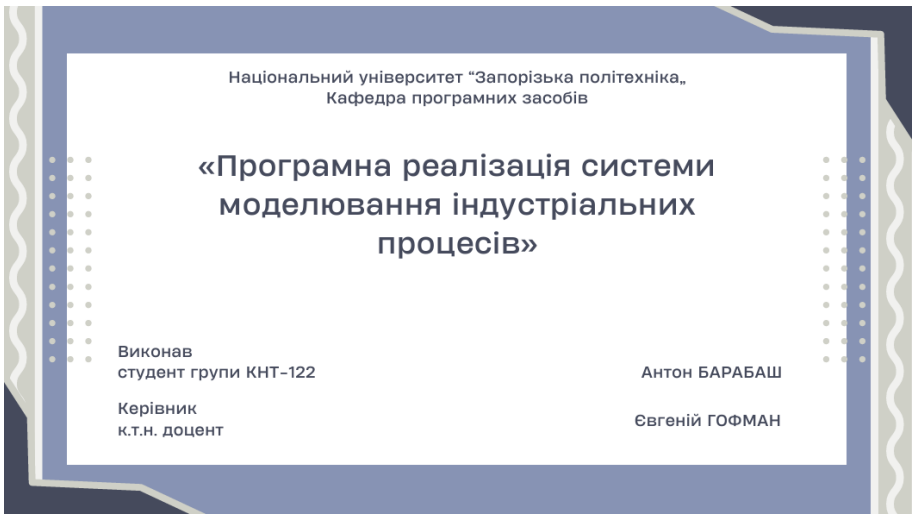

Рисунок В.1 – Слайд №1

Рисунок В.2 – Слайд №2

Назва модифікації	Енергетична система	Універсальна сумісність (FE)	Вплив на продуктивність сервера (TPS)	Поріг входження для користувача
Industrial Craft 2	Власна (EU)	Низька (потребує конвертерів)	Середній	Середній
Mekanism	Гібридна (FE, Joules)	Висока	Високий (через складну логіку труб)	Високий
Thermal Series	Стандартна (FE/RF)	Висока	Низький (відмінна оптимізація)	Низький
Create	Кінетична (SU)	Відсутня	Високий вплив на клієнт (FPS)	Високий
IndustrialCore (Розроблювана)	Стандартна (FE)	Висока	Низький (оптимізація BlockEntity)	Низький

Рисунок В.3 – Слайд №3

Рисунок В.4 – Слайд №4

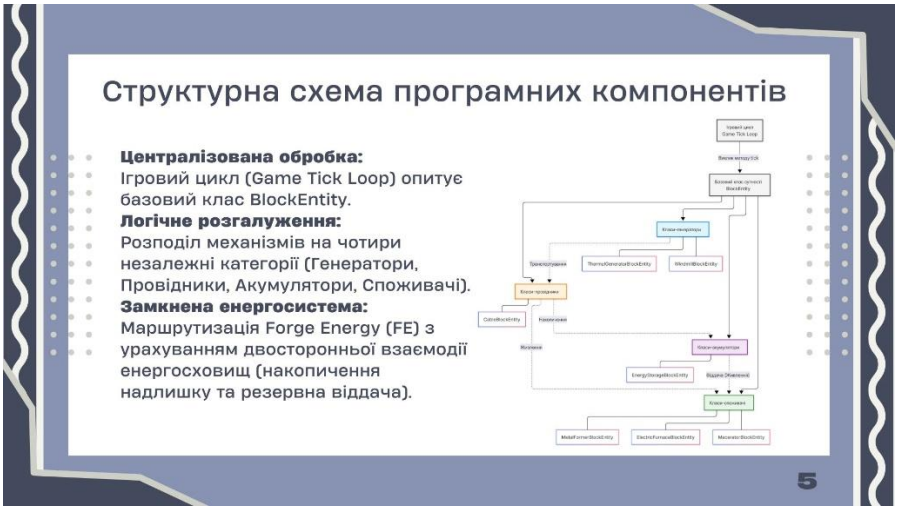

Рисунок В.5 – Слайд №5

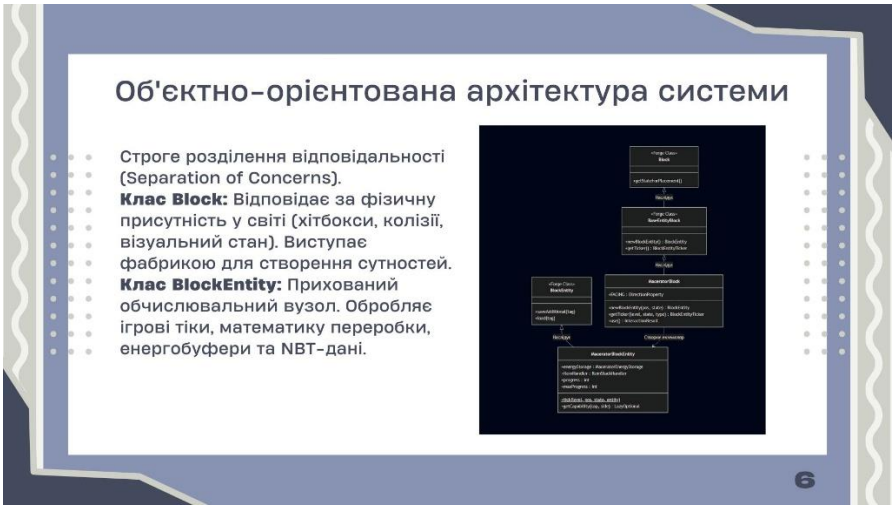

Рисунок В.6 – Слайд №6

Рисунок В.7 – Слайд №7

Рисунок В.8 – Слайд №8

Рисунок В.9 – Слайд №9

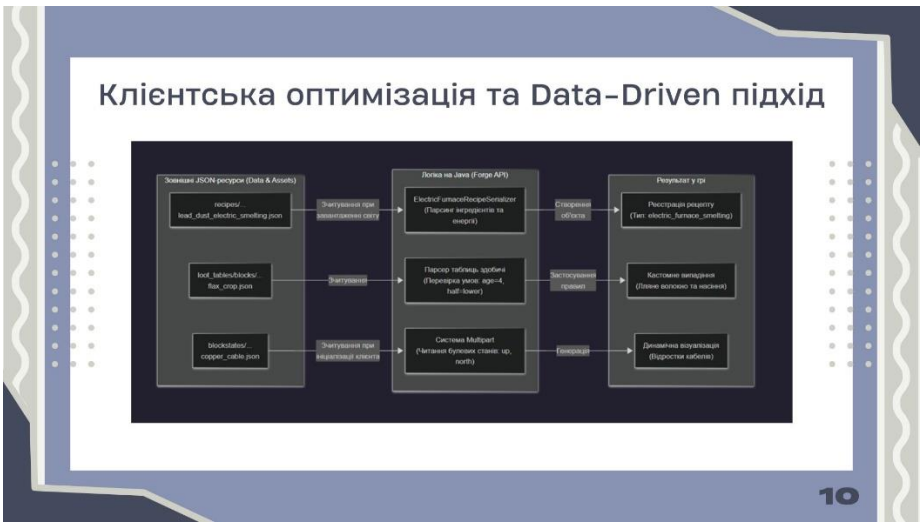

Рисунок В.10 – Слайд №10

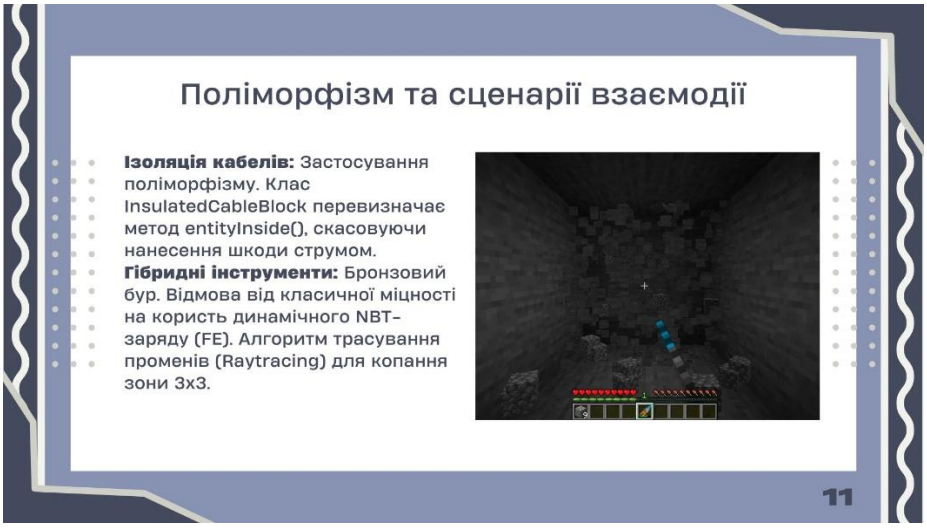

Рисунок В.11 – Слайд №11

Рисунок В.12 – Слайд №12

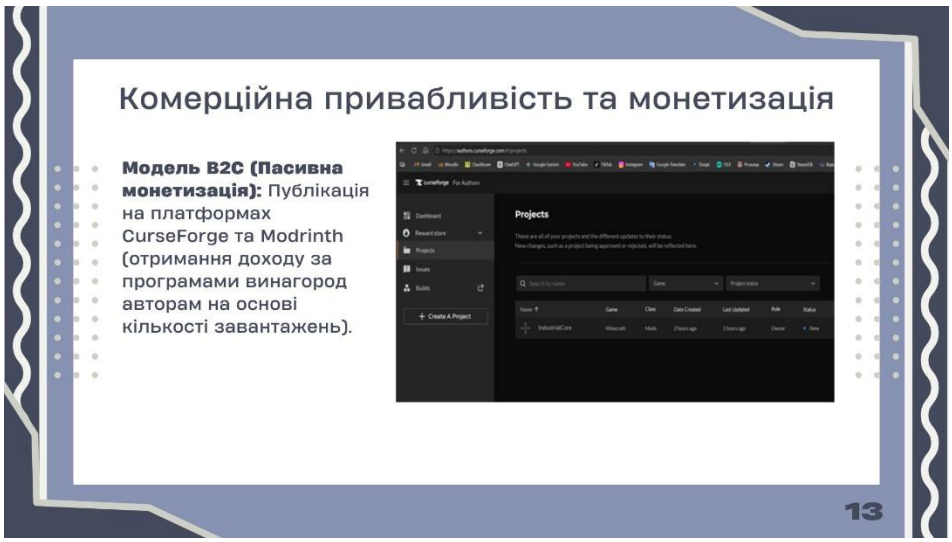

Рисунок В.13 – Слайд №13

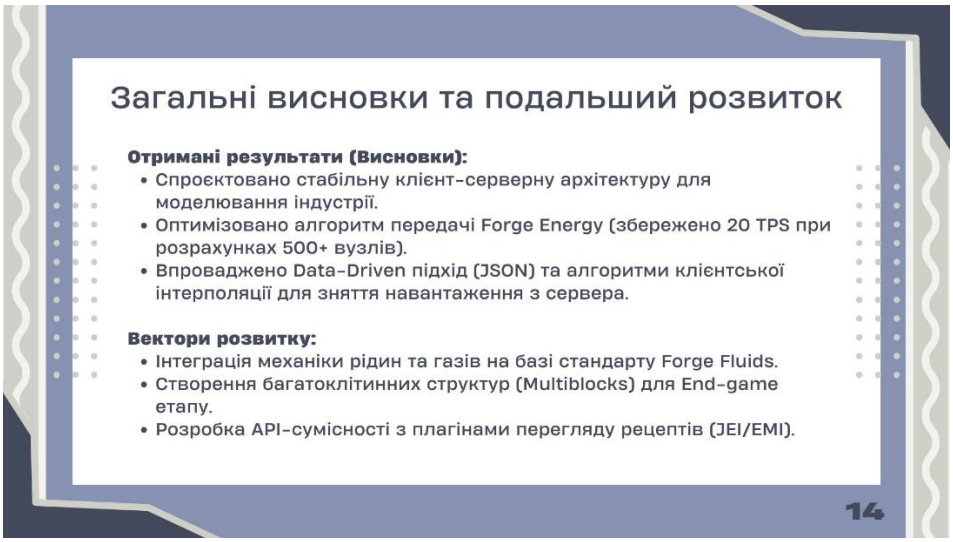

Рисунок В.14 – Слайд №14