

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 МОДИФІКАЦІЇ
ДЛЯ КОМП'ЮТЕРНОЇ ГРИ

SOFTWARE IMPLEMENTATION OF
A COMPUTER GAME MODIFICATION

Виконав(ла): студент(ка) 4 курсу, групи КНТ-121

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ГАВРИЛОВ Я.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЕОЩЕНКО С.Д.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

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
 « » 2025 року

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГАВРИЛОВА Яна Валерійовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Програмна реалізація модифікації для комп'ютерної гри. Software Implementation of a Computer Game Modification

керівник проєкту (роботи) д-р філософ., доцент ЛЕОЩЕНКО Сергій Дмитрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “28” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 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. Матеріали і методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

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	ЛЕОЩЕНКО С.Д., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Ян ГАВРИЛОВ
(Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Сергій ЛЕОЩЕНКО
(Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
90 с., 2 табл., 47 рис., 4 дод., 16 джерел.

МОДИФІКАЦІЯ, КОМП'ЮТЕРНА ГРА, MINECRAFT, ГЕЙМДЕВ,
JAVA.

Об'єкт дослідження – процес розробки модифікації для комп'ютерної гри Minecraft.

Предмет дослідження – засоби розробки модифікації для комп'ютерної гри Minecraft.

Метою роботи є дослідження засобів та безпосередньо розробка модифікації для популярної комп'ютерної гри Minecraft.

Матеріали, методи та технічні засоби: ігрове програмування, фреймворк Minecraft Forge, мова програмування Java, персональний комп'ютер з процесором з частотою не менше 2.5 ГГц під управлінням операційної системи Microsoft Windows 10 (64bit), IDE IntelliJ IDEA Community Edition 2024.2.3.

Результати. Розроблено повноцінну модифікацію для комп'ютерної гри Minecraft, котру гравці можуть встановити та запустити у своїй грі, використовуючи вбудований функціонал для імплементування ігрових модифікацій, наданий розробниками оригінальної гри та розробниками фреймворку Minecraft Forge.

Висновки. Реалізовано модифікацію для комп'ютерної гри Minecraft, котрий розширює функціонал оригінальної гри, додає до неї новий функціонал, а також значно розширює ігрові можливості гравця.

Галузь використання – розробка нових та використання вже пов'язаних з розробленою модифікацією сторонніх модифікацій.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 90 pages, 2 tables, 47 figures, 4 appendixes, 16 sources.

MODIFICATION, COMPUTER GAME, MINECRAFT, GAMEDEV, JAVA.

The object of research is the process of developing a modification for the computer game Minecraft.

The subject of the research is tools for developing a modification for the computer game Minecraft.

The aim of research is to study the tools and directly develop a modification for the popular computer game Minecraft.

Materials, methods, and technical tools: game programming, Minecraft Forge framework, Java programming language, personal computer with a processor running at a frequency of at least 2.5 GHz under the Microsoft Windows 10 (64-bit) operating system, IDE IntelliJ IDEA Community Edition 2024.2.3.

Results. A fully functional modification for the computer game Minecraft has been developed, which players can install and run in their game using the built-in functionality for implementing game modifications provided by the original game developers and the developers of the Minecraft Forge framework.

Conclusions. A modification for the computer game Minecraft has been implemented, which expands the functionality of the original game, adds new features, and significantly enhances the player's gaming capabilities.

Scope of use – the development of new and the use of already existing third-party modifications related to the created modification.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ.....	9
1 Аналіз предметної області	11
1.1 Аналіз існуючих запитів гравців на модифікації	11
1.2 Моніторинг існуючих модифікацій	12
1.3 Постановка задачі	13
1.4 Межі системи та розробки	14
1.5 Функціональні вимоги.....	15
1.6 Висновки за розділом 1	16
2 Матеріали і методи	18
2.1 Задача створення концепції та розробки модифікації	18
2.2 Методи створення концепції та розробки модифікації	20
2.3 Вибір засобів розробки.....	22
2.4 Висновки за розділом 2	23
3 Основні рішення щодо реалізації компонентів системи	24
3.1 Визначення сценаріїв взаємодії користувача з модифікацією.....	24
3.1.1 Взаємодія з предметами модифікації	26
3.1.2 Взаємодія з блоками модифікації	32
3.2 Визначення алгоритмів функціонування модифікації.....	34
3.3 Висновки за розділом 3	37
4 Експлуатація, тестування та експериментальне дослідження програми.....	38
4.1 Основне призначення програми.....	38
4.2 Вимоги до інтерфейсу	39
4.3 Умови виконання програми.....	40
4.4 Результати роботи програми	41
4.5 Висновки за розділом 4	51
Висновки	52
Перелік джерел посилання	54

Додаток А Технічне завдання	56
Додаток Б Опис програми	63
Додаток В Текст програми	68
Додаток Д Слайди презентації	79

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CTM – Connected Textures Mode;

NBT – Named Binary Tag;

NPC – Non-Player Character;

Амбієнт – атмосферні звуки, необхідні для посилення впливу на гравця;

Ванільна (vanilla) – оригінальна або немодифікована версія гри, механіка, або функціонал;

ДатаГен (DataGen) – інструмент для автоматичної генерації ресурсних і дата-файлів за ключовими параметрами, що спрощує та пришвидшує розробку, мінімізуючи ризик помилки;

Кастомна – користувацька або неоригінальна версія гри, механіка, або функціонал;

Кастомізація – можливість змінювати зовнішній вигляд;

Крафт – рецепт створення предмету;

Моб – внутрішньоігрова істота;

Мод – скорочення від модифікація;

Модпак – збірка з декількох сумісних між собою користувацьких модифікацій;

Чанк – одиниця виміру простору в Minecraft, має розмір 16 на 16 блоків та охоплює всю висоту ігрового світу в межах цієї площі;

ІІІ – штучний інтелект.

ВСТУП

У сучасному ігровому середовищі Minecraft [1] залишається однією з найпопулярніших та найвідкритіших до модифікації ігор. Створення користувацьких модів завжди викликало значний інтерес у спільноти, адже вони дозволяють не лише урізноманітнити геймплей, а й реалізовувати унікальні ідеї, що виходять за межі канонічного контенту. З розвитком технічної складової гри та розширенням інструментів для розробки, зокрема Forge API, створення модів стало доступнішим навіть для індивідуальних розробників.

Метою дипломної роботи є розробка користувацького моду для версії Minecraft 1.21.1 за допомогою Forge MDK [2], що додає новий функціонал взаємодії з котями: можливість збирати кігті та вичісувати котячу шерсть, а також використовувати ці предмети для створення кастомної броні та інструментів. Особливістю доданого контенту є наявність наряду пов'язаних з предметами ефектів, які з точки зору програмної реалізації відкривають вікно можливостей для інших розробників додавати більш варіативний функціонал як оригінальних, так й модифікованих предметів, використовуючи розроблені мною функції, в той час як для гравця доданий контент дає можливість розширити свої враження від взаємодії як з ванільним мирним мобом, з обмеженим функціоналом, так й додатковий елемент стратегії у виборі спорядження. Також в моді присутні й інші ігрові нововведення, такі як котяча м'ята, а також екстракт з неї, котрий надає кастомний ефект «заспокоєння», що дає гравцю змогу вільно приручати котів без страху що вони його злякаються, так як ефект наряду впливає на їх ІІІ.

Новизна даної модифікації полягає у незвичному підході до використання пасивних мобів у грі та створенні нетипового для ванільного Minecraft крафту, що ґрунтується на дружній взаємодії з тваринами. Це не лише урізноманітнює ігровий процес, а й сприяє формуванню нового способу геймплейної взаємодії в межах знайомої ігрової екосистеми, що не тільки дає

гравцю ширші можливості для вибору своєї ігрової стратегії, а й розширює як враження від вже звичних ігрових механік, так й надає нові можливості для створення багатьох інших модифікацій, котрі будуть використовувати розроблені технічні рішення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Перш ніж розпочати розробку програмного забезпечення, важливо провести ґрунтовний аналіз предметної області. Це дозволяє краще зрозуміти потреби користувачів, оцінити актуальність ідеї, а також визначити вимоги до функціоналу майбутньої системи. У контексті розробки модифікації для гри Minecraft такий аналіз передбачає вивчення запитів гравців, які прагнуть розширити стандартний ігровий досвід, та дослідження вже існуючих модифікацій, які можуть частково або повністю реалізовувати схожі ідеї.

У цьому розділі буде проаналізовано популярні запити гравців на модифікації, зокрема ті, що стосуються взаємодії з тваринами, кастомних предметів і нових механік крафту. Також буде здійснено моніторинг існуючих модів для Minecraft, що дозволить виявити наявні рішення та недоліки, які можна врахувати при реалізації власного проєкту. На основі зібраної інформації буде сформульовано постановку задачі, окреслено межі системи та розробки, а також сформульовано функціональні вимоги до модифікації.

1.1 Аналіз існуючих запитів гравців на модифікації

Серед гравців Minecraft завжди був помітний інтерес до дружелюбних мобів, які могли б стати не лише декоративними улюбленцями, а й справжніми помічниками – вірними компаньйонами, здатними підтримати гравця у складній ситуації або допомогти підготуватися до неї заздалегідь.

Проаналізувавши найбільшу платформу користувацьких модифікацій [3], а також нововведення у версії гри, для якої створюється мод, було зроблено висновок: ідея розширити функціонал саме котів є свіжою, унікальною та здатною зацікавити широку аудиторію, особливо фанатів даного моба.

У вибраній версії значно покращено можливості вовків, що створило певний дисбаланс на їхню користь – у той час як коти залишилися зі скромним набором механік. Саме тому було прийнято рішення розробити мод, який не

лише урівноважить ситуацію, а й зробить котів дійсно корисними ігровими партнерами з розширеною поведінкою та цікавішою взаємодією.

1.2 Моніторинг існуючих модифікацій

Провівши моніторинг найпопулярніших платформ користувацьких модифікацій, зокрема CurseForge [3], було встановлено, що модифікації, пов'язані з котами, є доволі популярними серед користувачів. Проте не дивлячись на це – всі вони стосуються лише зміни текстур [4], моделей [5] або амбієнтного звучання [6]. Також було виявлено, що повністю відсутні моди, які б змінювали або розширювали поведінку котів, роблячи їх більш функціональними та корисними компаньйонами для гравця. Подібні зміни могли б стимулювати активнішу взаємодію з цими та іншими дружелюбними мобами, сприяти глибшому зануренню в ігровий процес і, в перспективі, розширити ігрову стратегію та мислення користувача.

Косметичні модифікації, незважаючи на свою популярність, здебільшого лише умовно впливають на сприйняття гравцем внутрішньоігрових котів, оскільки не змінюють саму суть взаємодії з ними. Вони покращують зовнішнє оформлення, але не надають гравцеві додаткових мотивацій або практичної користі від присутності kota в грі. Натомість розроблена модифікація зробить котів функціонально корисними. Завдяки новим механікам, таким як видобуток ресурсів, створення предметів, додаткові ефекти від броні та інше, взаємодія з котами набуває реального сенсу й користі. Саме тому розроблена модифікація буде мати суттєві переваги над уже існуючими – вона формує нову гілку геймплею, в якій коти стають важливою складовою ігрової стратегії, а не лише декоративним елементом.

1.3 Постановка задачі

Основною метою роботи було створення користувацької модифікації для гри Minecraft, яка б розширювала функціональні можливості котів, роблячи їх більш інтерактивними, корисними та цікавими для гравця.

Для досягнення цієї мети необхідно було насамперед проаналізувати існуючі модифікації, що стосуються котів. В результаті цього аналізу було виявлено їхні обмеження, недостатню кількість механік та відсутність поведінкових змін.

Далі потрібно було дослідити технічні можливості реалізації нових механік у межах вибраної версії гри з використанням Forge API, для чого була проведена ретельна робота по аналізу технічних нововведень обраної версії API [7], а також функціональних меж та можливостей цієї версії.

Одним із ключових завдань стало проектування системи отримання ресурсів від котів – таких як кігті та котяча шерсть, з урахуванням ігрового балансу та без нанесення шкоди тваринам. Окремо було розроблено крафти інструментів і броні з нових матеріалів, а також нову руду – «блок викопних кігтів», що добувається в світі. Крім того, впроваджено торгівлю з мандрівним крамарем, який продає «насіння котячої м'яти», що дозволяє вирощувати однойменну рослину та створювати зілля «екстракт котячої м'яти», яке накладає на гравця кастомний ефект, котрий впливає на поведінку котів і спрощує їхнє приручення.

Завершальними етапами роботи стали тестування модифікації, усунення багів, інтеграція нових механік у вже наявні ігрові системи без порушення балансу, перевірка на сумісність з іншими користувацькими модифікаціями, а також підготовка документації щодо встановлення й використання моду.

1.4 Межі системи та розробки

У межах даної розробки реалізовано модифікацію для гри Minecraft на версію 1.21.1, спрямовану на розширення функціоналу котів як дружелюбних мобів. Основна увага зосереджена на зміні їх поведінки, додаванні нових предметів, механік взаємодії, системи приручення та використання котів як потенційних ігрових компаньйонів. До складу розробки входить: створення нових ресурсів (кігті, котяча шерсть), крафтів предметів («кігтіатор», броня, інструменти), нової рослини (котяча м'ята), алхімічного рецепту («екстракт котячої м'яти»), руди («блок викопних кігтів»), а також інтеграція цих компонентів у загальну ігрову систему.

Разом із цим, модифікація не передбачає змін у базових ігрових механіках, що не стосуються котів, а також не змінює поведінку інших мобів чи структуру ігрового світу загалом, хоча й передбачає вплив на ШІ окремих істот. Також не реалізовується штучний інтелект на рівні повноцінної бойової поведінки або супроводу, як у випадку вовків. Взаємодія з гравцем зберігає спрощений характер і обмежується прирученням, користуванням предметами, отриманням ресурсів та косметичними змінами.

Розробка проводилася виключно під Forge API, без сумісності з іншими модифікаційними платформами, зокрема такими як Fabric, або модпаками, хоча модифікація й має неконфліктну структуру відносно інших користувацьких модифікацій та модпаків, зокрема функціональних, націлених на показ шляхів отримання того чи іншого ресурсу, його використання, та інших подібних. Крім того, не передбачено створення повноцінного інтерфейсу керування або окремого GUI, оскільки основна мета – залишити мод легким для інтеграції у вже існуючі світи гравців та уникнути надмірної ускладненості.

Оскільки оригінальна комп'ютерна версія гри Minecraft розроблена мовою програмування Java, модифікація також реалізовувалась цією мовою. Це обумовлено особливостями архітектури гри та підтримкою Forge API, який сумісний саме з Java-версією Minecraft. Java-версія має як свої переваги, так і

певні недоліки, особливо у порівнянні з кросплатформеною версією Minecraft, написаною мовою C++. Порівняльні характеристики наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння мов програмування

Характеристика	Minecraft Java Edition (ПК)	Minecraft Bedrock Edition (мобільні пристрої, Windows Store)
Мова програмування	Java	C++
Гнучкість модифікацій	Висока (через відкритість і велику кількість API)	Низька (обмежена система аддонів)
Продуктивність	Нижча (вища ресурсоемність)	Вища (оптимізований нативний код)
Кросплатформеність	Обмежена (лише ПК)	Висока (ПК, мобільні пристрої, консолі)
Спільнота модифікаторів	Дуже активна, великий обсяг модів	Менш активна, обмеження з боку платформи
Інструменти для розробки модифікацій	Forge, Fabric, MCP, тощо	Add-ons, Minecraft Creator Tool

1.5 Функціональні вимоги

Розроблена модифікація повинна реалізовувати низку функціональних можливостей, які розширюють базовий ігровий досвід, пов'язаний з котами у Minecraft. Основні функціональні вимоги включають:

- генерація ресурсу «кіготь» з приручених котів з певним інтервалом часу, без нанесення шкоди мобу;
- можливість створення предмета «кігтіатор», за допомогою якого можна взаємодіяти з котами, отримуючи з них котячу шерсть;
- реалізація системи крафту інструментів та броні з блоків кігтів та шерсті, з відповідними характеристиками та внутрішньоігровим балансом;
- додавання нової руди – «блок викопних кігтів», яка генерується у світі й використовується як матеріал для отримання ресурсу «кіготь»;

- додавання торгової взаємодії з мандрівним крамарем, який продає «насіння котячої м'яти»;
- можливість вирощування котячої м'яти та отримання з неї квітки, що використовується для створення зілля;
- додавання зілля «екстракт котячої м'яти», яке при використанні тимчасово зменшує тривожність котів, дозволяючи гравцеві легше їх приручити, шляхом накладення на нього кастомного ефекту «заспокоєння»;
- забезпечення сумісності модифікації з оригінальними механіками гри, без порушення балансу або конфлікту з базовим функціоналом гри;
- відсутність критичних помилок під час використання модифікації, включно з коректною обробкою взаємодії з котами, крафтами, торгівлею та алхімією.

Усі нові механіки мають бути органічно інтегровані в гру, не потребує сторонніх модифікацій, а сам мод реалізовано на базі Forge, без використання coremod-підходу чи прямого редагування ядра Minecraft.

1.6 Висновки за розділом 1

У першому розділі було проведено огляд актуальності теми розробки користувацької модифікації для гри Minecraft, зосередженої на розширенні функціоналу котів. У результаті моніторингу найпопулярніших платформ користувацьких модифікацій встановлено, що хоча тема котів користується популярністю, всі вже існуючі моди фокусуються тільки на косметичних змінах (текстурах, моделях, амбієнті) та не впливають на їхню поведінку або роль у грі. Це дозволило зробити висновок про доцільність створення модифікації, яка додасть котам нових ігрових можливостей та зробить їх повноцінними компаньйонами для гравця.

У процесі аналізу запитів гравців та поточного стану ігрового балансу було виявлено, що після посиленого розвитку функціоналу вовків у нових версіях гри, коти залишились менш корисними. Тому розробка моду, що

урівноважить цей дисбаланс і додасть унікальні механіки саме для котів, є обґрунтованою.

Визначено основну ідею, мету та задачі проекту, а також функціональні вимоги до реалізації модифікації. Також було окреслено межі системи та розробки: мод не передбачає змін у базовій структурі гри, інших модах чи клієнтському ядрі Minecraft, а реалізується на базі Forge API, без використання coremod-підходу. Усі додані механіки гармонійно інтегруються у світ гри та не конфліктують із ванільним ігровим процесом.

2 МАТЕРІАЛИ І МЕТОДИ

У цьому розділі буде розглянуто основні етапи створення концепції та реалізації користувацької модифікації для гри Minecraft, що розширює функціонал котів як дружелюбних мобів. Детально описано методологію розробки, що охоплює формування ігрової ідеї, логіки взаємодії з ігровим світом та підбір засобів програмної реалізації.

Розділ містить обґрунтування вибору інструментів і технологій, які були використані для розробки модифікації, а також описує підхід до створення ігрових механік, що не порушують балансу гри та забезпечують новий рівень взаємодії з котами у межах середовища Minecraft.

Представлені матеріали й методи відображають практичну частину проєкту, на основі якої була реалізована унікальна модифікація, що відповідає попередньо сформульованим вимогам і цілям.

2.1 Задача створення концепції та розробки модифікації

Задача створення концепції модифікації полягала у проєктуванні комплексної та цілісної системи, яка б розширювала ігрові механіки Minecraft через поглиблення взаємодії гравця з дружніми мобами, а саме з котами. Головна мета – перетворити котів з переважно декоративних істот на корисних, багатоцільових компаньйонів, що стимулюють гравця до нових стратегічних рішень та ресурсної активності.

У рамках проєкту було сформовано внутрішню логіку модифікації, де кожен новий ігровий елемент або механіка є як наслідком попередньої взаємодії, так і передумовою для подальших дій. Це забезпечує природну інтеграцію нового контенту у вже існуючу систему гри, без порушення її балансу. Діаграма логіки взаємодій елементів модифікації була розроблена за допомогою draw.io [8] та зображена на рисунку 2.1.

Котяча шерсть, отримана з кота за допомогою кігтізатора, виконує роль проміжного ресурсу у крафтах. Разом із блоками кігтів вона використовується для створення броні, яка, крім базового захисту, має унікальні ефекти, пов'язані з поведінкою котів (наприклад, зниження шкоди при падінні, прискорення під час руху крадькома, та інші).

У модифікацію також було інтегровано елемент сільського господарства через котячу м'яту. Насіння цієї рослини можна отримати шляхом обміну з мандрівним крамарем, що стимулює гравця звертати увагу на випадкові події у світі. З вирощеної котячої м'яти виготовляється екстракт – зілля, що впливає на штучний інтелект котів, тимчасово роблячи їх менш пугливими. Це дозволяє гравцю легше приручати тварин і налагоджувати свою взаємодію з ними.

Всі ігрові елементи концептуально та механічно об'єднані в єдину ігрову екосистему, де жодна дія не є ізольованою. Кожна нова механіка має геймплейне обґрунтування, що формує чіткий гейм-дизайнерський цикл: Взаємодія → Ресурс → Розвиток → Нова взаємодія.

Таким чином, основна задача модифікації – не лише розширення функціоналу одного типу мобів, а й зміна парадигми гравця щодо їх значення у грі. Коти перестають бути лише декоративним елементом і стають частиною ресурсної, бойової та стратегічної системи Minecraft.

2.2 Методи створення концепції та розробки модифікації

Для ефективної розробки модифікації було застосовано комплекс методів, що охоплюють як етапи формування концепції, так і безпосередньої реалізації технічного рішення.

Метод аналізу та порівняння. На першому етапі було здійснено аналіз наявних модифікацій, присвячених мобам, зокрема котам. Здійснено порівняння функціональності існуючих модів із можливостями, які надає базова версія гри (vanilla Minecraft). Це дозволило виявити невикористані або слабо реалізовані аспекти, які могли б стати основою нової модифікації.

Метод синтезу та логічного моделювання. Створення концепції модифікації базувалося на поєднанні кількох ігрових механік – добування ресурсів, фермерства, приручення мобів, крафту інструментів і броні, а також використання зілля. Була побудована внутрішня модель, яка відображає причинно-наслідкові зв'язки між діями гравця та результатами цих дій. Це дозволило сформувати замкнений геймплейний цикл, де кожен компонент логічно витікає з попереднього (рис. 2.1).

Метод ітеративної розробки (agile-підхід). Розробка модифікації здійснювалась поступово, з постійною перевіркою працездатності кожного окремого елемента та зворотним зв'язком із гравцями, що брали участь у тестуванні. Завдяки цьому вдалось поступово удосконалити баланс механік і врахувати можливі помилки на ранніх етапах.

Об'єктно-орієнтований підхід у програмуванні. Оскільки Minecraft написаний мовою Java, реалізація модифікації також здійснювалась з використанням принципів ООП, що дозволило структурувати код у вигляді незалежних, повторно використовуваних класів: коти, предмети, блоки, зілля, інструменти, рецепти крафту, DataGen тощо.

Метод використання сторонніх бібліотек і API. Для інтеграції модифікації в гру було використано середовище Minecraft Forge, що надало доступ до внутрішніх механік гри та спростило процес модифікації без втручання в клієнтське ядро. Це дозволило забезпечити сумісність з іншими модами та уникнути конфліктів.

У сукупності застосовані методи забезпечили реалізацію модифікації, яка не лише розширює функціональність гри, але й органічно вписується в її структуру, не порушуючи внутрішньої логіки Minecraft і зберігаючи відчуття «ванільного» досвіду.

2.3 Вибір засобів розробки

Для реалізації модифікації було обрано набір інструментів, що оптимально підходять для розробки під Minecraft Java Edition та відповідають специфіці поставленого завдання. Основним рушієм розробки став Forge Mod Loader (Forge) – популярний API та середовище для створення модифікацій, яке забезпечує доступ до внутрішніх структур гри без потреби змінювати клієнтське ядро.

Мова програмування. Основною мовою розробки стала Java, оскільки саме на ній написана оригінальна версія Minecraft для комп'ютерів. Це дозволило використовувати об'єктно-орієнтований підхід, а також забезпечило високу сумісність модифікації з ігровими механіками.

Інтегроване середовище розробки (IDE). Для написання, компіляції та налагодження коду використовувалось IntelliJ IDEA [10], що забезпечує зручну роботу з Java-проектами, має розширену підтримку Gradle-білдів та інструментів Forge.

Інструменти білду та запуску. Основною системою збирання проекту був Gradle, який дозволяє ефективно управляти залежностями, компіляцією та пакуванням модифікації у формат .jar.

Генерація JSON-файлів. Всі необхідні ресурси у вигляді JSON-файлів (моделі, рецепти, лут-таблиці, тощо) генерувалися автоматично за допомогою DataGen, з використанням власноруч створеного класу DataGenerators, що дозволило уникнути ручного редагування конфігурацій та мінімізувати ризики помилок. Такий підхід повністю автономний і не залежав від сторонніх інструментів.

Документація та підтримка. Єдиним зовнішнім джерелом, яке було використано у процесі розробки, була офіційна документація Forge. Розробка здійснювалася самостійно, без залучення сторонніх ресурсів, форумів чи відкритих спільнот, всі текстури були намальовані спеціально для даної роботи,

за допомогою безкоштовного онлайн-сервісу [11], та є повністю авторськими, що підкреслює індивідуальний характер роботи.

2.4 Висновки за розділом 2

У цьому розділі було сформовано загальну ідею модифікації та визначено, яким чином реалізовані її ключові механіки. Були встановлені логічні зв'язки між основними елементами гри – котами, предметами, ресурсами – та їх впливом один на одного. Це дозволило побудувати чітку структуру взаємодій, на основі якої й відбувалась подальша розробка.

Також було описано методи, за допомогою яких створювалась концепція. Зокрема, основну роль відіграли особистий ігровий досвід, спостереження за поведінкою гравців, а також розуміння внутрішніх механік Minecraft.

Обґрунтовано вибір мови програмування, середовища розробки та інструментів. Особливу увагу було приділено тому, що всі ресурси модифікації – такі як моделі, рецепти чи лут-таблиці – генерувалися автоматично за допомогою власного DataGenerators класу, без сторонніх інструментів чи платформ.

Таким чином, на основі отриманих висновків було сформовано міцний фундамент, на якому й базуватиметься подальша практична реалізація модифікації.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

У цьому розділі детально розглядаються практичні аспекти реалізації модифікації, зокрема – способи взаємодії користувача з її компонентами та алгоритми, що забезпечують коректну роботу системи. На основі концепції, описаної в попередніх розділах, сформовано сценарії використання кожного з основних елементів модифікації: предметів, блоків, мобів та зілля.

Також увагу зосереджено на описі внутрішньої логіки функціонування – як відбувається генерація ресурсів, взаємодія між об'єктами гри та обробка подій, пов'язаних із використанням нових механік. Усі ці аспекти проєкту розглядаються з точки зору ігрової зручності, технічної ефективності та узгодженості з базовою логікою Minecraft.

Результати цього розділу дозволять краще зрозуміти, як саме реалізовані механіки модифікації, та стануть основою для оцінки її працездатності і впливу на ігровий процес.

3.1 Визначення сценаріїв взаємодії користувача з модифікацією

Модифікація була створена з метою поглиблення взаємодії гравця з котами в Minecraft, зробивши цих мобів не лише декоративними, а й функціонально корисними впродовж ігрового процесу. У рамках реалізованої концепції було закладено систему пов'язаних між собою ресурсів, предметів, блоків та механік, що формують новий шлях розвитку гравця – «шлях кота».

Усі ключові об'єкти модифікації взаємодіють між собою у межах циклічної економіки ресурсів: нові предмети добуваються, переробляються, об'єднуються в нові інструменти та броню, і мають як стратегічну, так і тактичну цінність у бою, дослідженні світу, або виживанні. В основі системи – ресурс «кіготь», який можна отримати як шляхом видобутку, так і шляхом взаємодії з котами.

Важливою складовою є й розвиток сільськогосподарського напрямку – вирощування котячої м'яти як спеціального ресурсу, що дозволяє впливати на поведінку котів і відкривати додаткові можливості взаємодії з ними. Нові ефекти, які додаються через споживання або використання певних предметів, створюють унікальний досвід та стимулюють експерименти з комбінаціями ресурсів.

Гравець самостійно приймає рішення, у який спосіб розвивати «котячу гілку» – зосередитись на броні, інструментах чи на вирощуванні рослин, що забезпечує гнучкість та індивідуалізацію ігрового стилю.

Також модифікація підтримує два види локалізації, а саме українську та англійську, а для простоти взаємодії з модом, гравець може побачити всі предмети та блоки з моду в окремих відповідних вкладках креативного режиму, а всю важливу інформацію про нього в панелі «mods» головного вікна гри (рис. 3.1).



Рисунок 3.1 – Основна інформація про модифікацію

3.1.1 Взаємодія з предметами модифікації

У рамках розробленої модифікації було реалізовано кілька нових предметів, що тісно пов'язані між собою логікою циклічного розвитку, заглиблюючи ігрову механіку та відкриваючи перед гравцем нові способи добування ресурсів і вдосконалення спорядження. Основну роль у цьому ланцюжку відіграють кігті, які можна або видобути як руду у світі, або отримати шляхом взаємодії з котами.

Кігті – це базовий ресурс модифікації. Гравець може отримати їх:

- при взаємодії з котами раз на 5 ігрових днів (у кількості 3–6 штук);
- шляхом видобутку руди викопних кігтів на висоті 32–64 блоки (1–3 штуки, залежно від чарів «Удача»);
- або здобути блок руди цілком за допомогою чарів «Шовковий дотик».

Кігті можна:

- використовувати для створення кігтіатора, блоку кігтів, або як ресурс для ремонту інструментів та броні;
- переплавити у звичайні кістки (1 кіготь = 1 кістка).

Текстура кігтів зображена на рисунку 3.2.



Рисунок 3.2 – Текстура предмету «кіготь»

Кігтізатор – одноразовий предмет, що дозволяє отримати шерсть з будь-якого кота (прирученого чи ні). Має обмежену міцність, однак підтримує зачарування на незламність, що дозволяє продовжити термін його служби. Текстура кігтізатора зображена на рисунку 3.3.



Рисунок 3.3 – Текстура предмету «кігтізатор»

Котяча Шерсть – унікальний предмет, який не використовується у ванільному Minecraft, але має цінність у межах модифікації. Її можна здобути лише через використання кігтізатора шляхом взаємодії з котом. Використовується як інгредієнт у крафтах броні, в комбінації з блоками кігтів. Текстура котячої шерсті зображена на рисунку 3.4.



Рисунок 3.4 – Текстура предмету «котяча шерсть»

Насіння котячої м'яти – може бути отримане як рідкісна пропозиція під час торгівлі з мандрівним крамарем. Висаджується в ґрунт, росте аналогічно до будь-якої іншої посівної культури.

Після повного росту дає:

- кілька насінин;
- кілька одиниць котячої м'яти.

Текстура насіння котячої м'яти зображено на рисунку 3.5.

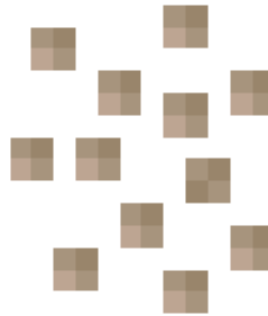


Рисунок 3.5 – Текстура предмету «насіння котячої м'яти»

Котяча м'ята – рослинний ресурс, що використовується у двох формах:

- їстівний предмет, що симулює отруйність для гравця, шляхом накладення ефектів нудоти та сліпоти;
- інгредієнт для створення екстракту котячої м'яти.

Текстура котячої м'яти зображено на рисунку 3.6.



Рисунок 3.6 – Текстура предмету «котяча м'ята»

Екстракт котячої м'яти – зілля, що виготовляється з котячої м'яти та скляної пляшки. При вживанні надає кастомний ефект «заспокоєння», який діє 30 секунд та робить котів менш пугливими по відношенню до гравця (дозволяє

легше до них наблизитися або приручити). Для відповідного ефекту була розроблена окрема іконка інтерфейсу, аналогічна до тієї, що використовується в якості логотипу модифікації (рис. 3.1). Текстура екстракту котячої м'яти зображена на рисунку 3.7.

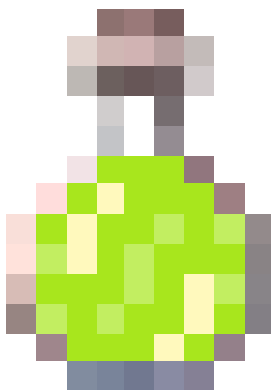


Рисунок 3.7 – Текстура предмету «екстракт котячої м'яти»

Інструменти та броня – створюються з блоків кігтів, та блоків кігтів у поєднанні з котячою шерстю, відповідно.

За характеристиками еквівалентні діамантовим аналогам, але мають:

- підвищений шанс на покращені зачарування;
- броня – унікальні ефекти, наведені у таблиці 3.2.

Таблиця 3.2 – Унікальні ефекти броні

Елемент броні	Ефект
Шолом	Змушує ШІ кріперів боятися гравця так само, як вони бояться котів – вони починають тікати.
Нагрудник	Зменшення радіусу агресії ШІ мобів на гравця вдвічі (наприклад, скелет помічає гравця не з 16, а з 8 блоків).
Штані	Прискорення руху в режимі крадькома, але не вище базової швидкості. При наявності зачарування «Біг крадькома III» – швидкість руху в режимі крадькома дорівнює звичайній.
Чоботи	Зменшення шкоди від падіння вдвічі, утримовування рихлого снігу під ногами в звичайний блок.
Будь-яка частина броні	Захист від замерзання в рихлому снігу.

Також представлені інструменти та елементи броні мають унікальні тривимірні (3D) текстури.

Броня реалізована із підтримкою індивідуальних шаблонів для кожної частини екіпірування, завдяки чому вона не лише виконує свою функціональну роль у грі, а й слугує потужним інструментом для візуального самовираження гравця. Підтримка повноцінної системи шаблонів дозволяє адаптувати текстури до будь-якої моделі гравця, що забезпечує коректне відображення в усіх можливих ситуаціях – від анімацій руху до видимості в інвентарі та від третьої особи. Цей підхід сприяє глибшому зануренню у процес кастомізації персонажа, заохочуючи гравців активно експериментувати із зовнішнім виглядом та створювати унікальні образи.

Таким чином, модифікація розширює стандартні можливості гри не лише в механічному, а й в естетичному аспекті, що особливо важливо в умовах сучасної спільноти Minecraft, орієнтованої на персоналізацію та креативність. Приклади вигляду інструментів та броні в різних ігрових ситуаціях зображені на рисунках 3.8–3.10.



Рисунок 3.8 – Вигляд броні з накладеним шаблоном «тиша», у порівнянні з діамантовим аналогом

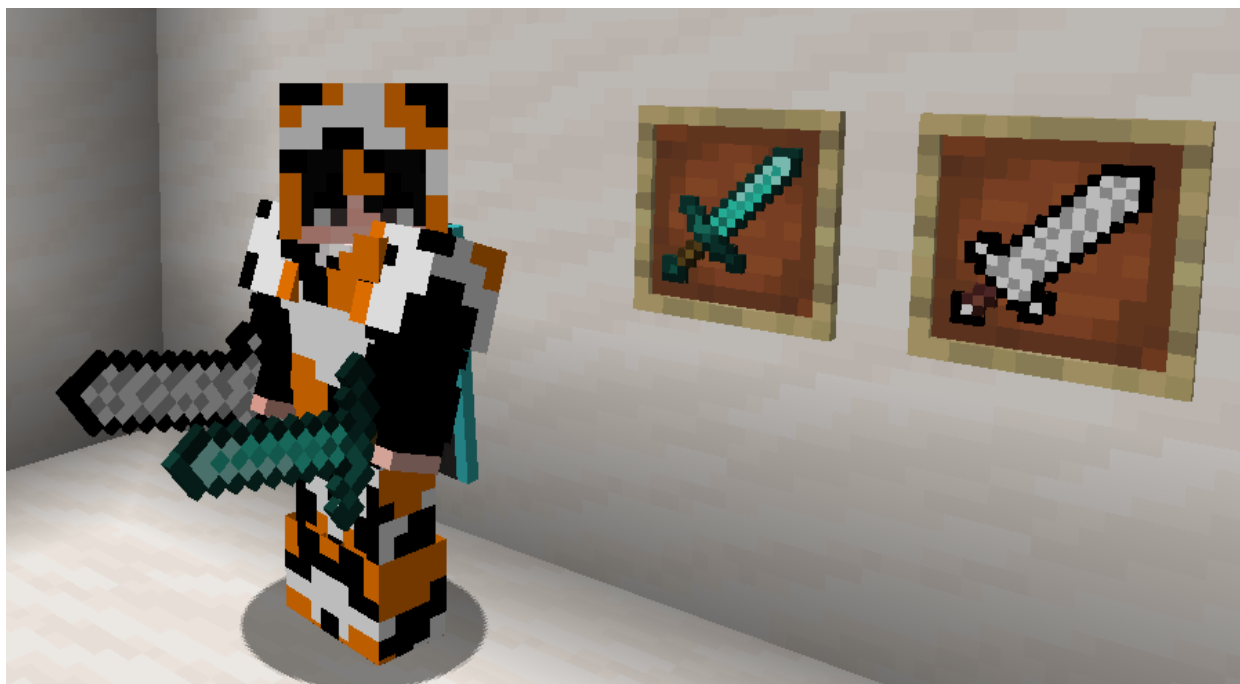


Рисунок 3.9 – Вигляд броні безпосередньо на гравці, порівняння текстур та моделей ванільних предметів (діамантового меч) та кастомних (кігтяний меч)



Рисунок 3.10 – Відображення броні, ефекту та предметів на гравці безпосередньо в інтерфейсі інвентаря

3.1.2 Взаємодія з блоками модифікації

У межах розробленої модифікації було реалізовано два основні типи блоків: блок кігтів та блок викопних кігтів (руда), кожен з яких відіграє важливу роль у логіці розвитку ігрового процесу та надає гравцю нові можливості для взаємодії з ігровим світом.

Блок викопних кігтів генерується природнім чином у світі Minecraft на висотах від 32 до 64 блоків. При видобутку звичайним інструментом, з неї випадає один кіготь, однак за використання кайла із зачаруванням «Удача» гравець може отримати більше. Якщо ж руда видобувається із зачаруванням «Шовковий дотик», вона зберігається як повноцінний блок і може бути використана для декоративних або технічних цілей, наприклад, у автоматизованих фермах або як частина інтер'єру. Також під час переплавлення блоку викопних кігтів гравець отримує по одному кігтю за кожен переплавлений блок.

Блок кігтів є крафтовим елементом, що створюється з дев'яти кігтів. Його можна знову розкласти на входні інгредієнти, що робить його ефективним способом зберігання ресурсу. Крім того, цей блок може бути використаний для створення кістяного блоку шляхом переплавлення, а також входить до рецептів крафту інструментів та елементів броні. Такий підхід дозволяє гравцеві не тільки зберігати ресурс у компактному вигляді, а й застосовувати його в подальшому для розвитку спорядження.

Окрім цього, обидва блоки мають унікальні текстури (рис. 3.11–3.12), які гармонійно вписуються в стилістику гри та не вибиваються з її візуального контексту, мають безшовний характер (СТМ), зберігаючи цілісність естетичного сприйняття світу Minecraft. Їх наявність стимулює гравця до дослідження світу, збору ресурсів, а також планування подальших дій щодо використання здобутих матеріалів.



Рисунок 3.11 – Порівняння текстур вугільної руди з блоком викопних кігтів

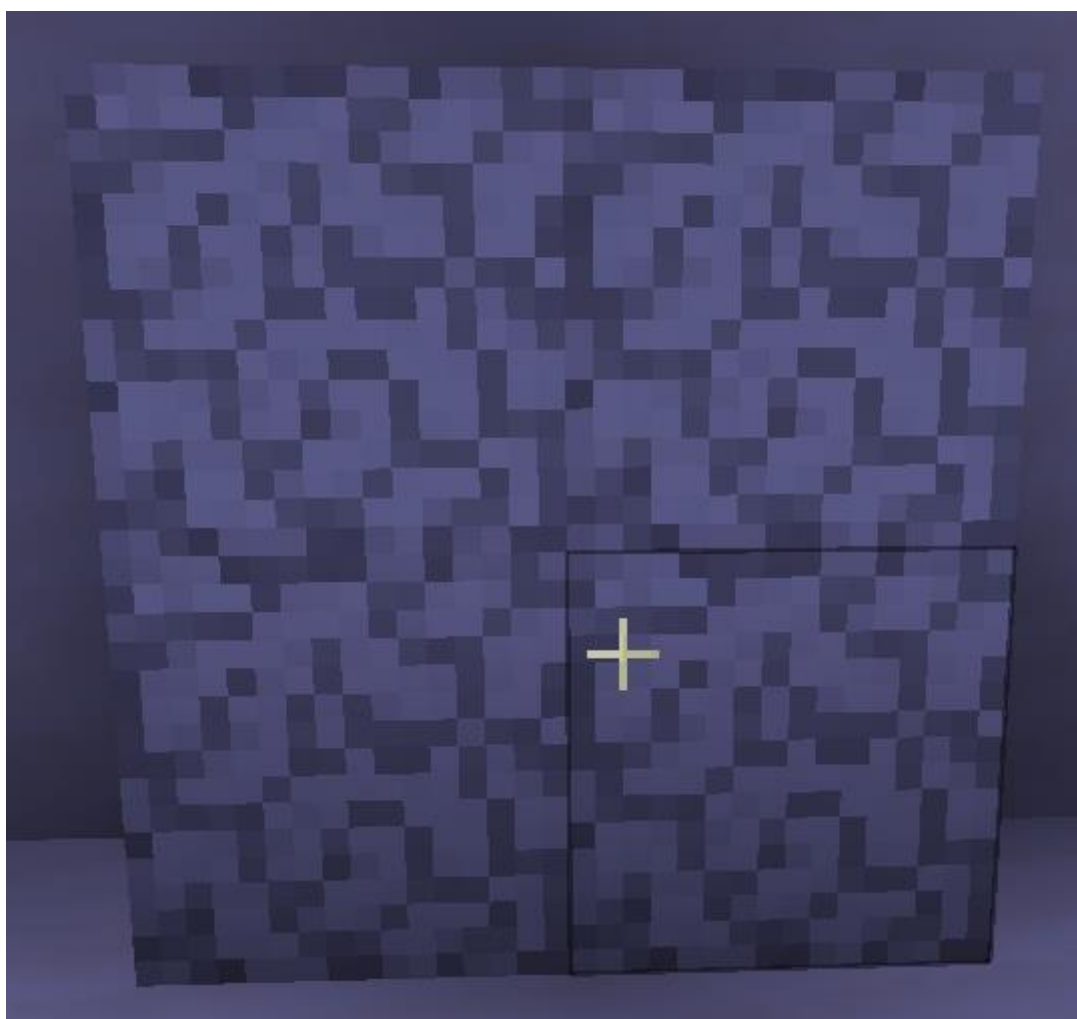


Рисунок 3.12 – Текстура блоку кігтів

На рисунку 3.13 зображено один з варіантів використання блоку кігтів безпосередньо в крафтах.

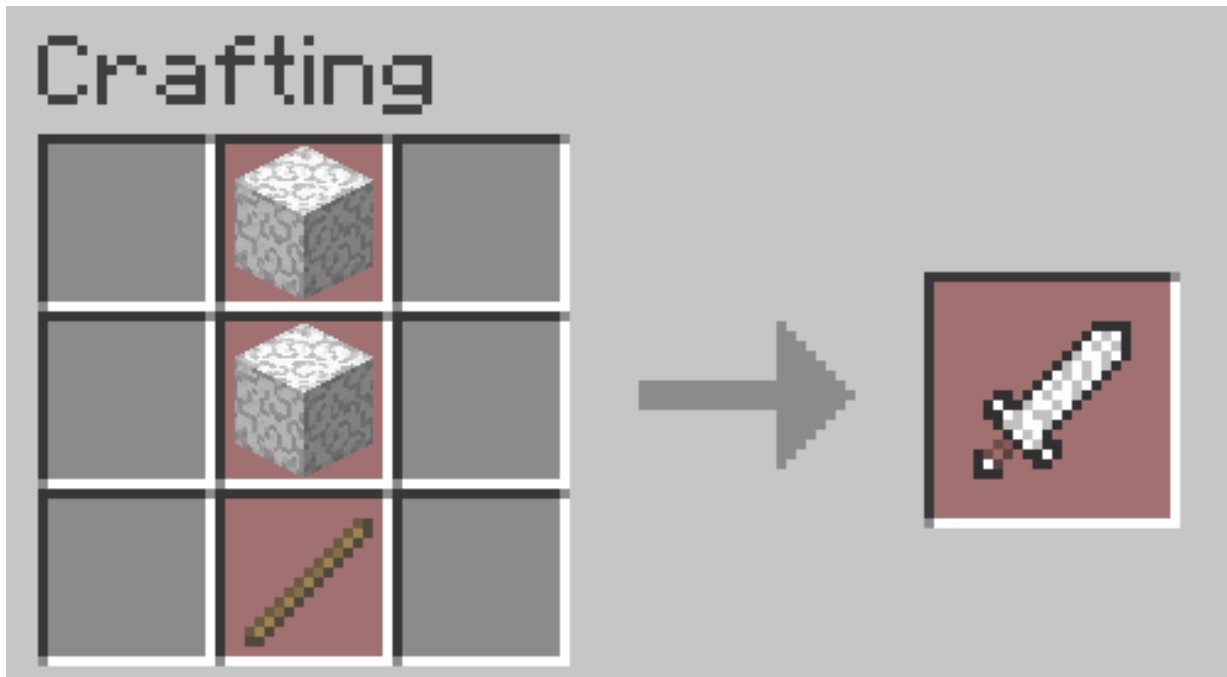


Рисунок 3.13 – Приклад використання блоку кігтів в створенні предметів

Таким чином, система блоків у межах модифікації відіграє не лише декоративну роль, а й функціональну, поглиблюючи ігровий процес та стимулюючи гравця до більш усвідомленої і стратегічної взаємодії з оточенням.

3.2 Визначення алгоритмів функціонування модифікації

Реалізація функціональних можливостей модифікації базується на низці алгоритмів, які забезпечують логіку взаємодії гравця з новими об'єктами, предметами та живими істотами в ігровому світі Minecraft. Усі алгоритми були розроблені з урахуванням гнучкості, оптимальності виконання та мінімального впливу на продуктивність гри.

Генерація руди викопних кігтів. При генерації світу активується модифікований алгоритм заповнення підземного простору, який визначає появу блоків руди викопних кігтів у межах висот від 32 до 64 блоків. Кількість жил на

чанк та розмір кожної жили контролюється конфігураційними параметрами, що дозволяє збалансувати рідкість ресурсу.

Випадіння кігтів з котів.

Алгоритм періодично перевіряє, чи пройшло 5 ігрових днів від останнього добування кігтів з конкретного kota (відстежується за допомогою внутрішнього таймера в NBT-даних). Після закінчення цього інтервалу дозволяється повторне отримання ресурсу, що стимулює гравця до утримання кількох котів і побудови власної системи фермерства.

Робота кігтізатора. При взаємодії предмета з котом гравець отримує одиницю котячої шерсті, а сам предмет витрачається. Алгоритм також враховує чарування «Незламність», яке з певною ймовірністю запобігає знищенню предмета при використанні.

Механіка котячої м'яти. Після висадки насіння запускається цикл росту, аналогічний іншим сільськогосподарським культурам гри. При досягненні фінальної стадії дозрівання алгоритм генерує випадуючі предмети – насіння та котячу м'яту. При використанні м'яти з пляшечкою створюється зілля, яке накладає на гравця ефект «заспокоєння», що змінює поведінку найближчих котів (вони перестають лякатися гравця та тікати від нього, що значно спрощує їх приручення).

Визначення поведінки броні та інструментів. Алгоритми броні реалізують унікальні ефекти шляхом підключення до подій гри, таких як падіння, пересування або наближення мобів. Наприклад:

- чоботи перевіряють тип блоку під гравцем та замінюють рихлий сніг на звичайний;
- нагрудник перехоплює запити агресивних мобів на виявлення гравця та модифікує відстань їх реакції на гравця;
- шолом взаємодіє з логікою кріперів, викликаючи втечу, аналогічну реакції на звичайних котів.

Крім того, всі елементи модифікації працюють в межах стандартного клієнта Forge, не змінюючи його ядро, а лише додаючи розширення за

допомогою подій, тегів та ресурсних елементів, що дозволяє легко підтримувати та оновлювати мод. Кроки встановлення та запуску гри з розробленою модифікацією зображені на рисунках 3.14–3.16.

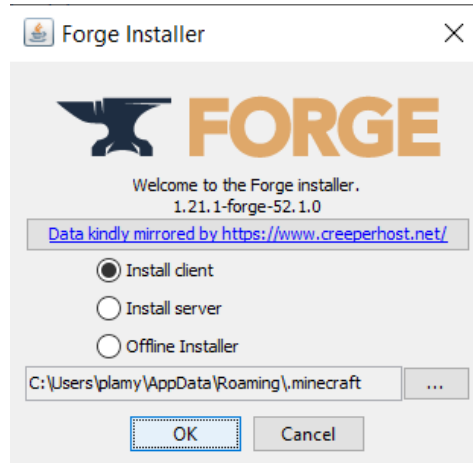


Рисунок 3.14 – Встановлення клієнту Forge потрібної версії

AppData > Roaming > .minecraft > mods			
Имя	Дата изменения	Тип	Размер
 diplomod-1.0+forge1.21.1	13.04.2025 21:16	Executable Jar File	119 КБ

Рисунок 3.15 – Встановлення файлу моду до виконуваної папки модифікацій Minecraft



Рисунок 3.16 – Запуск клієнту Forge з потрібною версією гри

3.3 Висновки за розділом 3

У даному розділі було детально розглянуто ключові рішення, що лежать в основі реалізації компонентів створеної модифікації. Було визначено сценарії взаємодії користувача з основними об'єктами гри, включно з новими предметами, блоками, рослинами та живими істотами. Ці сценарії спрямовані на створення цілісного ігрового циклу, де всі додані механіки взаємопов'язані між собою та мають логічне продовження.

Особлива увага була приділена встановленню та запуску гри з розробленою модифікацією, а також алгоритмам функціонування модифікації, які забезпечують не лише технічну працездатність доданого функціоналу, але й високий рівень інтеграції в існуючу систему гри Minecraft. Реалізовані механіки не потребують змін у клієнтському ядрі гри, а використовують можливості Forge API, що підвищує гнучкість розробки та забезпечує стабільність роботи модифікації на різних конфігураціях.

Таким чином, модифікація є прикладом системного підходу до розширення ігрових можливостей, де кожен елемент створеної структури має чітке функціональне призначення та доповнює загальну концепцію геймплею.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

У цьому розділі розглянуто основні аспекти практичного використання розробленої модифікації: її призначення, вимоги до взаємодії з користувачем, технічні умови виконання, а також результати роботи в ігровому середовищі. Також було проведено тестування стабільності, сумісності та логічної цілісності модифікації в умовах реального геймплею.

Мета цього розділу – продемонструвати працездатність реалізованого функціоналу, перевірити його відповідність поставленим вимогам та показати ефективність обраних рішень у контексті користувацького досвіду.

4.1 Основне призначення програми

Основним призначенням створеної модифікації для гри Minecraft є функціональне розширення ігрової механіки, пов'язаної з мобами-котями, з метою підвищення їхньої практичної користі, інтерактивності та залученості у геймплей. У ванільній версії гри коти відіграють переважно пасивну роль – їхня присутність обмежується декоративною функцією, супроводом гравця та пасивним захистом від фантомів. У той час як інші приручені моби, наприклад вовки, лами або віслюки, мають виражені геймплейні властивості (бойова підтримка, транспортування ресурсів тощо), коти в більшості випадків залишаються елементом атмосфери, а не ігрової стратегії.

Розроблена модифікація покликана змінити це положення, перетворивши котів на повноцінних супутників у виживанні. Вона впроваджує низку нових механік, що надають гравцеві змогу ефективніше взаємодіяти з мобами-котями й отримувати з них практичну користь. Зокрема:

- реалізовано систему добування кігтів із котів, які є ключовим компонентом для створення кастомної броні, інструментів та унікального предмету – кігтіатора;

- додано механіку добування шерсті, яка використовується як ресурс для крафту;
- створено повноцінний набір спорядження (броня та інструменти) з унікальними ефектами, які змінюють поведінку гравця, зокрема дають додаткові бафи, прискорення або інші бонуси в бою;
- впроваджено нову рослину – котячу м'яту, яку можна вирощувати, збирати та використовувати для створення спеціального зілля, що полегшує приручення котів і створює додаткову стратегію для гравця;
- додано новий тип руди — викопні кігті, що розширює аспект дослідження і добування ресурсів.

Усі ці елементи створюють цілісну ігрову систему, що мотивує гравця активно взаємодіяти з котями як з ключовими NPC-компаньйонами, а не просто декоративними істотами. Таким чином, модифікація не лише урізноманітнює ігровий процес, а й змінює парадигму ставлення до мобів-котів у грі, роблячи їх корисними з точки зору виживання, прогресії та стратегії.

Крім того, мод має освітній і пізнавальний потенціал, оскільки формує у гравців розуміння догляду за тваринами, їх значущості у житті людини, а також привносить емоційно-позитивний аспект у гру, що особливо важливо для будь-якої аудиторії. Саме поєднання функціонального, стратегічного й емоційного компонентів і визначає головне призначення розробленої модифікації – надати користувачеві новий досвід взаємодії з улюбленими мобами через глибоко інтегровані ігрові механіки.

4.2 Вимоги до інтерфейсу

Інтерфейс взаємодії з модифікацією реалізовано з дотриманням принципів мінімалізму та інтеграції в основний стиль гри Minecraft. Оскільки модифікація не потребує створення окремих графічних меню чи складних екранів налаштувань, усі її функціональні елементи взаємодіють із гравцем

через вже існуючі в грі механізми – крафтову сітку, інвентар, інтерфейс взаємодії з крамарями, а також стандартні способи використання предметів.

Основні вимоги до інтерфейсу модифікації полягають у:

- інтуїтивності – нові предмети, ресурси та механіки не потребують спеціального навчання чи документації для їх розуміння. Гравець може експериментувати, використовуючи звичні підходи гри;
- візуальній відповідності стилю Minecraft – усі 3D-моделі броні, предметів та блоків мають візуальне оформлення, стилістично наближене до ванільного контенту гри, що забезпечує гармонійне сприйняття;
- підтримці шаблонів текстур – броня підтримує кастомізацію через ковальські шаблони, що дає змогу гравцеві змінювати зовнішній вигляд відповідно до власних уподобань.

Таким чином, модифікація не вимагає від користувача жодного додаткового програмного забезпечення чи вивчення нових інтерфейсів, а її функціонал повністю інтуїтивно зрозумілий та природно вбудований у базову гру.

4.3 Умови виконання програми

Для коректної роботи розробленої модифікації необхідно дотримуватись певних технічних та програмних умов, які гарантують стабільне та коректне виконання функціоналу в ігровому середовищі Minecraft, без порушення цілісності інших аспектів та механік оригінальної гри, що дасть гравцю змогу повною мірою оцінити можливості розробленої модифікації.

Програмні умови:

- версія гри: Minecraft Java Edition версії 1.21.1 – саме під цю версію було реалізовано та протестовано модифікацію;
- платформа модифікації: Forge Mod Loader відповідної версії (Forge 1.21.1), встановлений у клієнт гри;

- операційна система: мод протестовано на ОС Windows 10/11, однак він також повинен стабільно працювати на інших ОС, які підтримують Minecraft Java Edition та Forge (наприклад, Linux або macOS);

- версія Java: Java 17 [12] або новіша, відповідно до вимог гри починаючи з Minecraft 1.18.

Апаратні умови:

- оперативна пам'ять (RAM): мінімум 4 Гб, рекомендовано 8 Гб або більше для комфортної гри з модифікаціями;

- процесор: двоядерний процесор з тактовою частотою не нижче 2.5 ГГц;

- відеокарта: з підтримкою OpenGL 4.5 або новішої;

- місце на диску: мінімум 300 Мб вільного простору (з урахуванням клієнта гри та Forge).

Модифікація не вимагає підключення до інтернету для базової роботи, проте для її завантаження, встановлення Forge, а також перевірки оновлень або взаємодії з іншими гравцями в мультиплеєрі — доступ до мережі може бути необхідним.

4.4 Результати роботи програми

У результаті розробки модифікації для Minecraft було реалізовано функціональний ігровий контент, що розширює базові механіки гри та додає нові можливості для взаємодії з котами та навколишнім світом. Основні результати полягають в успішній інтеграції розроблених програмних рішень, розширюючи взаємодію гравця з різними аспектами гри.

Успішно інтегровано нові ресурси:

- руду «блок викопних кігтів», що природно генерується в ігровому світі в межах висот 32–64 блоки;

- предмети: кігті, шерсть, кігтіатор, насіння котячої м'яти, котячу м'яту, екстракт м'яти тощо;

- нові блоки: блок кігтів, блок викопних кігтів.

Реалізовано нові механіки:

- вилучення кігтів з котів з інтервалом у 5 ігрових днів;
- отримання шерсті з котів за допомогою кігтізатора;
- крафт броні та інструментів з нових матеріалів;
- вирощування котячої м'яти як сільськогосподарської культури;
- застосування зілля «екстракт котячої м'яти», що спрощує приручення котів і взаємодію з ними;
- кастомні ефекти броні, що впливають на поведінку мобів, пересування гравця та взаємодію зі снігом.

Гарантовано стабільну роботу:

- проведено локальне тестування усіх основних функцій;
- забезпечено сумісність з Forge 1.21.1;
- не виявлено серйозних помилок чи конфліктів з внутрішніми механіками Minecraft.

Покращено естетику гри:

- додано повноцінні 3D-моделі броні та блоків;
- забезпечено підтримку кастомних текстур та шаблонів;
- нові предмети гармонійно вписуються у візуальний стиль Minecraft.

Усі реалізовані елементи були протестовані не тільки в одиночному режимі гри, а й в багатокористувацькому на сервері, з використанням чистого клієнта Minecraft та встановленого Forge. Модифікація також сумісна з іншими користувацькими модами, зокрема JEI (Just Enough Items) [13], який показує рецепти предметів, а також їх можливі використання. Отримані результати підтверджують як технічну стабільність, так і ігрову цінність доданої модифікації.

Усі функції тестувались в одиночному режимі гри одразу після імплементації, для чого було використано спеціальний тестовий снапшот-світ, названий аналогічно до логіки найменування снапшотів розробниками самого

Minecraft, що не тільки є елементом зв'язку розробленої модифікації з оригінальною грою, а й спрощує розуміння та структурування в процесі тестування доданих технічних наробіток. Снапшот-світ має назву 25w20e, що при розшифруванні розкриває інформацію про те, що останній снапшот моєї модифікації був розроблений та протестований перед повноцінним релізом на 20 тижні 2025 року, також літера «e» в назві вказує на порядковий номер версії тестування, відповідно до алфавіту це означає, що в підсумку мені знадобилось всього 5 бета-тестів для створення моду. Відносно завдання ці 5 версій були ітераційно рівномірно розподілені при розробці:

- створення класу `DataGenerators`, що реалізує функціонування алгоритмів `DataGen`, завдяки чому ризики помилки під час заповнення JSON-файлів були зведені до мінімуму;
- створення та заповнення класів, та даних, для безпосереднього додавання до гри усіх необхідних предметів, текстур, локалізацій, іконок та вкладок креативного режиму;
- створення усіх необхідних умов та підключення до внутрішньоігрових подій, потрібних для коректного функціонування модифікованих івентів, таких як випадіння кігтів з котів, вплив ефектів від зілля та броні на III істот, та інше;
- перевірка коректності функціонування всіх розроблених елементів модифікації після їх безпосереднього додавання до гри;
- підсумкове налагодження та бета-тестування всіх імплементованих функцій та механік, фінальна збірка модифікації в виконуваний .jar файл.

Снапшот-світ для бета-тестування всіх ітерацій імплементування розроблених елементів модифікації зображений на рисунку 4.1.



Рисунок 4.1 – Снапшот-світ бета-тестів модифікації під назвою 25w20e

Перевірка події на випадіння кігтів з котів після проходження 5 ігрових діб для конкретної особини, в незалежності від того, чи приборкана вона, чи ні, зображена на рисунках 4.2–4.3.



Рисунок 4.2 – Перевірка події на випадіння кігтів (частина 1)



Рисунок 4.3 – Перевірка події на випадіння кігтів (частина 2)

На рисунках 4.2 та 4.3 продемонстровано двох котів, лівий з яких – приборканий, а правий – дикий (для наглядності тестування в незалежності від типу кота). Після призиву та приборкання першого кота, за допомогою команди `time` [14] було додано одну ігрову добу, після чого було призвано ще одного кота, завдяки чому другий кіт мав NBT-тег днів існування рівним 0, в той час як перший кіт мав тег рівний 1. Після командного додавання ще 4 ігрових днів, на рисунку 4.2 можна побачити що лівий кіт досяг NBT-тегу днів існування рівного 5, що спровокувало спрацьовування події випадіння з нього кігтів та обнуління тегу, в той час як правий кіт ще не досяг необхідної кількості днів та нічого не викинув (4 дня з 5). Після командного додавання ще однієї ігрової доби – вже правий кіт досяг необхідного NBT-тегу днів існування рівного 5, що спровокувало його механізм випадіння кігтів та обнуління тегу, демонстрація чого зображена на рисунку 4.3, в цей час тепер вже лівий кіт нічого не викинув, оскільки після "вчорашнього" обнуління тегу – тепер він в нього дорівнює одній, доданий командою, добі.

Подія вичісування шерсті з кота (прирученого чи дикого) за допомогою кігтіатора зображена рисунках 4.4–4.5.



Рисунок 4.4 – Перевірка події на вичісування з котів шерсті за допомогою кігтіатора (частина 1)



Рисунок 4.5 – Перевірка події на вичісування з котів шерсті за допомогою кігтізатора (частина 2)

На рисунках 4.4 та 4.5 зображено етапи перевірки вичісування шерсті з котів за допомогою кігтізатора. Головною умовою виконання події є наявність кігтізатора в руці гравця (рис. 4.4), у випадку виконання умови та взаємодії гравця з котом (приборканим чи диким) за допомогою правої кнопки миші – він отримає котячу шерсть в розмірі 1 одиниці (рис. 4.5), після чого кігтізатор зламається з характерним звуком поламки інструменту в Minecraft, якщо тільки на нього не буде накладено одне з зачарувань на подовження міцності предмета.

Подія додавання торгу насіння котячої м'яти до пропозицій мандрівного крамаря з рідкісним шансом (ціна одного предмету складає цілих 5 смарагдів, на відміну від звичайних та незвичних предметів з цінами 1 та 2–3 смарагди,

відповідно), а також інтеграція події модифікації на додавання блоку викопних кігтів до внутрішнього ігрового генератору світу зображені рисунках 4.6–4.7.

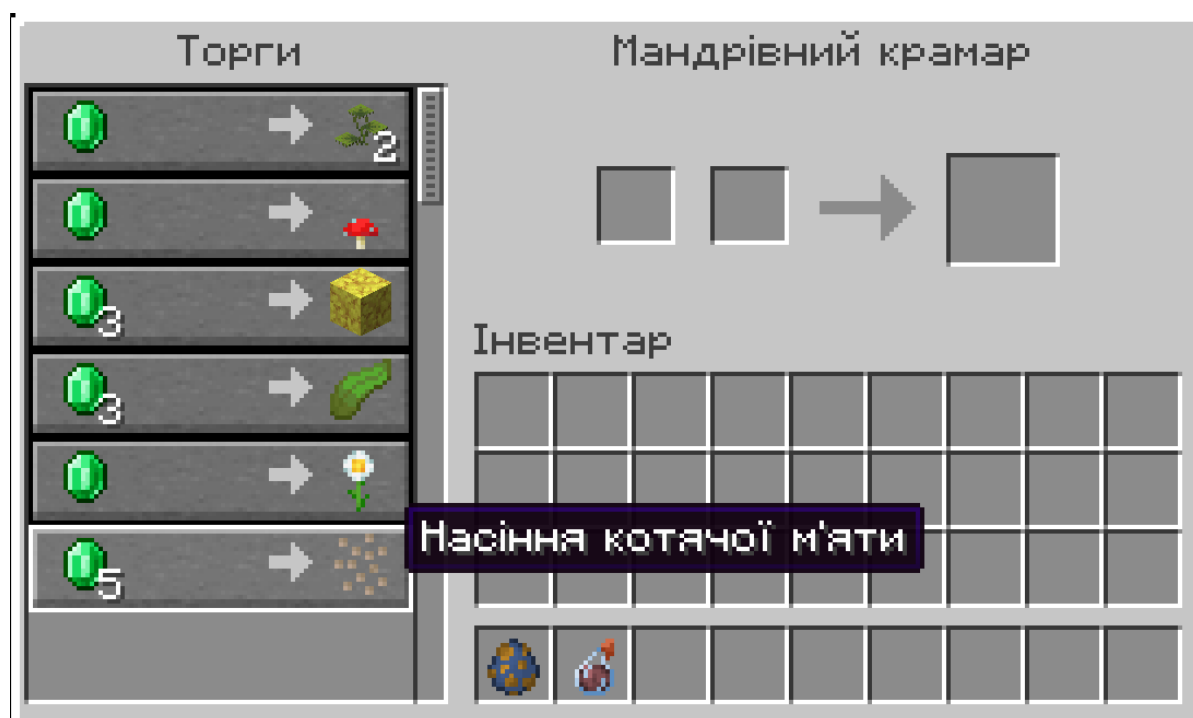


Рисунок 4.6 – Перевірка наявності події додавання пропозиції для мандрівного крамаря



Рисунок 4.7 – Перевірка наявності події додавання блоку викопних кігтів до внутрішнього ігрового генератору світу

На рисунку 4.7 зображено як за допомогою команди fill [15] були видалені усі блоки одного чанку, окрім руд, а також включений режим налагодження для показу кордонів чанків, для більшої наглядності, що блок викопних кігтів дійсно генерується в світі в достатній кількості на чанк, в межах висот від 32 до 64, щоб гравець сам мав змогу обирати яким чином він бажає розвивати свою ігрову стратегію, прирученням котів, чи дослідженням світу.

Під час всього етапу тестування критичних багів та помилок виявлено не було, що вказує на високу якість програмної реалізації. Після завершення розробки модифікація була протестована також і на сервері, запущеному за допомогою безкоштовного хостингу Aternos [16], в багатокористувацькому режимі гри, під час чого також не було виявлено ніяких критичних багів та помилок, а сам процес онлайн-тестування зображений на рисунках 4.8–4.10.



Рисунок 4.8 – Сумісність розробленої модифікації з Just Enough Items (JEI)



Рисунок 4.9 – Коректність відображення інструментів та броні з шаблонами в мультиплеєрі



Рисунок 4.10 – Відображення розроблених 3D текстур посіву на сервері

4.5 Висновки за розділом 4

У даному розділі було проведено аналіз експлуатаційних характеристик, тестування та оцінку роботи розробленої модифікації. В результаті можна зробити такі висновки:

- створена модифікація повністю виконує покладені на неї функції та забезпечує розширення ігрових механік без порушення балансу оригінальної гри;
- реалізований інтерфейс та логіка взаємодії з новими предметами і блоками є інтуїтивно зрозумілими та легко освоюються гравцями без додаткових інструкцій;
- умови виконання модифікації обмежуються лише наявністю встановленого Forge відповідної версії, що відповідає сучасним вимогам до Minecraft-моддингу;
- усі основні функціональні компоненти модифікації були ретельно протестовані в ігровому середовищі – помилки не виявлено, продуктивність системи не знижено;
- впроваджені механіки та візуальні доповнення покращують досвід гравця та стимулюють глибше занурення у гру за рахунок нових цілей, можливостей кастомізації та взаємодії з улюбленими істотами гри – котами.

Таким чином, розроблена модифікація є технічно стабільною, функціонально завершеною та придатною до використання гравцями, зацікавленими в новому ігровому досвіді в межах добре знайомого середовища Minecraft.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було розроблено програмну модифікацію до гри Minecraft з використанням середовища Forge для версії 1.21.1, що суттєво розширює функціонал гри шляхом додавання нових механік, пов'язаних з взаємодією з котами, предметами, бронєю та рослинами. Для цього було виконано ряд завдань:

- проаналізовано особливості архітектури Minecraft та обмеження системи з урахуванням середовища модифікації Forge;
- сформульовано концепцію модифікації, яка враховує ігровий баланс, цілісність стилістики та ігрові сценарії взаємодії;
- обрано інструменти розробки, такі як Java, Forge API, середовище IntelliJ IDEA, а також побудовано власний генератор даних (клас DataGenerators);
- реалізовано та протестовано функціональні компоненти модифікації, включаючи нові предмети, блоки, броню, механіки видобутку ресурсів і взаємодії з тваринами.

Розроблений застосунок був реалізований повністю вручну, без використання сторонніх модів, бібліотек чи кодів спільноти — за виключенням офіційної документації Forge, що гарантує унікальність та авторський підхід до проєкту.

Застосунок було протестовано в реальному ігровому середовищі, з використанням стандартного клієнта Minecraft і офіційного Forge, без виявлення критичних помилок або конфліктів з базовим функціоналом гри.

Застосунок реалізує такі функціональні вимоги:

- отримання кігтів з котів або через видобуток викопної руди;
- створення нових предметів: кігтінатор, інструменти, броня, блок кігтів;
- взаємодія з новою рослиною – котячою м'ятою, та створення на її основі зілля «екстракт котячої м'яти»;

– додавання броні з унікальними властивостями, яка змінює ігрову поведінку мобів і взаємодію гравця з навколишнім середовищем.

В результаті розробки реалізовано повноцінне ігрове доповнення, що гармонійно інтегрується в ігровий процес Minecraft, зберігаючи ігровий баланс, додаючи нові цікаві механіки та розширюючи можливості гравця, зокрема в аспекті кастомізації, добування ресурсів та соціальної взаємодії з мобами.

Усі завдання дипломної кваліфікаційної роботи бакалавра повністю виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Minecraft Home Page [Electronic resource]. – Access mode: <https://www.minecraft.net>.
2. Forge MDK Version 1.21.1 Page [Electronic resource]. – Access mode: https://www.files.minecraftforge.net/net/minecraftforge/forge/index_1.21.1.html.
3. CurseForge Cats Mod Search Results [Electronic resource]. – Access mode: <https://www.curseforge.com/minecraft/search?page=1&Size=20&sortBy=relevancy&search=cats>.
4. Mark's Wolf And Cat textures (16x16) [Electronic resource]. – Access mode: <https://www.curseforge.com/minecraft/texture-packs/marks-wolf-and-cat-textures>.
5. Build Battle Cat Model [Electronic resource]. – Access mode: <https://www.curseforge.com/minecraft/texture-packs/build-battle-cat-models>.
6. [TEMPLATE] Music Resource Pack For Easy Modification [Electronic resource]. – Access mode: <https://www.curseforge.com/minecraft/template-music-resource-pack-for-easy-modification>.
7. Forge API Version 1.21.1 Documentation Home Page [Electronic resource]. – Access mode: <https://www.docs.minecraftforge.net/en/1.21.x/>.
8. Draw.io [Electronic resource]. – Access mode: <https://www.drawio.com>.
9. Minecraft Fandom Wiki [Electronic resource]. – Access mode: <https://www.minecraft.fandom.com/wiki/Fossil>.
10. IntelliJ IDEA Community Edition Install [Electronic resource]. – Access mode: <https://www.jetbrains.com/idea/download/?section=windows>.
11. Pixilart Online Home Page [Electronic resource]. – Access mode: <https://www.pixilart.com/draw?ref=home-page>.
12. Java SE 17 Archive Downloads (17.0.12 and earlier) [Electronic resource]. – Access mode: <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>.

13. CurseForge Just Enough Items (JEI) [Electronic resource]. – Access mode: <https://www.curseforge.com/minecraft/mc-mods/jei>.
14. Minecraft Time Add Command [Electronic resource]. – Access mode: <https://www.minecraft.wiki/w/Commands/time>.
15. Minecraft Fill Command [Electronic resource]. – Access mode: <https://www.minecraft.wiki/w/Commands/Fill>.
16. Aternos [Electronic resource]. – Access mode: <https://www.aternos.org>.

ДОДАТОК А
Технічне завдання

Вступ

Розроблюваний проєкт на тему «Програмна реалізація модифікації для комп'ютерної гри» є користувацьким розширенням функціоналу гри, що додає нову внутрішньоігрову механіку, пов'язану з котами, новими предметами, блоками, інструментами, бронею та кастомними ефектами. Основною метою створення модифікації є забезпечення гравців новим досвідом у взаємодії з ігровим світом, збереження атмосфери оригінальної гри та розширення геймплейних можливостей без порушення балансу.

Актуальність такого проєкту пояснюється стабільно високою популярністю Minecraft серед гравців різного віку та зростаючим попитом на якісний модифікований контент, що не тільки урізноманітнює гру, а й поглиблює її механіки. Запропонована модифікація також стимулює гравців до дослідження світу, взаємодії з котами, створення унікального спорядження, а також дозволяє реалізувати творчий підхід до кастомізації свого персонажа.

Очікуваним результатом є створення повноцінної модифікації, яка повністю інтегрується у оригінальну гру, є стабільною, не потребує сторонніх залежностей, та розширює ігрові можливості через нові механіки, предмети і ефекти, при цьому залишаючись зручною у використанні як для новачків, так і для досвідчених гравців.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмна реалізація модифікації для комп'ютерної гри», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

А.2 Основні вимоги до програми, що розробляється

А.2.1 Вимоги до функціональних характеристик

Функціональні вимоги є основою при створенні програмного забезпечення, оскільки саме вони визначають, які дії користувач може виконувати за допомогою модифікації, та як реагуватиме на них програма. Чітко сформульовані функціональні вимоги дозволяють забезпечити стабільну роботу модифікації, логічну взаємодію елементів гри та загальну відповідність очікуванням гравця. У процесі розробки, налагодження та тестування модифікації ці характеристики були орієнтиром для перевірки якості реалізації геймплейних механік.

Функціональні можливості модифікації повинні забезпечувати реалізацію таких дій користувача:

- отримання ресурсу «кіготь» із ванільних котів з обмеженням у часі (раз на 5 ігрових днів);
- знаходження руди «блок викопних кігтів» на висотах 32–64 блоки, з можливістю видобутку предмету або блоку залежно від інструмента;
- крафт предметів (кігтіатор, блок кігтів, тощо) та подальше їх використання (взаємодія з мобами, використання в рецептах, тощо);
- взаємодія з предметом кігтіатором, що дозволяє добути з kota шерсть (одиниця на використання);
- використання шерсті в крафті кастомної броні, що має спеціальні ефекти та властивості;
- отримання насіння котячої м'яти з торгівлі з мандрівним крамарем, його вирощування, збір ресурсів (насіння, квітка) та подальше використання;
- створення зілля з ефектом «заспокоєння» з котячої м'яти, яке тимчасово зменшує лякливість котів;
- використання інструментів і зброї, еквівалентних діамантовим за характеристиками, з підвищеним шансом кращих зачарувань;

- носіння броні з унікальними ефектами: зменшення урону від падіння, прискорення пересування крадькома, зменшення агресії мобів, захист від кріперів шляхом їх відлякування;
- взаємодія з новими блоками для декору, крафту та переплавки;
- забезпечення стабільної роботи модифікації в межах оригінального геймплею без конфліктів із основними механіками гри;
- використання системи DataGen для автоматичної генерації даних ресурсів без необхідності ручного створення JSON-файлів;
- підтримка шаблонів для броні з кастомними 3D-моделями, що підвищують рівень персоналізації персонажа.

A.2.2 Вимоги до інтерфейсу програми

Хоча більшість функцій розробленої модифікації не потребує окремого графічного інтерфейсу, важливим є дотримання принципу інтеграції нових елементів у вже наявний ігровий інтерфейс Minecraft таким чином, щоб користувацький досвід залишався звичним, зрозумілим та інтуїтивно простим. Інтерфейс повинен бути максимально природним, органічно поєднуватися з ванільною стилістикою гри та не створювати в користувача враження "зовнішньої" або чужорідної модифікації.

Основні вимоги до інтерфейсу:

- відсутність зайвих елементів. Усі механіки реалізовані без потреби у графічному GUI – крафти використовують стандартну верстакову систему, ефекти та можливості броні працюють автоматично при її носінні, ресурси здобуваються через взаємодію з ігровими об'єктами;
- використання стандартної системи крафту. Усі нові рецепти розміщуються у системі крафту гри та зберігають логіку розташування ресурсів, аналогічну до оригінальної;
- іконки та текстури предметів. Усі предмети, блоки, броня та інструменти мають унікальні іконки та текстури, створені у стилістиці

Minecraft, що дозволяє гравцю легко розрізняти нові об'єкти без порушення візуального балансу гри;

- кастомні 3D-моделі броні. Інтерфейс інвентаря коректно відображає усі елементи броні та їхній зовнішній вигляд на персонажі, забезпечуючи підтримку шаблонів для кастомізації;

- ефекти та повідомлення. Накладання ефектів (заспокоєння, нудота, сліпотата) супроводжується стандартною системою статусів та візуалізацій у HUD, що зберігає послідовність із ванільним ігровим процесом, також присутній вибір однієї з локалізації модифікації, а саме української та англійської;

- відображення інформації. Гравець може побачити наслідки взаємодії з модифікаційними об'єктами без необхідності додаткових вікон чи меню – через зміни в стані об'єкта, повідомлення в чаті або графічні індикатори.

У результаті реалізований інтерфейс повністю відповідає стилю Minecraft, не перевантажує гравця новими вікнами або елементами та забезпечує зручну й природну взаємодію з усіма компонентами модифікації.

A.2.3 Вимоги до надійності

Надійність є критично важливою характеристикою для модифікацій, які інтегруються у складну та динамічну систему, якою є ігровий рушій Minecraft. Враховуючи, що модифікація працює на основі Forge API, її стабільність та коректність мають забезпечуватися на рівні як коду, так і взаємодії з ігровим світом, іншими об'єктами та можливими зовнішніми чинниками.

Основні вимоги до надійності включають:

- безпечне підключення модифікації до гри. Після інсталяції мод не повинен викликати збоїв при запуску Minecraft або конфліктів з іншими модами, які не змінюють аналогічний функціонал;

- обробка виняткових ситуацій. Усі основні дії (взаємодія з блоками, предметами, створення об'єктів, генерація структури) повинні супроводжуватися перевітками на валідність, щоб уникнути помилок під час гри;
- сумісність з оновленнями гри. Структура коду побудована таким чином, щоб зменшити ризики несумісності після оновлень Forge або гри, зокрема шляхом використання DataGen для генерації JSON-файлів та максимальної відповідності сучасним стандартам модифікацій;
- стійкість до некоректних дій гравця. Модифікація не повинна спричиняти помилки навіть у разі нестандартної поведінки користувача, наприклад, спроби використання предметів у невідповідному контексті або руйнування нестандартних блоків;
- протестованість механік. Усі функціональні можливості були протестовані вручну у тестовому середовищі на предмет стабільності, повторюваності результатів, а також взаємодії з ванільною механікою гри;
- відсутність втрати даних. У результаті використання модифікації не повинно виникати жодного пошкодження збережень гри чи втрати прогресу гравця.

Завдяки дотриманню цих вимог, розроблена модифікація демонструє високу стабільність під час роботи та гарантує безпечний ігровий досвід без критичних помилок.

A.2.4 Вимоги до складу та параметрів технічний засобів

Для коректної роботи розробленої модифікації необхідне дотримання певних вимог до апаратного та програмного забезпечення. Оскільки модифікація створена для гри Minecraft Java Edition з використанням Forge API, її використання передбачає наявність персонального комп'ютера з достатньою обчислювальною потужністю, що дозволяє стабільно запускати гру разом із

додатковим програмним навантаженням, а також, бажано, зі стабільним підключенням до інтернету, для коректного інсталювання всіх необхідних програмних компонентів.

Мінімальні вимоги до технічних засобів:

- процесор: 4-ядерний процесор з тактовою частотою не нижче 2.5 ГГц;
- оперативна пам'ять: 8 Гб (рекомендується виділення не менше 4 Гб оперативної пам'яті для Java-машини гри);
- відеокарта: Відеокарта з підтримкою OpenGL 4.5;
- накопичувач: SSD або HDD з не менше 2 Гб вільного місця для розміщення гри, Forge, модифікацій і збережень;
- операційна система: Windows 10 / 11, Linux або macOS (64-bit);
- версія Java: Java SE Development Kit 17 або сумісний з Minecraft версії 1.21.1.

Рекомендовані вимоги для комфортної гри з модифікацією:

- процесор: 6-ядерний процесор з частотою від 3.0 ГГц;
- оперативна пам'ять: 16 Гб;
- відеокарта: NVIDIA GeForce GTX 1650 / AMD Radeon RX 580 або вище;
- накопичувач: SSD для прискореного запуску гри та зменшення часу завантаження світу;
- інтернет-з'єднання: для завантаження Forge, модифікації, а також можливості участі у мультиплеєрі (опціонально).

Важливо відзначити, що модифікація не є ресурсоємною, але за рахунок додаткових моделей, текстур та скриптів потребує трохи більше ресурсів, ніж оригінальна версія гри. Тому на слабших системах можуть спостерігатися незначні просадки FPS під час активної взаємодії з об'єктами модифікації.

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Розроблена програма за темою «Програмна реалізація модифікації для комп'ютерної гри» призначена для розширення ігрового досвіду користувачів через інтеграцію нових предметів, блоків, броні, ресурсів, інтерфейсних елементів і механік. Модифікація дозволяє поглибити систему крафту, додати нові можливості взаємодії з тваринами (зокрема котами), а також візуально урізноманітнити гру завдяки унікальним 3D-моделям і текстурам.

Програма розроблена для використання на платформі Minecraft Java Edition 1.21.1 з API Forge, та орієнтована на гравців, які прагнуть кастомізувати ігровий світ і отримати нові геймплейні можливості без втрати сумісності з основною версією гри. Модифікація не вимагає зміни ядра гри, що забезпечує зручність встановлення та сумісність з іншими модами.

Модифікація має зручну структуру ресурсів і даних, підтримує автоматичну генерацію рецептів, лут-таблиць та моделей завдяки системі DataGen, а також забезпечує якісний ігровий досвід за рахунок кастомних ефектів, логіки штучного інтелекту й візуальних покращень.

Б.2 Функціональне призначення

Розроблена модифікація призначена для розширення базового функціоналу гри Minecraft шляхом додавання нових ігрових механік, предметів, ресурсів, інструментів, броні та інтерактивних елементів. Основною метою є покращення користувацького досвіду через запровадження унікального способу взаємодії з котами, нових можливостей крафту, добування ресурсів, а також кастомізації персонажа.

Модифікація забезпечує реалізацію наступних функціональних можливостей:

- отримання ресурсу «кігті» з котів або шляхом добування спеціальної руди;

- створення нового інструменту – кігтізатора, який дозволяє добувати шерсть із котів;
- крафт унікальних інструментів і броні з новими ефектами та візуальним оформленням;
- вирощування рослини – котячої м'яти, та створення на її основі зілля заспокоєння для полегшеного приручення котів;
- додавання кастомних ефектів до броні (захист від падіння, зменшення агресії мобів, ефект маскуванню гравця під kota для криперів, тощо);
- застосування сучасних засобів генерації даних (DataGen) для зручності розробки та масштабування.

Таким чином, модифікація розширює можливості гри, не порушуючи її оригінального балансу, й спрямований на гравців, зацікавлених у новому, але природному для ігрового світу, контенті.

Б.3 Опис логічної структури

Логічна структура розробленої модифікації побудована за модульним принципом, що забезпечує розмежування відповідальності між окремими компонентами програми та спрощує підтримку, тестування й масштабування проєкту.

Кожен логічний компонент виконує визначене коло функцій:

- базові ресурси (ресурсні файли): містять інформацію про текстури, моделі, мови, рецепти крафту, лут таблиці, властивості блоків і предметів. Ці дані формуються частково вручну, а частково автоматизовано через DataGen для зменшення ймовірності помилок та пришвидшення розробки.
- логіка предметів та блоків: окремі класи реалізують функціональність для кожного предмета, зокрема кігтів, кігтізатора, блоку кігтів, шерсті, а також інструментів і елементів броні. Вони містять методи для обробки взаємодії гравця з предметом, властивості міцності, ефекти та реакції на події гри.

- спеціалізовані механіки: окремі обробники (providers) що відповідають за генерування руди у світі, періодичну можливість випадіння кігтів з котів, застосування кігтізатора до котів, впливу кастомних ефектів на ігрові події гравця та ШІ істот;

- інтеграція з ванільною грою: реалізовано підтримку шаблонів для броні, адаптовану генерацію руди у світі з дотриманням балансу, а також можливість використання предметів у вже наявних системах гри без конфліктів.

- тестування та діагностика: мод передбачає логування ключових подій, що дозволяє відстежувати виконання функцій, а також оперативно виявляти й усувати потенційні помилки.

Такий підхід до структурування логіки модифікації забезпечує її гнучкість, надійність і сумісність із подальшими оновленнями гри або іншими модами.

Б.4 Використані технічні засоби

У процесі розробки програмного забезпечення використовувався стандартний набір апаратного та програмного забезпечення, що забезпечив ефективну реалізацію, тестування та запуск модифікації гри Minecraft.

Апаратне забезпечення:

- процесор: Intel(R) Core(TM) i5-8400H CPU, 2.5 ГГц (6 ядер / 6 потоків);
- оперативна пам'ять: 16 Гб DDR4;
- накопичувач: SSD 256 Гб;
- відеокарта: NVIDIA GeForce GTX 1050 Ti;
- операційна система: Windows 10 (64-bit).

Програмне забезпечення:

- Java Development Kit (JDK) 17 – основне середовище виконання та компіляції коду для Minecraft 1.21.1;

- IntelliJ IDEA – інтегроване середовище розробки, яке використовувалось для написання та відлагодження коду;
- Minecraft Forge 1.21.1 MDK – фреймворк для створення модифікацій гри;
- Minecraft Java Edition – середовище для безпосереднього запуску й тестування модифікації;
- Pixelart online – онлайн графічний редактор для створення та обробки текстур;
- Gradle – система автоматичної збірки модифікації;
- Minecraft Mod Testing Environment (IDE run config) – налаштоване середовище запуску гри безпосередньо з середовища розробки.

Використання зазначених технічних засобів дозволило забезпечити стабільну, контрольовану й гнучку розробку модифікації, а також полегшило її тестування та налагодження.

ДОДАТОК В
Текст програми

В.1 Файл Diplomod.java

```

package diplomod;

import com.mojang.logging.LogUtils;
import diplomod.blocks.ModBlocks;
import diplomod.datagen.DataGenerators;
import diplomod.effects.ModEffects;
import diplomod.items.ModCreativeModTabs;
import diplomod.items.ModItems;
import net.minecraft.client.renderer.RenderType;
import net.minecraft.world.item.CreativeModeTabs;
import net.minecraftforge.api.distmarker.Dist;
import net.minecraftforge.common.MinecraftForge;
import net.minecraftforge.event.BuildCreativeModeTabContentsEvent;
import net.minecraftforge.event.server.ServerStartingEvent;
import net.minecraftforge.eventbus.api.IEventBus;
import net.minecraftforge.eventbus.api.SubscribeEvent;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.fml.event.lifecycle.FMLClientSetupEvent;
import net.minecraftforge.fml.event.lifecycle.FMLCommonSetupEvent;
import net.minecraftforge.fml.javafmlmod.FMLJavaModLoadingContext;
import org.slf4j.Logger;

import static net.minecraft.client.renderer.ItemBlockRenderTypes.setRenderLayer;

@Mod(DiploMod.MOD_ID)
public class DiploMod
{
    public static final String MOD_ID = "diplomod";

    public static final Logger LOGGER = LogUtils.getLogger();

    public DiploMod(FMLJavaModLoadingContext context)
    {
        IEventBus bus = context.getModEventBus();

        bus.addListener(this::commonSetup);
        MinecraftForge.EVENT_BUS.register(this);
        MinecraftForge.EVENT_BUS.register(DataGenerators.class);

        ModCreativeModTabs.register(bus);
        ModEffects.register(bus);
        ModItems.register(bus);
        ModBlocks.register(bus);

        bus.addListener(this::addCreative);
    }

```

```

private void commonSetup(final FMLCommonSetupEvent event)
{

}

private void addCreative(BuildCreativeModeTabContentsEvent event)
{
    if (event.getTabKey() == CreativeModeTabs.INGREDIENTS) {
        event.accept(ModItems.CLAW);
        event.accept(ModItems.CAT_NIP);
        event.accept(ModItems.CAT_FUR);
    }
    if (event.getTabKey() == CreativeModeTabs.NATURAL_BLOCKS) {
        event.accept(ModItems.CAT_NIP_SEEDS);
        event.accept(ModItems.CAT_NIP);
    }
    if (event.getTabKey() == CreativeModeTabs.FOOD_AND_DRINKS) {
        event.accept(ModItems.CAT_NIP);
        event.accept(ModItems.CAT_NIP_POTION);
    }
    if (event.getTabKey() == CreativeModeTabs.TOOLS_AND_UTILITIES){
        event.accept(ModItems.CLAWMINATOR);
        event.accept(ModItems.CLAW_AXE);
        event.accept(ModItems.CLAW_PICKAXE);
        event.accept(ModItems.CLAW_SHOVEL);
        event.accept(ModItems.CLAW_HOE);
    }
    if (event.getTabKey() == CreativeModeTabs.COMBAT){
        event.accept(ModItems.CLAW_HELMET);
        event.accept(ModItems.CLAW_CHESTPLATE);
        event.accept(ModItems.CLAW_LEGGINGS);
        event.accept(ModItems.CLAW_BOOTS);
        event.accept(ModItems.CLAW_SWORD);
        event.accept(ModItems.CLAW_AXE);
    }
    if (event.getTabKey() == CreativeModeTabs.BUILDING_BLOCKS) {
        event.accept(ModBlocks.CLAW_BLOCK);
        event.accept(ModBlocks.CLAW_FOSSIL_BLOCK);
    }
}

@SubscribeEvent
public void onServerStarting(ServerStartingEvent event)
{

}

@Mod.EventBusSubscriber(modid = MOD_ID, bus = Mod.EventBusSubscriber.Bus.MOD,
value = Dist.CLIENT)
public static class ClientModEvents
{

```

```

    @SubscribeEvent
    public static void onClientSetup(FMLClientSetupEvent event)
    {
        setRenderLayer(ModBlocks.CAT_NIP_CROP.get(), RenderType.cutout());
    }
}

```

B.2 Файл DataGenerators.java

```

@Mod.EventBusSubscriber(modid = DiploMod.MOD_ID, bus =
Mod.EventBusSubscriber.Bus.MOD)
public class DataGenerators {
    @SubscribeEvent
    public static void gatherData(GatherDataEvent event){
        DataGenerator generator = event.getGenerator();
        PackOutput packOutput = generator.getPackOutput();
        ExistingFileHelper existingFileHelper = event.getExistingFileHelper();
        CompletableFuture<HolderLookup.Provider> lookupProvider = event.getLookupProvider();

        generator.addProvider(event.includeServer(), new LootTableProvider(packOutput,
Collections.emptySet(),
        List.of(new LootTableProvider.SubProviderEntry(ModBlockLootTableProvider::new,
        LootContextParamSets.BLOCK)), lookupProvider));
        generator.addProvider(event.includeServer(), new ModRecipeProvider(packOutput,
lookupProvider));

        generator.addProvider(event.includeClient(), new ModItemModelProvider(packOutput,
existingFileHelper));
        generator.addProvider(event.includeClient(), new ModBlockStateProvider(packOutput,
existingFileHelper));
        BlockTagsProvider blockTagsProvider = new ModBlockTagProvider(packOutput,
lookupProvider, existingFileHelper);
        generator.addProvider(event.includeServer(), blockTagsProvider);
        generator.addProvider(event.includeServer(), new ModItemTagProvider(packOutput,
lookupProvider, blockTagsProvider.contentsGetter(), existingFileHelper));

        generator.addProvider(event.includeServer(), new ModWorldGenProvider(packOutput,
lookupProvider));
    }
}

```

B.3 Файл ModBlockStateProvider.java

```

public class ModBlockStateProvider extends BlockStateProvider {
    public ModBlockStateProvider(PackOutput output, ExistingFileHelper exFileHelper){
        super(output, DiploMod.MOD_ID, exFileHelper);
    }
}

```

```

@Override
protected void registerStatesAndModels() {
    blockWithItem(ModBlocks.CLAW_BLOCK);
    blockWithItem(ModBlocks.CLAW_FOSSIL_BLOCK);

    makeCatNipCrop((CropBlock) CAT_NIP_CROP.get(), "cat_nip_stage", "cat_nip_stage");
}

public void makeCatNipCrop(CropBlock block, String modelName, String textureName){
    Function<BlockState, ConfiguredModel[]> function = state -> cat_nip_states(state, block,
modelName, textureName);

    getVariantBuilder(block).forAllStates(function);
}

private ConfiguredModel[] cat_nip_states(BlockState state, CropBlock block, String
modelName, String textureName){
    ConfiguredModel[] models = new ConfiguredModel[1];
    models[0] = new ConfiguredModel(models().crop((modelName +
state.getValue(((CatNipCropBlock) block).getAgeProperty())),
    ResourceLocation.fromNamespaceAndPath(DiploMod.MOD_ID, "block/" +
textureName + state.getValue(((CatNipCropBlock) block).getAgeProperty()))));
    return models;
}

private void blockWithItem(RegistryObject<Block> blockRegistryObject){
    simpleBlockWithItem(blockRegistryObject.get(), cubeAll(blockRegistryObject.get()));
}
}

```

В.4 Файл ModItemModelProvider.java

```

public class ModItemModelProvider extends ItemModelProvider {

    private static LinkedHashMap<ResourceKey<TrimMaterial>, Float> trimMaterials = new
LinkedHashMap<>();
    static {
        trimMaterials.put(TrimMaterials.QUARTZ, 0.1F);
        trimMaterials.put(TrimMaterials.IRON, 0.2F);
        trimMaterials.put(TrimMaterials.NETHERITE, 0.3F);
        trimMaterials.put(TrimMaterials.REDSTONE, 0.4F);
        trimMaterials.put(TrimMaterials.COPPER, 0.5F);
        trimMaterials.put(TrimMaterials.GOLD, 0.6F);
        trimMaterials.put(TrimMaterials.EMERALD, 0.7F);
        trimMaterials.put(TrimMaterials.DIAMOND, 0.8F);
        trimMaterials.put(TrimMaterials.LAPIS, 0.9F);
        trimMaterials.put(TrimMaterials.AMETHYST, 1.0F);
    }
}

```

```

public ModItemModelProvider(PackOutput output, ExistingFileHelper existingFileHelper){
    super(output, DiploMod.MOD_ID, existingFileHelper);
}

@Override
protected void registerModels() {
    basicItem(ModItems.CLAW.get());
    basicItem(ModItems.CLAWMINATOR.get());
    basicItem(ModItems.CAT_FUR.get());
    trimmedArmorItem(ModItems.CLAW_HELMET);
    trimmedArmorItem(ModItems.CLAW_CHESTPLATE);
    trimmedArmorItem(ModItems.CLAW_LEGGINGS);
    trimmedArmorItem(ModItems.CLAW_BOOTS);
    handheldItem(ModItems.CLAW_SWORD);
    handheldItem(ModItems.CLAW_AXE);
    handheldItem(ModItems.CLAW_PICKAXE);
    handheldItem(ModItems.CLAW_SHOVEL);
    handheldItem(ModItems.CLAW_HOE);
    basicItem(ModItems.CAT_NIP_SEEDS.get());
    basicItem(ModItems.CAT_NIP.get());
    basicItem(ModItems.CAT_NIP_POTION.get());
}

private void trimmedArmorItem(RegistryObject<ArmorItem> itemRegistryObject) {
    final String MOD_ID = DiploMod.MOD_ID;

    if(itemRegistryObject.get() instanceof ArmorItem armorItem) {
        trimMaterials.forEach((trimMaterial, value) -> {
            float trimValue = value;

            String armorType = switch (armorItem.getEquipmentSlot()) {
                case HEAD -> "helmet";
                case CHEST -> "chestplate";
                case LEGS -> "leggings";
                case FEET -> "boots";
                default -> «««;
            };

            String armorItemPath = armorItem.toString();
            String trimPath = "trims/items/" + armorType + "_trim_" +
trimMaterial.location().getPath();
            String currentTrimName = armorItemPath + "_" + trimMaterial.location().getPath() +
"_trim";
            ResourceLocation armorItemResLoc = ResourceLocation.parse(armorItemPath);
            ResourceLocation trimResLoc = ResourceLocation.parse(trimPath);
            ResourceLocation trimNameResLoc = ResourceLocation.parse(currentTrimName);

            existingFileHelper.trackGenerated(trimResLoc, PackType.CLIENT_RESOURCES,
".png", "textures");

            getBuilder(currentTrimName)
                .parent(new ModelFile.UncheckedModelFile("item/generated"))

```

```

        .texture("layer0", armorItemResLoc.getNamespace() + ":item/" +
armorItemResLoc.getPath())
        .texture("layer1", trimResLoc);

        this.withExistingParent(itemRegistryObject.getId().getPath(),
            mcLoc("item/generated"))
            .override()
            .model(new ModelFile.UncheckedModelFile(trimNameResLoc.getNamespace() +
":item/" + trimNameResLoc.getPath()))
            .predicate(mcLoc("trim_type"), trimValue).end()
            .texture("layer0",
                ResourceLocation.fromNamespaceAndPath(MOD_ID,
                    "item/" + itemRegistryObject.getId().getPath()));
    });
}
}

private ItemModelBuilder handheldItem(RegistryObject<Item> item) {
    return withExistingParent(item.getId().getPath(),
        ResourceLocation.parse("item/handheld")).texture("layer0",
        ResourceLocation.fromNamespaceAndPath(DiploMod.MOD_ID, "item/" +
item.getId().getPath()));
}
}

```

B.5 Файл CatClawDropper.java

```

@Mod.EventBusSubscriber(modid = DiploMod.MOD_ID)
public class CatClawDropper {
    private static final int DAYS_INTERVAL = 5;
    private static final String SPAWN_DAY_TAG = "SpawnDay";

    @SubscribeEvent
    public static void onEntityJoinLevel(EntityJoinLevelEvent event) {
        Entity entity = event.getEntity();
        if (entity instanceof Cat cat && event.getLevel() instanceof ServerLevel serverLevel) {
            long currentDay = serverLevel.getDayTime() / 24000;

            if (!cat.getPersistentData().contains(SPAWN_DAY_TAG)) {
                cat.getPersistentData().putLong(SPAWN_DAY_TAG, currentDay);
            }
        }
    }

    @SubscribeEvent
    public static void onLevelTick(TickEvent.LevelTickEvent event) {
        if (event.level instanceof ServerLevel serverLevel && event.phase ==
TickEvent.Phase.END) {
            long currentDay = serverLevel.getDayTime() / 24000;

```

```

for (Entity entity : serverLevel.getEntities().getAll()) {
    if (entity instanceof Cat cat) {
        CompoundTag data = cat.getPersistentData();

        if (data.contains(SPAWN_DAY_TAG)) {
            long spawnDay = data.getLong(SPAWN_DAY_TAG);

            if (currentDay - spawnDay >= DAYS_INTERVAL) {
                Random random = new Random();
                int clawsToDrop = 3 + random.nextInt(3);
                cat.spawnAtLocation(new ItemStack(ModItems.CLAW.get(), clawsToDrop));

                data.putLong(SPAWN_DAY_TAG, currentDay);
            }
        }
    }
}

```

В.6 Файл ModWorldGenProvider.java

```

public class ModWorldGenProvider extends DatapackBuiltinEntriesProvider {

    public static final RegistrySetBuilder BUILDER = new RegistrySetBuilder()
        .add(Registries.CONFIGURED_FEATURE, ModConfiguredFeatures::bootstrap)
        .add(Registries.PLACED_FEATURE, ModPlacedFeatures::bootstrap)
        .add(ForgeRegistries.Keys.BIOME_MODIFIERS, ModBiomeModifiers::bootstrap);

    public ModWorldGenProvider(PackOutput output,
        CompletableFuture<HolderLookup.Provider> registries){
        super(output, registries, BUILDER, Set.of(DiploMod.MOD_ID));
    }
}

```

В.7 Файл ArmorEffect.java

```

@Mod.EventBusSubscriber(modid = DiploMod.MOD_ID)
public class ArmorEvent {
    @SubscribeEvent
    public static void onPlayerTick(TickEvent.PlayerTickEvent event) {
        if (event.phase != TickEvent.Phase.START) return;

        Player player = event.player;
    }
}

```

```

    if (isWearingHelmet(player)) {
        player.level().getEntitiesOfClass(Creeper.class, player.getBoundingBox().inflate(10))
            .forEach(creeper -> {
                creeper.setTarget(null);
                if (creeper.distanceTo(player) < 10) {
                    AvoidEntityGoal<Player> avoidGoal = new AvoidEntityGoal<>(creeper,
Player.class, 6.0F, 1.0, 1.2);
                    if (!creeper.goalSelector.getAvailableGoals().contains(avoidGoal)) {
                        creeper.goalSelector.addGoal(1, avoidGoal);
                    }
                }
            });
    } else {
        player.level().getEntitiesOfClass(Creeper.class, player.getBoundingBox().inflate(10))
            .forEach(creeper -> {
                creeper.setTarget(player);
                creeper.goalSelector.getAvailableGoals().stream()
                    .filter(wrappedGoal -> wrappedGoal != null && wrappedGoal.getGoal()
instanceof AvoidEntityGoal)
                    .forEach(wrappedGoal -> {
                        if (wrappedGoal != null) {
                            creeper.goalSelector.removeGoal(wrappedGoal.getGoal());
                        }
                    });
            });
    }
}

```

```

    if (isWearingChestplate(player)) {
        player.level().getEntities(player, player.getBoundingBox().inflate(32), entity -> entity
instanceof Mob)
            .forEach(entity -> {
                Mob mob = (Mob) entity;
                if (mob.getTarget() == player) {
                    AttributeInstance followRangeAttr =
mob.getAttribute(Attributes.FOLLOW_RANGE);
                    double defaultAggroRange = followRangeAttr != null ?
followRangeAttr.getValue() : 16.0;
                    double reducedAggroRange = defaultAggroRange / 2.0;

                    if (mob.distanceTo(player) > reducedAggroRange) {
                        mob.setTarget(null);
                    }
                }
            });
    }
}

```

```

AttributeInstance movementSpeed = player.getAttribute(Attributes.SNEAKING_SPEED);
if (isWearingLeggings(player) && player.isCrouching() && movementSpeed != null) {
    movementSpeed.setBaseValue(0.475);
} else if (movementSpeed != null) {
    movementSpeed.setBaseValue(0.3);
}

```

```

    }

    if (isWearingBoots(player)){
        Level level = player.level();
        BlockState blockUnderPlayer = level.getBlockState(player.blockPosition().below());
        if (blockUnderPlayer.is(Blocks.POWDER_SNOW)) {
            level.setBlock(player.blockPosition().below(),
Blocks.SNOW_BLOCK.defaultBlockState(), 3);
        }
    }

    if (isWearingBoots(player) && player.fallDistance > 0) {
        player.fallDistance *= 0.5;
    }
}

private static boolean isWearingHelmet(Player player) {
    return player.getItemBySlot(EquipmentSlot.HEAD).getItem() ==
ModItems.CLAW_HELMET.get();
}

private static boolean isWearingChestplate(Player player) {
    return player.getItemBySlot(EquipmentSlot.CHEST).getItem() ==
ModItems.CLAW_CHESTPLATE.get();
}

private static boolean isWearingLeggings(Player player) {
    return player.getItemBySlot(EquipmentSlot.LEGS).getItem() ==
ModItems.CLAW_LEGGINGS.get();
}

private static boolean isWearingBoots(Player player) {
    return player.getItemBySlot(EquipmentSlot.FEET).getItem() ==
ModItems.CLAW_BOOTS.get();
}
}

```

В.8 Файл ClawminatorItem.java

```

public class ClawminatorItem extends ShearsItem {
    public ClawminatorItem(Properties properties) {
        super(properties);
    }

    @Override
    public InteractionResult interactLivingEntity(ItemStack stack, Player player, LivingEntity
target, InteractionHand hand) {
        Level level = player.level();

        if (!level.isClientSide && target instanceof Cat) {

```

```

player.addItem(new ItemStack(ModItems.CAT_FUR.get()));
if(!player.isCreative()) {
    stack.setDamageValue(stack.getDamageValue() + 1);

    if (stack.getDamageValue() >= stack.getMaxDamage()) {
        if (player.level() instanceof ServerLevel serverLevel) {
            serverLevel.playSound(null, player.getX(), player.getY(), player.getZ(),
SoundEvents.ITEM_BREAK, player.getSoundSource(), 1.0F, 1.0F);
        }
        player.setItemInHand(hand, ItemStack.EMPTY);
    }
}
return InteractionResult.SUCCESS;
}
return super.interactLivingEntity(stack, player, target, hand);
}
}

```

В.9 Файл CalmEffect.java

```

public class CalmEffect extends MobEffect {
    public CalmEffect(MobEffectCategory mobEffectCategory, int color) {
        super(mobEffectCategory, color);
    }

    @Override
    public boolean applyEffectTick(LivingEntity pLivingEntity, int pAmplifier) {
        if (!pLivingEntity.level().isClientSide()) {
            pLivingEntity.level().getEntitiesOfClass(Cat.class,
pLivingEntity.getBoundingBox().inflate(10))
                .forEach(cat -> {
                    cat.getNavigation().stop();
                    cat.setTarget(null);
                });
        }
        return true;
    }

    @Override
    public boolean shouldApplyEffectTickThisTick(int pDuration, int pAmplifier) {
        return true;
    }
}

```

ДОДАТОК Д
Слайди презентації

ПРОГРАМНА РЕАЛІЗАЦІЯ МОДИФІКАЦІЇ ДЛЯ КОМП'ЮТЕРНОЇ ГРИ

КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ НУ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
АВТОР - СТУДЕНТ ГРУПИ КНТ-121 ЯН ГАВРИЛОВ
КЕРІВНИК – Д-Р. ФІЛОСОФІЇ, ДОЦЕНТ СЕРГІЙ ЛЕОЩЕНКО

Рисунок Д.1 – Слайд 1

ОБ'ЄКТ, ПРЕДМЕТ ТА МЕТА РОБОТИ

Об'єкт дослідження — процес розробки модифікації для комп'ютерної гри MINECRAFT.

Предмет дослідження — засоби розробки модифікації для комп'ютерної гри MINECRAFT.

Метою роботи є дослідження засобів та безпосередньо розробка модифікації для популярної комп'ютерної гри MINECRAFT.

2

Рисунок Д.2 – Слайд 2

ПОРІВНЯННЯ МОВ ПРОГРАМУВАННЯ

Характеристика	Minecraft Java Edition (ПК)	Minecraft Bedrock Edition (мобільні пристрої, Windows Store)
Мова програмування	Java	C++
Гнучкість модифікації	Висока (через відкритість і велику кількість API)	Низька (обмежена система аддонів)
Продуктивність	Нижча (вища ресурсоемність)	Вища (оптимізований нативний код)
Кросплатформеність	Обмежена (лише ПК)	Висока (ПК, мобільні пристрої, консолі)
Спільнота модифікаторів	Дуже активна, великий обсяг модів	Менш активна, обмеження з боку платформи
Інструменти для розробки модифікацій	Forge, Fabric, MCP, тощо	Add-ons, Minecraft Creator Tool

3

Рисунок Д.3 – Слайд 3

ПОСТАНОВКА ЗАДАЧІ

ОСНОВНОЮ МЕТОЮ РОБОТИ БУЛО СТВОРЕННЯ КОРИСТУВАЦЬКОЇ МОДИФІКАЦІЇ ДЛЯ ГРИ MINECRAFT, ЯКА Б РОЗШИРЮВАЛА ФУНКЦІОНАЛЬНІ МОЖЛИВОСТІ КОТІВ, РОБЛЯЧИ ЇХ БІЛЬШ ІНТЕРАКТИВНИМИ, КОРИСНИМИ ТА ІЦКАВИМИ ДЛЯ ГРАВЦЯ.

Для досягнення цієї мети необхідно було насамперед проаналізувати існуючі модифікації, що стосуються котів. В результаті цього аналізу було виявлено їхні обмеження, недостатню кількість механік та відсутність поведінкових змін.

Далі потрібно було дослідити технічні можливості реалізації нових механік у межах вибраної версії гри з використанням Forge API, для чого була проведена ретельна робота по аналізу технічних нововведень обраної версії API, а також функціональних меж та можливостей цієї версії.

ЗАВЕРШАЛЬНИМИ ЕТАПАМИ РОБОТИ СТАЛИ ТЕСТУВАННЯ МОДИФІКАЦІЇ, УСУНЕННЯ БАГІВ, ІНТЕГРАЦІЯ НОВИХ МЕХАНІК У ВЖЕ НАЯВНІ ІГРОВІ СИСТЕМИ БЕЗ ПОРУШЕННЯ БАЛАНСУ, ПЕРЕВІРКА НА СУМІСНІСТЬ З ІНШИМИ КОРИСТУВАЦЬКИМИ МОДИФІКАЦІЯМИ, А ТАКОЖ ПІДГОТОВКА ДОКУМЕНТАЦІЇ ЩОДО ВСТАНОВЛЕННЯ Й ВИКОРИСТАННЯ МОДУ.

4

Рисунок Д.4 – Слайд 4



Рисунок Д.5 – Слайд 5



Рисунок Д.6 – Слайд 6



Рисунок Д.7 – Слайд 7

ВИБІР ЗАСОБІВ РОЗРОБКИ

Критерій	Вибране технічне рішення
Мова програмування	Java, оскільки саме на ній написана оригінальна версія Minecraft для комп'ютерів.
Інтегроване середовище розробки (IDE)	IntelliJ IDEA, що забезпечує зручну роботу з Java-проєктами, має розширену підтримку Gradle-білдів та інструментів Forge.
Інструменти білду та запуску	Gradle, який дозволяє ефективно управляти залежностями, компіляцією та пакуванням модифікації у формат .jar.
Генерація JSON-файлів	DataGen, з використанням власноруч створеного класу DataGenerators, що дозволило уникнути ручного редагування конфігурацій та мінімізувати ризики помилок. Такий підхід повністю автономний і не залежав від сторонніх інструментів.
Документація та підтримка	Єдиним зовнішнім джерелом, яке було використано у процесі розробки, була офіційна документація Forge.

8

Рисунок Д.8 – Слайд 8

ВИЗНАЧЕННЯ СЦЕНАРІЇВ ВЗАЄМОДІЇ КОРИСТУВАЧА З МОДИФІКАЦІЄЮ

Модифікація була створена з метою поглиблення взаємодії гравця з котами в MINECRAFT, зробивши цих мобів не лише декоративними, а й функціонально корисними впродовж ігрового процесу. У рамках реалізованої концепції було закладено систему пов'язаних між собою ресурсів, предметів, блоків та механік, що формують новий шлях розвитку гравця – «шлях кота».

Усі ключові об'єкти модифікації взаємодіють між собою у межах циклічної економіки ресурсів: нові предмети добуваються, переробляються, об'єднуються в нові інструменти та броню, і мають як стратегічну, так і тактичну цінність у бою, дослідженні світу, або виживанні. В основі системи – ресурс «кіготь», який можна отримати як шляхом видобутку, так і шляхом взаємодії з котами.

Важливою складовою є й розвиток сільськогосподарського напрямку – вирощування котячої м'яти як спеціального ресурсу, що дозволяє впливати на поведінку котів і відкривати додаткові можливості взаємодії з ними. Нові ефекти, які додаються через споживання або використання певних предметів, створюють унікальний досвід та стимулюють експерименти з комбінаціями ресурсів.

Гравець самостійно приймає рішення, у який спосіб розвивати «котячу гілку» – зосередитись на броні, інструментах чи на вирощуванні рослин, що забезпечує гнучкість та індивідуалізацію ігрового стилю.

9

Рисунок Д.9 – Слайд 9

ВЗАЄМОДІЯ З ПРЕДМЕТАМИ МОДИФІКАЦІЇ



10

Рисунок Д.10 – Слайд 10

ВЗАЄМОДІЯ З БЛОКАМИ МОДИФІКАЦІЇ



11

Рисунок Д.11 – Слайд 11

РОЗРОБЛЕНІ ЕФЕКТИ БРОНІ

Елемент броні	Ефект
Шолом	Змушує ШІ кріперів боятися гравця так само, як вони бояться котів – вони починають тікати.
Нагрудник	Зменшення радіусу агресії ШІ мобів на гравця вдвічі (наприклад, скелет помічає гравця не з 16, а з 8 блоків).
Штані	Прискорення руху в режимі крадькома, але не вище базової швидкості. При наявності зачарування «Біг крадькома II» – швидкість руху в режимі крадькома дорівнює звичайній.
Чоботи	Зменшення шкоди від падіння вдвічі, утримовування рихлого снігу під ногами в звичайний блок.
Будь-яка частина броні	Захист від замерзання в рихлому снігу.

12

Рисунок Д.12 – Слайд 12

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ



13

Рисунок Д.13 – Слайд 13

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ



14

Рисунок Д.14 – Слайд 14



Рисунок Д.15 – Слайд 15

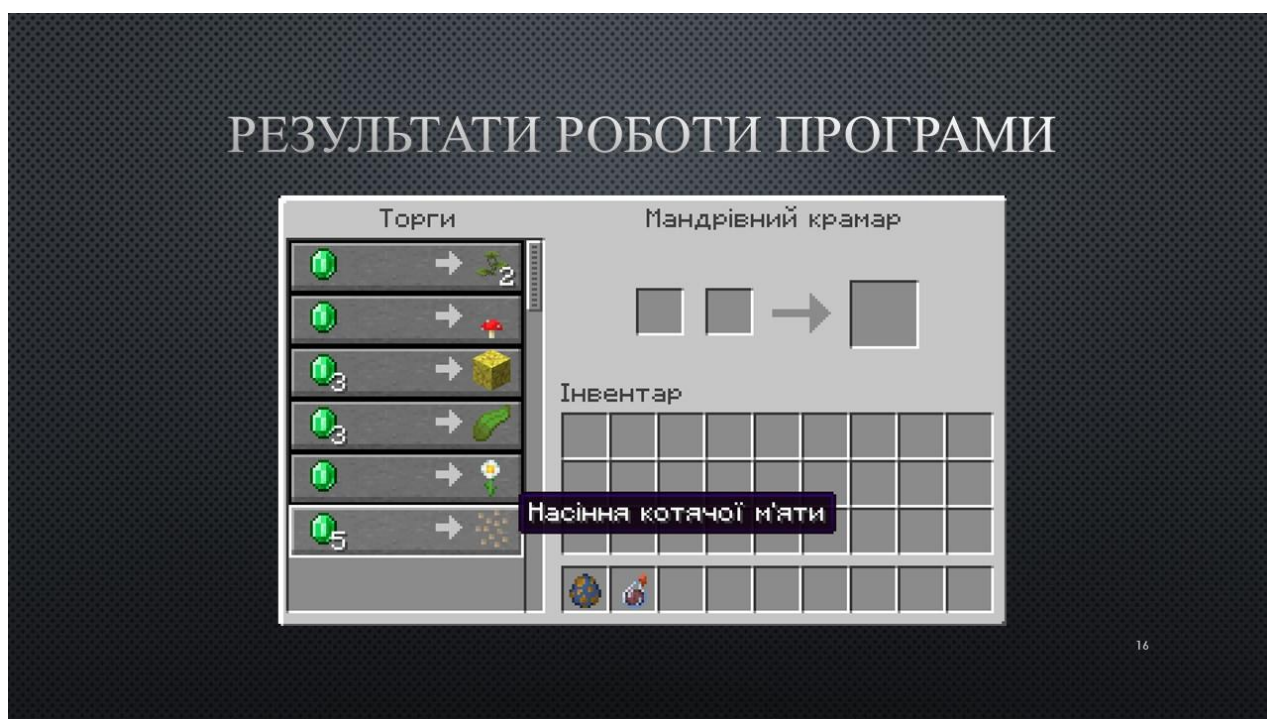


Рисунок Д.16 – Слайд 16

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ



17

Рисунок Д.17 – Слайд 17

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ



18

Рисунок Д.18 – Слайд 18

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ



19

Рисунок Д.19 – Слайд 19

РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ



20

Рисунок Д.20 – Слайд 20

ВИСНОВКИ

У ХОДІ ВИКОНАННЯ ДИПЛОМНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА БУЛО РОЗРОБЛЕНО МОДИФІКАЦІЮ ДО ГРИ MINECRAFT, ЩО СУТТЄВО РОЗШИРЮЄ ФУНКЦІОНАЛЬНІСТЬ ПРИРУЧЕНИХ КОТІВ, ПЕРЕТВОРЮЮЧИ ЇХ З ДЕКОРАТИВНИХ МОБІВ НА ПОВНОЦІННИХ СУПУТНИКІВ У ГРІ. ДЛЯ ДОСЯГНЕННЯ ЦЬОЇ МЕТИ БУЛО ВИКОНАНО РЯД ЗАВДАНЬ: ПРОВЕДЕНО АНАЛІЗ ІСНУЮЧИХ МОДИФІКАЦІЙ, ВИЗНАЧЕНО ЇХНІ ОБМЕЖЕННЯ, СФОРМУЛЬОВАНО КОНЦЕПЦІЮ РОЗШИРЕННЯ ПОВЕДІНКИ КОТІВ, РЕАЛІЗОВАНО НОВІ ІГРОВІ МЕХАНІКИ, ПРЕДМЕТИ, РЕСУРСИ ТА ЕФЕКТИ, ЩО БЕЗПОСЕРЕДНЬО ПОВ'ЯЗАНІ З ЦИМИ МОБАМИ.

РОЗРОБЛЕНИЙ МОД УСПІШНО РЕАЛІЗОВУЄ УСІ ПОСТАВЛЕНІ ФУНКЦІОНАЛЬНІ ЗАВДАННЯ, ВІДПОВІДАЄ ВИМОГАМ СУМІСНОСТІ, СТАБІЛЬНОСТІ ТА ІГРОВОЇ ЗБАЛАНСОВАНOSTІ. ЗАВДЯКИ ГЛИБОКІЙ ІНТЕГРАЦІЇ З ІГРОВОЮ ЛОГІКОЮ, ПІДТРИМЦІ АКТУАЛЬНОЇ ВЕРСІЇ ГРИ ТА НОВИЗНІ РЕАЛІЗОВАНИХ ІДЕЙ, МОДИФІКАЦІЯ МАЄ ЗНАЧНИЙ ПОТЕНЦІАЛ ДЛЯ ПОПУЛЯРИЗАЦІЇ СЕРЕД ГРАВЦІВ, А ТАКОЖ ПОДАЛЬШОГО РОЗВИТКУ З РОЗШИРЕННЯМ ГЕЙМПЛЕЙНИХ МОЖЛИВОСТЕЙ У МАЙБУТНЬОМУ.

Рисунок Д.21 – Слайд 21