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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для управління конфігураціями гри World of Tanks»

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

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

_____ Арсен СОЛОВ'ЯН
(підпис)

Виконав: здобувач вищої освіти групи ПД-44

Арсен СОЛОВ'ЯН

Керівник: Ігор ГАМАНЮК
ст. викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Солов'яну Арсену Олександровичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для управління конфігураціями гри World of Tanks»
керівник кваліфікаційної роботи ст. викладач кафедри ІПЗ Ігор ГАМАНЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: технічна документація гри World of Tanks, специфікація конфігураційного файлу preferences.xml, теоретичні відомості та технічна документація щодо мови програмування C++, фреймворку Qt, бібліотеки Pugixml та API Wargaming.net, Google Gemini.

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

1. Аналіз предметної галузі, існуючих засобів керування конфігураціями гри World of Tanks та визначення вимог до програмного забезпечення.
2. Огляд та обґрунтування вибору технологій та інструментальних засобів для реалізації застосунку.

3. Проектування архітектури програмного застосунку та його інтерфейсу користувача.
4. Програмна реалізація основного функціоналу та тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграми варіантів використання.
5. Моделювання функціональності
6. Контекстна діаграма
7. Діаграма класів
8. Діаграма предметної галузі
9. Діаграми послідовності
10. Діаграма послідовності перегляду конфігурації
11. Діаграми діяльності
12. Діаграма станів застосунку
13. Діаграма станів роботи з конфігураційним файлом
14. Діаграма розгортання
15. Логічна архітектура
16. Діаграма компонентів
17. Діаграма артефактів
18. Модульне тестування
19. Результати користувацького тестування
20. Екранні форми.
21. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз застосунку для управління конфігураціями	14.03-21.03.2025	
4	Огляд та аналіз засобів реалізації застосунку для управління конфігураціями	22.03-25.03.2025	
5	Проектування архітектури застосунку для управління конфігураціями	26.03-07.04.2025	
6	Програмна реалізація та тестування застосунку для управління конфігураційними файлами	08.04-28.04.2025	

7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Арсен СОЛОВ’ЯН
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 65 стор., 2 табл., 33 рис., 21 джерел.

Мета роботи – спрощення та підвищення надійності процесу керування конфігураційними файлами гри World of Tanks, а також розширення можливостей гравців шляхом інтеграції доступу до актуальної ігрової статистики та довідкової інформації.

Об'єкт дослідження – процес управління користувацькими конфігураціями та доступу до супутньої ігрової інформації у грі World of Tanks.

Предмет дослідження – методи та програмні засоби для автоматизації та спрощення процесу керування конфігураційними файлами гри World of Tanks та інтеграції додаткових інформаційних сервісів.

Короткий зміст роботи: В роботі досліджено проблематику керування конфігураційним файлом preferences.xml гри World of Tanks та проаналізовано існуючі підходи та інструменти для роботи з налаштуваннями гри. Визначено функціональні та нефункціональні вимоги до програмного забезпечення, спрямованого на вирішення виявлених проблем. Обґрунтовано вибір стеку технологій, включаючи мову програмування C++, фреймворк Qt для розробки графічного інтерфейсу, бібліотеку Pugixml для парсингу XML та інтеграцію з API Wargaming.net та Google Gemini. Спроектовано архітектуру застосунку з розподілом на логічні шари. Реалізовано ключові функціональні можливості застосунку, такі як: збереження та відновлення резервних копій конфігураційного файлу, редагування налаштувань через графічний інтерфейс, валідація файлу, пошук статистики гравців та взаємодія з AI-помічником. Проведено тестування розробленого застосунку для підтвердження його працездатності та відповідності вимогам.

Сферою використання застосунку є персональне використання гравцями гри World of Tanks для зручного та безпечного керування своїми ігровими налаштуваннями та доступу до додаткової ігрової інформації.

КЛЮЧОВІ СЛОВА: WORLD OF TANKS, КОНФІГУРАЦІЯ, PREFERENCES.XML, УПРАВЛІННЯ НАЛАШТУВАННЯМИ, РЕЗЕРВНЕ КОПІЮВАННЯ, QT, C++, WARGAMING API, GOOGLE GEMINI API.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment

API – Application Programming Interface

XML – Extensible Markup Language

WOT – World of Tanks

UI – User Interface

FPS – Frames per second

ВСТУП

Актуальність: У світі комп'ютерних ігор, зокрема в такій популярній масовій багатокористувацькій онлайн-грі як World of Tanks, тонке налаштування ігрового клієнта відіграє значну роль у забезпеченні комфортного і ефективного ігрового процесу. Параметри графіки, звуку, інтерфейсу та управління, що зберігаються у конфігураційному файлі preferences.xml, суттєво впливають на продуктивність гри, візуальне сприйняття та зручність взаємодії для кожного окремого гравця. Незважаючи на наявність внутрішньо ігрового меню налаштувань, воно надає доступ лише до обмеженого набору параметрів [1]. Більш глибоке налаштування часто вимагає прямого редагування файлу preferences.xml. Ручне редагування цього XML-файлу є процесом, що вимагає специфічних технічних знань, уважності та несе високий ризик внесення синтаксичних помилок, що може призвести до некоректної роботи або скидання всіх налаштувань гри. Крім того, відсутні зручні вбудовані засоби для легкого створення резервних копій, відновлення попередніх конфігурацій чи простого обміну налаштуваннями між гравцями. Існуючі сторонні інструменти, такі як модпаки чи застарілі "твікери", не надають комплексного та безпечного вирішення всіх цих проблем. Таким чином, існує актуальна потреба у розробці спеціалізованого програмного забезпечення, яке б спростило процес керування конфігураціями гри World of Tanks, мінімізувало ризики ручного втручання та розширило можливості гравців за рахунок інтеграції додаткових сервісів.

Об’єктом дослідження є процес управління користувацькими конфігураціями та доступу до супутньої ігрової інформації у грі World of Tanks.

Метою роботи є спрощення та підвищення надійності процесу керування конфігураційними файлами гри World of Tanks для широкого кола гравців, а також розширення їх можливостей шляхом надання інтегрованого доступу до актуальної ігрової статистики та довідкової інформації.

Методи дослідження: Для досягнення поставленої мети в роботі були використані наступні методи дослідження: проведено аналіз предметної галузі, що включає дослідження структури конфігураційного файлу preferences.xml та виявлення проблем і потреб користувачів, здійснено огляд та порівняльний аналіз існуючих інструментальних засобів для керування налаштуваннями World of Tanks. На основі аналізу обґрунтовано вибір технологій та інструментів розробки, зокрема мови програмування C++, фреймворку Qt для побудови графічного інтерфейсу, бібліотеки Pugixml для роботи з XML, а також API Wargaming.net та Google Gemini для інтеграції зовнішніх сервісів. Виконано проектування архітектури програмного застосунку, здійснено програмну реалізацію ключових функціональних модулів застосунку та проведено тестування розробленого програмного забезпечення.

Наукова новизна роботи полягає у розробці комплексного програмного застосунку для управління конфігураціями World of Tanks, який, на відміну від існуючих рішень, об'єднує в собі зручний графічний інтерфейс для редагування файлу preferences.xml з функціями безпечного резервного копіювання, відновлення та валідації конфігурацій, а також інтегрує доступ до актуальної ігрової статистики гравців через Wargaming API та функціонал інтерактивного помічника на базі Google Gemini API.

Практична значущість результатів полягає у створенні функціонального та зручного у використанні програмного застосунку "WOT Configurator", який дозволяє гравцям World of Tanks легко та безпечно керувати своїми ігровими налаштуваннями, уникнути типових помилок при ручному редагуванні конфігураційного файлу, швидко створювати та відновлювати резервні копії, а також отримувати доступ до своєї ігрової статистики та довідкової інформації безпосередньо через інтерфейс застосунку. Використання розробленого застосунку спростить процес налаштування гри, заощадить час користувачів та підвищить загальний комфорт від гри.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ГРИ WORLD OF TANKS

1.1 Програмне забезпечення для управління конфігураціями гри World of Tanks

Програмне забезпечення для управління конфігураціями гри World of Tanks – це спеціалізований інструмент, призначений для спрощення та підвищення безпеки процесу налаштування ігрового клієнта World of Tanks. На відміну від вбудованого в гру меню налаштувань, такі застосунки надають розширені можливості для роботи з конфігураційним файлом preferences.xml, де зберігається більшість параметрів гри. Вони допомагають користувачам тонко адаптувати візуальні, звукові, управлінські та інші аспекти гри, часто надаючи доступ до прихованих або неочевидних параметрів.

Основна мета такого програмного забезпечення полягає у забезпеченні ефективного, зручного та безпечного способу керування складними налаштуваннями гри, що дозволяє гравцям оптимізувати ігровий процес відповідно до своїх уподобань та можливостей комп'ютера без ризику пошкодження ігрових файлів.

Основні компоненти типового програмного забезпечення для управління конфігураціями ігор можуть включати:

– Графічний інтерфейс користувача (GUI): Інтуїтивно зрозумілий інтерфейс, що перетворює ієрархічну структуру конфігураційних файлів на доступні для редагування елементи (перемикачі, поля вводу).

– Модуль роботи з файлами: Функціонал для читання, запису та пошуку конфігураційних файлів у файловій системі користувача.

– Система резервного копіювання та відновлення: Механізм для створення копій поточних налаштувань та можливості повернення до них у разі потреби.

– Функції валідації: Перевірка коректності структури та значень параметрів перед збереженням змін.

– Управління профілями: Можливість зберігати різні набори налаштувань під окремими іменами для швидкого перемикання.

Основні цілі програмного забезпечення для управління конфігураціями гри World of Tanks:

– Спрощення доступу до розширених налаштувань;

– Мінімізація ризиків помилок при редагуванні файлів;

– Забезпечення можливості швидкого резервного копіювання та відновлення;

– Надання зручного інструменту для персоналізації ігрового процесу;

– Підвищення стабільності ігрового клієнта за рахунок коректних налаштувань.

Оцінимо переваги та недоліки спеціалізованого програмного забезпечення як засобу для управління конфігураціями гри World of Tanks.

Переваги:

– Зручність використання: Надає графічний інтерфейс, що усуває необхідність ручного редагування XML-файлів та знання їх структури.

– Безпека: Зменшує ризик внесення синтаксичних помилок та пошкодження файлів, часто включає функції валідації та автоматичного резервного копіювання.

– Доступність розширених налаштувань: Надає доступ до параметрів, які можуть бути приховані у стандартному ігровому меню.

– Економія часу: Спрощує процес налаштування та перемикання між різними конфігураціями.

– Надійність: Розробляється з урахуванням специфіки файлу preferences.xml, що забезпечує коректну взаємодію.

Недоліки:

– Потреба у встановленні: Вимагає встановлення окремого програмного забезпечення на комп'ютер користувача.

– Спеціалізація: Призначене лише для гри World of Tanks.

1.2 Специфіка конфігураційного файлу preferences.xml гри World of Tanks

Ключовим елементом, з яким взаємодіє розроблений застосунок, є конфігураційний файл гри World of Tanks під назвою preferences.xml. Цей файл є основним сховищем користувацьких налаштувань і містить широкий спектр параметрів, що визначають роботу ігрового клієнта, візуальне та звукове оформлення, а також поведінку елементів інтерфейсу та управління.

Специфіка файлу preferences.xml полягає в наступному:

– Формат XML: Файл має структуру XML (Extensible Markup Language) [2], яка базується на використанні тегів для опису даних. XML є стандартним форматом для зберігання та обміну структурованими даними, однак вимагає дотримання суворих синтаксичних правил.

– Ієрархічна структура: Налаштування організовані в ієрархічне дерево елементів, де параметри можуть бути вкладеними один в одного. Це дозволяє групувати пов'язані налаштування, але ускладнює навігацію та розуміння взаємозв'язків при ручному перегляді.

– Різноманіття параметрів: Файл містить велику кількість параметрів різного типу (цілочисельні, дробові, текстові), що відповідають за деталі графічних налаштувань (якість текстур, тіней, ефектів), параметри звуку (гучність, розподіл по каналах), налаштування управління (чутливість миші, призначення клавіш) та елементи інтерфейсу (вигляд прицілу, міні-карти, бойового інтерфейсу).

– Чутливість до помилок: Будь-які синтаксичні помилки у структурі XML, неправильні імена тегів або некоректні значення параметрів можуть призвести до того, що гра не зможе коректно зчитати файл. Це часто спричиняє скидання налаштувань до стандартних значень, некоректне відображення графіки, проблеми зі звуком або навіть аварійне завершення роботи ігрового клієнта.

– Розташування: Файл preferences.xml зазвичай розташований у каталозі даних користувача гри, що може відрізнятися залежно від операційної системи та

способу встановлення гри. Необхідність точного визначення цього шляху є важливою для програмного забезпечення, що працює з файлом.

– Розуміння цієї специфіки є критично важливим для розробки ефективного інструменту управління конфігураціями, який зможе коректно парсити, відображати, редагувати, валідувати та зберігати дані у файлі preferences.xml, забезпечуючи при цьому безпеку та цілісність налаштувань користувача.

1.3 Цільова аудиторія застосунку

Розроблений програмний застосунок для управління конфігураціями гри World of Tanks орієнтований на широке коло гравців цієї популярної онлайн-гри. Цільова аудиторія включає користувачів з різним рівнем технічної підкованості та різними потребами щодо налаштування ігрового клієнта.

До основних категорій користувачів, для яких застосунок буде найбільш корисним, належать:

– Початківці та казуальні гравці: Користувачі, які не мають глибоких знань про структуру файлу preferences.xml та не хочуть ризикувати, редагуючи його вручну. Їм потрібен простий та безпечний спосіб покращити продуктивність гри на своєму обладнанні або змінити базові візуальні налаштування, які не зручно шукати в стандартному меню. Застосунок надає їм інтуїтивно зрозумілий інтерфейс та захищає від випадкових помилок.

– Досвідчені гравці та ентузіасти: Гравці, які прагнуть максимально оптимізувати свій ігровий досвід для досягнення найкращих результатів. Вони можуть потребувати тонкого налаштування параметрів, що впливають на видимість, прицілювання, або інші аспекти геймплею, які доступні лише через preferences.xml. Функції профілів конфігурацій та зручного редагування є важливими для цієї категорії.

– Користувачі з різним апаратним забезпеченням: Гравці як зі слабкими комп'ютерами, які потребують оптимізації налаштувань для досягнення прийнятного рівня FPS (кадрів в секунду), так і власники потужних систем, які

хочуть виставити максимальні графічні параметри. Застосунок спрощує пошук та зміну параметрів, критичних для продуктивності.

– Користувачі, що використовують ігрові модифікації: Гравці, які встановлюють сторонні модифікації та можуть зіштовхнутися з необхідністю коригування preferences.xml для їхньої коректної роботи або вирішення конфліктів налаштувань.

– Будь-який гравець, що цінує зручність та безпеку: Кожен гравець World of Tanks, який бажає мати надійний інструмент для керування своїми налаштуваннями, що дозволяє легко створювати резервні копії перед експериментами та швидко відновлювати робочу конфігурацію у разі виникнення проблем.

Таким чином, цільова аудиторія застосунку є досить широкою і охоплює практично всіх активних гравців World of Tanks, які хоча б зрідка замислюються над оптимізацією або зміною своїх ігрових налаштувань поза рамками стандартного ігрового інтерфейсу.

1.4 Дослідження існуючих рішень для управління конфігураціями World of Tanks

Перед розробкою нового програмного забезпечення був проведений детальний аналіз методів та інструментів, які гравці World of Tanks використовують наразі для управління своїми ігровими налаштуваннями, зокрема конфігураційним файлом preferences.xml. Аналіз існуючих рішень дозволив виявити їхні сильні та слабкі сторони, а також визначити незадоволені потреби користувачів, що стало основою для формування вимог до розроблюваного застосунку.

Основними підходами та інструментами, що використовуються гравцями, є:

1.4.1 Вбудоване ігрове меню налаштувань World of Tanks

Опис та призначення: Це офіційний інструмент, наданий розробниками Wargaming безпосередньо в ігровому клієнті. Дозволяє гравцям змінювати основні параметри гри через графічний інтерфейс [1].

Можливості: Дозволяє налаштовувати ключові параметри графіки (вибір рівня якості, роздільна здатність, вертикальна синхронізація), звуку (загальна гучність, окремі канали), управління (основні налаштування чутливості, інверсія), елементи інтерфейсу (маркери, приціли, інформаційні панелі). Має функцію автовизначення оптимальних налаштувань графіки.

Переваги:

– Простота та доступність для всіх гравців.

– Гарантована сумісність з поточною версією гри.

– Безпечність використання (неможливо ввести критично неправильні значення).

Недоліки:

– Надає доступ лише до обмеженого набору налаштувань порівняно з файлом preferences.xml.

– Відсутня можливість зберігати та завантажувати різні профілі налаштувань.

– Немає функції резервного копіювання чи відновлення конфігурації.

– Немає можливості валідації чи імпорту/експорту налаштувань у вигляді файлу.

– Не надає доступу до статистики гравця чи довідкової інформації поза грою.

Рис. 1.1 Інтерфейс внутрішньо ігрового меню налаштувань World of Tanks

1.4.2 Ручне редагування файлу preferences.xml

Опис та призначення: Пряма модифікація конфігураційного файлу preferences.xml за допомогою будь-якого текстового редактора (Notepad, Notepad++, VS Code тощо).

Можливості: Надає повний доступ до всіх параметрів конфігурації, що зберігаються у цьому файлі, включаючи ті, що не виведені в ігрове меню. Дозволяє копіювати та переносити файл для обміну налаштуваннями.

Переваги:

– Максимальна гнучкість та контроль над усіма доступними налаштуваннями.

– Не потребує встановлення додаткового ПЗ (окрім текстового редактора). Недоліки:

– Вимагає від користувача розуміння структури XML та призначення конкретних тегів/параметрів.

– Дуже високий ризик зробити помилку в синтаксисі XML, що призведе до неможливості читання файлу грою та скидання налаштувань.

– Легко ввести значення, що виходять за допустимі межі, що може спричинити проблеми в грі.

– Відсутні будь-які засоби валідації, резервного копіювання, керування профілями.

– Вкрай незручно для частої зміни або порівняння конфігурацій.

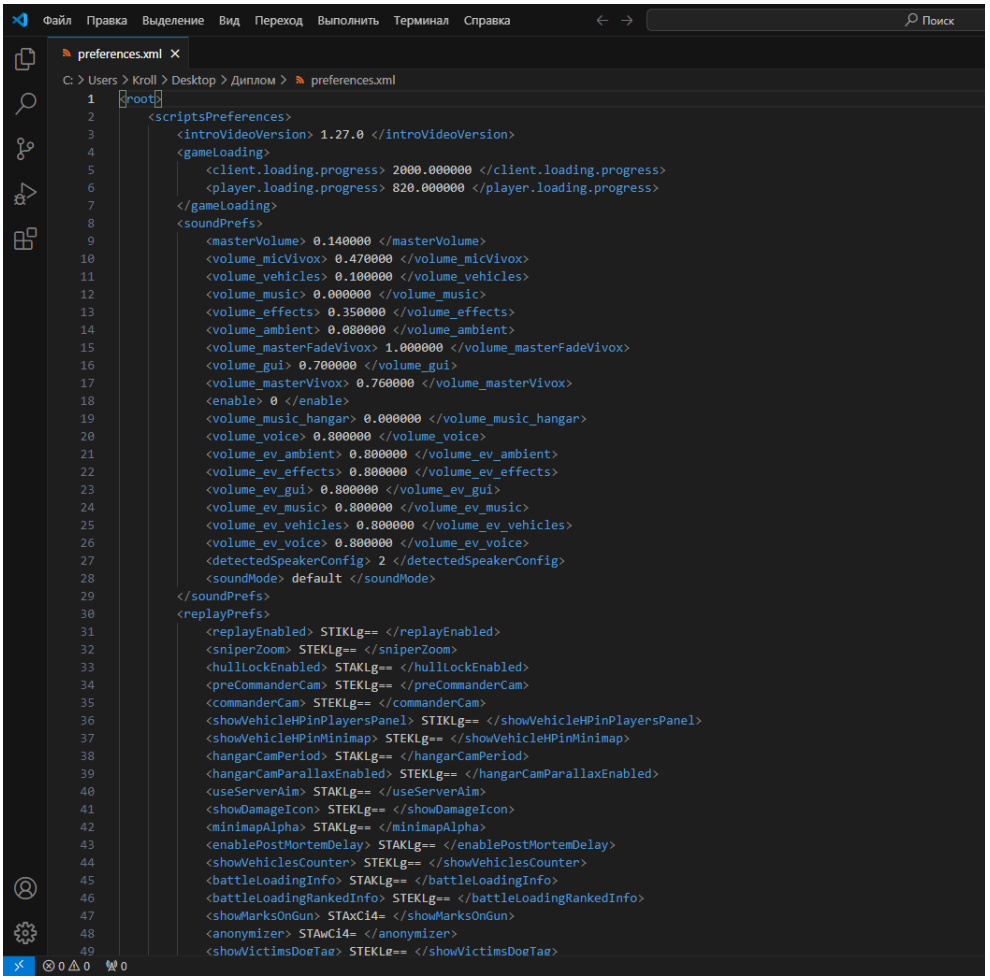


Рис. 1.2 Інтерфейс текстового редактора VSC з відкритим конфігураційним файлом preferences.xml

1.4.3 Модпаки (на прикладі KorbenTeam Modpack)

Опис та призначення: Модпаки є збірками різноманітних сторонніх модифікацій (модів), які змінюють або доповнюють інтерфейс та функціонал гри. KorbenTeam Modpack - один з популярних прикладів у СНД-спільноті [3].

Встановлення та керування модами зазвичай відбувається через спеціальний інсталятор модпаку.

Можливості щодо керування налаштуваннями: Основна функція — встановлення та оновлення модів. Деякі моди всередині модпаку можуть змінювати певні налаштування гри або візуальні аспекти (наприклад, приціли, панелі пошкоджень, XVM), що може опосередковано записувати зміни у preferences.xml.

Однак, сам інсталятор модпаку не є інструментом для детального редагування preferences.xml, його резервного копіювання чи валідації.

Переваги:

– Надають доступ до великої кількості модів в одному місці.

– Зручний процес встановлення та оновлення вибраних модів.

– Можуть містити корисні UI/геймплейні покращення.

Недоліки:

– Не є спеціалізованим інструментом для керування preferences.xml.

– Можливі конфлікти між модами.

– Можуть впливати на продуктивність гри.

– Залежність від оновлень модпаку після кожного патчу гри.

– Відсутні функції резервного копіювання/відновлення та валідації preferences.xml.

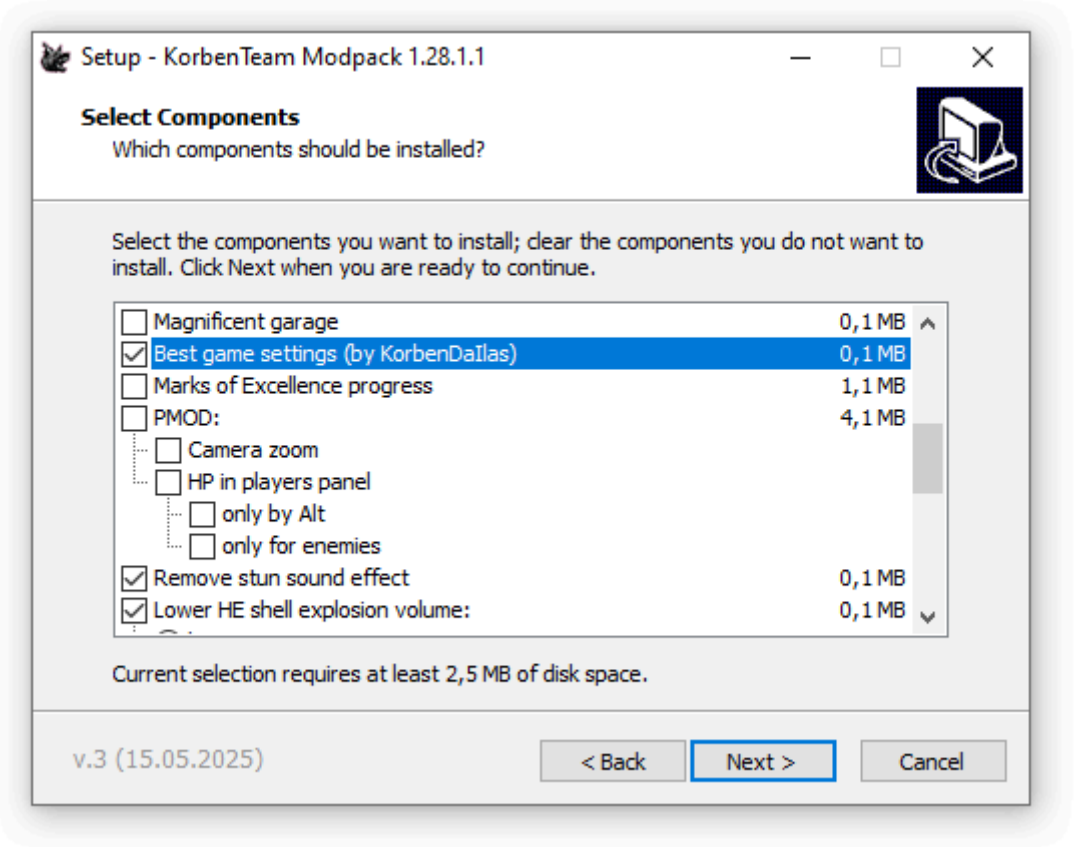


Рис. 1.3 Інтерфейс інсталятора модпаку KorbenTeam Modpack

1.4.4 "Твікери" (на прикладі WoT Tweaker Plus)

Опис та призначення: Історично, "твікери" для WoT були утилітами, призначеними переважно для підвищення частоти кадрів (FPS) на слабких комп'ютерах шляхом відключення або спрощення певних графічних ефектів (дим від пострілів, вибухи, туман, щільність листя тощо). WoT Tweaker та його варіації (як Plus) [4] були популярними раніше.

Можливості щодо керування налаштуваннями: Ці інструменти зазвичай працювали шляхом модифікації не preferences.xml, а файлів ресурсів гри або інших конфігураційних файлів (напр., engine_config.xml). Вони не надавали інтерфейсу для гнучкого налаштування параметрів preferences.xml. Їх мета — вимкнути ефекти, а не налаштувати їх.

Переваги:

– Можливість (у минулому) отримати приріст FPS на дуже слабких системах.

Недоліки:

– Не редагують preferences.xml у потрібному сенсі.

– Можуть бути застарілими та не працювати з сучасними версіями гри через зміни у файловій структурі чи захисті клієнта.

– Використання може вважатися Wargaming небажаним або навіть порушенням правил (залежно від того, що саме модифікується).

– Не надають функцій резервного копіювання, валідації, керування профілями preferences.xml.

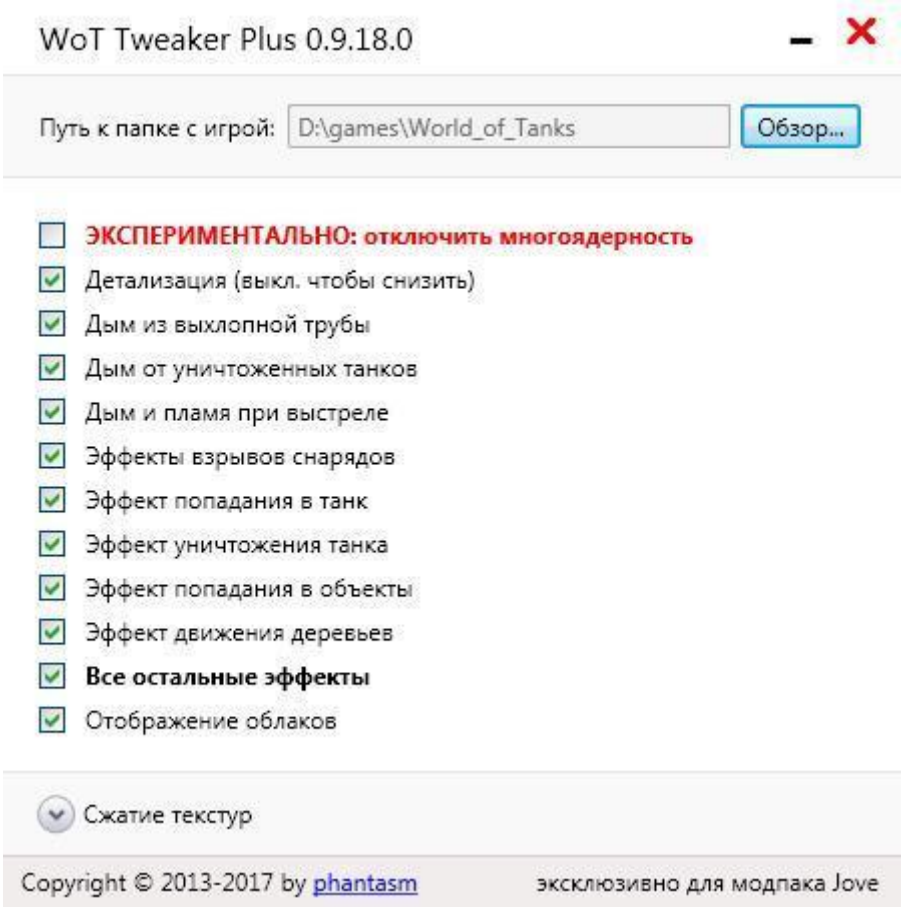


Рис. 1.4 Інтерфейс застосунку WoT Tweaker Plus

1.4.5 WoT Settings & Fixes

Опис та призначення: WoT Settings & Fixes є ще одним прикладом сторонньої утиліти, розробленої спільнотою для полегшення доступу до деяких налаштувань гри. Цей інструмент спрямований на вирішення специфічних

проблем або надання доступу до окремих параметрів, які незручно змінювати іншими способами.

Можливості щодо керування налаштуваннями: На відміну від "твікерів", WoT Settings & Fixes, судячи з його функціоналу, може надавати графічний інтерфейс для відображення та зміни певних поточних налаштувань гри. Це дозволяє користувачам коригувати окремі параметри без прямого редагування XML.

Переваги:

– Надає графічний інтерфейс для деяких налаштувань.

– Може вирішувати конкретні проблеми з налаштуваннями.

Недоліки:

– Не надає повного контролю над усіма параметрами preferences.xml.

– Відсутній функціонал створення, відновлення чи заміни файлу конфігурації.

– Відсутні функції резервного копіювання та валідації.

– Підтримка та актуальність можуть залежати від активності розробника.

Для наочного порівняння можливостей розглянутих інструментів щодо управління конфігураціями гри World of Tanks, наведемо їхню зведену таблицю.

Таблиця 1.1

Порівняння існуючих інструментальних засобів

	WotStarter	KorbenTeam Modpack	Wot Tweaker Plus	WoT Settings & Fixes	WOT Configurator
Заміна на існуючий cfg файл	-	+	-	-	+
Створення резервної копії cfg файлу	-	-	-	-	+

Продовження таблиці 1.1

Порівняння існуючих інструментальних засобів

Відновлення налаштувань з резервної копії cfg файлу	-	-	-	-	+
Відображення поточних налаштувань гри	-	-	-	+	+
Зміна налаштувань гри	+	-	+	+	+

1.5 Визначення вимог до застосунку

Проаналізувавши сутність застосунку, його специфіку, цільову аудиторію та аналогів, було визначено наступні вимоги до застосунку:

– Надавати можливість зберігати поточну конфігурацію гри World of Tanks (файл preferences.xml) у вигляді резервної копії.

– Надавати можливість завантажувати (відновлювати) збережену конфігурацію (резервну копію), замінюючи нею активний файл preferences.xml.

– Автоматично визначати шлях до каталогу встановленої гри World of Tanks.

– Надавати можливість перегляду вмісту файлу конфігурації preferences.xml у структурованому вигляді через графічний інтерфейс.

– Надавати можливість редагування ключових параметрів конфігурації через графічний інтерфейс (без необхідності прямого редагування XML).

– Зберігати зміни, внесені користувачем, у файл preferences.xml.

– Створювати автоматичну резервну копію поточного файлу preferences.xml перед його перезаписом або редагуванням.

– Надавати можливість відновлення конфігурації з раніше створених резервних копій.

– Дозволяти користувачеві зберігати різні конфігурації під унікальними іменами (профілями).

– Перевіряти коректність структури XML файлу preferences.xml перед збереженням змін або завантаженням.

– Надавати доступ до основної статистики гравця через інтеграцію з Wargaming.net Public API.

– Реалізувати функціонал інтелектуального помічника на основі Google Gemini API для надання довідкової інформації про гру та налаштування.

– Забезпечити швидкий, інтуїтивно зрозумілий та адаптивний графічний інтерфейс користувача.

1.6 Висновок аналізу

Аналіз існуючих засобів показує, що жоден з них не надає комплексного, зручного та безпечного рішення для глибокого керування файлом налаштувань preferences.xml гри World of Tanks. Внутрішньо ігрові налаштування є надто обмеженими, ручне редагування небезпечне і незручне, модпаки фокусуються переважно на модифікаціях геймплею/інтерфейсу та не є спеціалізованими менеджерами конфігурацій, а "твікери" є застарілими, ризикованими та не призначені для комплексного керування налаштуваннями preferences.xml. Існують окремі інструменти для перегляду статистики, але вони не інтегровані з процесом налаштування. WoT Settings & Fixes надає певні можливості, але не є повним рішенням для управління файлом конфігурації.

Таким чином, існує обґрунтована потреба у розробці спеціалізованого графічного застосунку, такого як WOT Configurator, який би надавав користувачам зручний інтерфейс для редагування налаштувань, функції резервного копіювання/відновлення, валідації файлів, а також додаткові можливості (наприклад, перегляд статистики), заповнюючи прогалину в існуючих інструментах для гравців World of Tanks. Розроблений застосунок має на меті об'єднати переваги наявних підходів, усунувши їхні ключові недоліки.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

Для успішної розробки програмного забезпечення, що відповідає поставленим вимогам та ефективно вирішує визначені задачі, був обраний та обґрунтований відповідний комплекс інструментальних засобів та технологій. Вибір здійснювався з урахуванням специфіки предметної області (робота з конфігураційними файлами, десктопний застосунок), необхідної продуктивності, зручності розробки та подальшої підтримки.

2.1 Середовище розробки

Інтегровані середовища розробки (IDE) відіграють ключову роль у підвищенні продуктивності та ефективності процесу створення програмного забезпечення, об'єднуючи необхідні інструменти в єдиному інтерфейсі. Вони надають розробнику комплекс інструментів, необхідних для написання, компіляції, відлагодження та тестування коду в єдиному середовищі.

2.1.1 Visual Studio Code

Visual Studio Code (VS Code) є легковаговим, але надзвичайно потужним та гнучким редактором вихідного коду, розробленим Microsoft. Він підтримує велику кількість мов програмування через систему розширень, що робить його придатним для широкого спектру задач розробки. Хоча VS Code сам по собі не є повноцінним IDE у традиційному розумінні, за допомогою відповідних розширень він може надавати функціональність IDE, включаючи підсвічування синтаксису, автодоповнення коду, інтеграцію з налагоджувачами та системами контролю версій [5]. У даному проекті VS Code використовувався як основне середовище для написання коду на C++, завдяки його широкій підтримці мови, зручному інтерфейсу, швидкодії та можливості інтеграції з CMake та налагоджувачами для C++.

2.1.2 Qt Creator

Qt Creator – це крос платформне інтегроване середовище розробки, спеціально створене для розробки програмного забезпечення з використанням фреймворку Qt [6]. Воно надає розробникам інтуїтивно зрозумілі інструменти для дизайну користувацьких інтерфейсів за допомогою Qt Designer, редагування коду, налагодження та збірки проектів на основі системи збірки qmake або CMake.

В рамках розробки застосунку Qt Creator використовувався для проектування графічного інтерфейсу користувача, візуального редагування форм та віджетів, що значно прискорило процес створення UI для роботи з конфігураційними параметрами.

2.1.3 CMake

CMake – це кросплатформенний, незалежний від компілятора інструмент для автоматизації збірки програмного забезпечення [7]. Він використовує прості файли конфігурації (CMakeLists.txt) для генерації файлів збірки (наприклад, makefiles або проектів Visual Studio) для різних середовищ розробки та платформ.

У даному проекті CMake був використаний для управління процесом збірки застосунку на C++ з використанням фреймворку Qt. Це забезпечило гнучкість конфігурації збірки, спростило керування залежностями та зробило процес компіляції відтворюваним на різних системах.

2.2 Мова програмування

Мова програмування слугує фундаментом для написання інструкцій, які безпосередньо виконує комп'ютер, визначаючи його поведінку та функціональність. Вибір мови програмування суттєво впливає на архітектуру програмного забезпечення, продуктивність, зручність розробки та можливості масштабування.

2.2.1 C++

C++ є потужною та високоефективною мовою програмування широкого призначення, яка активно використовується завдяки своїй підтримці різних парадигм, таких як процедурне, об'єктно-орієнтоване та узагальнене програмування. Вона відома своєю високою продуктивністю, близькістю до апаратного забезпечення та можливістю ефективного керування системними ресурсами, включаючи пам'ять [8].

Вибір мови програмування C++ для розробки застосунку управління конфігураціями World of Tanks був обґрунтований необхідністю високої швидкості роботи з файловою системою та ефективної обробки даних конфігураційного файлу preferences.xml. C++ забезпечує низький рівень абстракції, що дозволяє оптимізувати критичні до продуктивності операції, пов'язані з читанням, парсингом та записом XML-даних. Використання стандарту C++17 надало доступ до сучасних можливостей мови.

2.3 Фреймворки та бібліотеки

Фреймворки та бібліотеки є наборами готового коду та інструментів, що надають розробникам реалізацію типових функцій та архітектурних рішень, значно прискорюючи процес розробки та забезпечуючи стандартизацію.

2.3.1 Qt

Qt – це потужний, крос платформний фреймворк для розробки програмного забезпечення з графічним інтерфейсом користувача та без нього. Він надає широкий спектр модулів для роботи з графікою, мережею, базами даних, XML, файловою системою та багатьма іншими аспектами розробки застосунків. Qt написаний на C++ і надає зручний API для створення нативних застосунків [9].

Фреймворк Qt був обраний для розробки застосунку управління конфігураціями завдяки його винятковим можливостям у створенні графічного

інтерфейсу користувача. Qt Designer дозволив візуально спроектувати зручні форми для відображення та редагування параметрів preferences.xml.

Крім того, Qt надав надійні інструменти для роботи з файловою системою, що є критично важливим для читання та запису конфігураційних файлів, а також можливості для здійснення мережевих запитів для інтеграції із зовнішніми API (Wargaming.net Public API, Google Gemini API).

2.3.2 Pugixml

Pugixml – це легка, швидка та компактна бібліотека для обробки XML-даних на C++ [10]. Вона відзначається високою швидкістю парсингу, низьким споживанням пам'яті та простим у використанні API для доступу до елементів та атрибутів XML-документа.

2.4 Система контролю версій

Система контролю версій є фундаментальним компонентом сучасного підходу до розробки програмного забезпечення. Вона дозволяє систематично відстежувати зміни у вихідному коді, забезпечує ефективну співпрацю членів команди над проектом та надає засоби для управління різними версіями програмного продукту.

2.4.1 Git та GitHub

Git — це розподілена система контролю версій, що дозволяє розробникам ефективно керувати еволюцією коду, створювати окремі гілки для паралельної розробки, інтегрувати зміни та повертатися до попередніх станів проекту [11].

GitHub – це веб-сервіс для хостингу репозиторіїв Git, який надає додаткові інструменти для спільної роботи над проектами.

У процесі розробки застосунку "WOT Configurator" система контролю версій Git використовувалася для відстеження всіх змін у вихідному коді. Веб-сервіс GitHub слугував для віддаленого зберігання репозиторію проекту.

3 ПРОЕКТУВАННЯ

Проектування програмного забезпечення є критично важливим етапом життєвого циклу розробки, на якому визначається структура системи, її компоненти, їх взаємодія та розподіл відповідальності. Детальний дизайн забезпечує надійність, масштабованість та зручність подальшої реалізації та підтримки застосунку. У даному розділі представлено архітектуру, логічну та фізичну структури, а також ключові сценарії взаємодії розробленого програмного забезпечення "WOT Configurator".

3.1 Проектування архітектури застосунку

Вибір відповідної архітектури є фундаментальним рішенням, що визначає загальну структуру системи та взаємозв'язок її частин. Для застосунку "WOT Configurator" була обрана багатошарова архітектура, яка забезпечує чітке розділення відповідальності між різними функціональними рівнями [12]. Такий підхід сприяє підвищенню модульності, спрощенню процесу розробки та тестування окремих компонентів, а також полегшує масштабування та подальшу модифікацію системи [13].

Загальна структура архітектури застосунку представлена на діаграмі логічної архітектури:

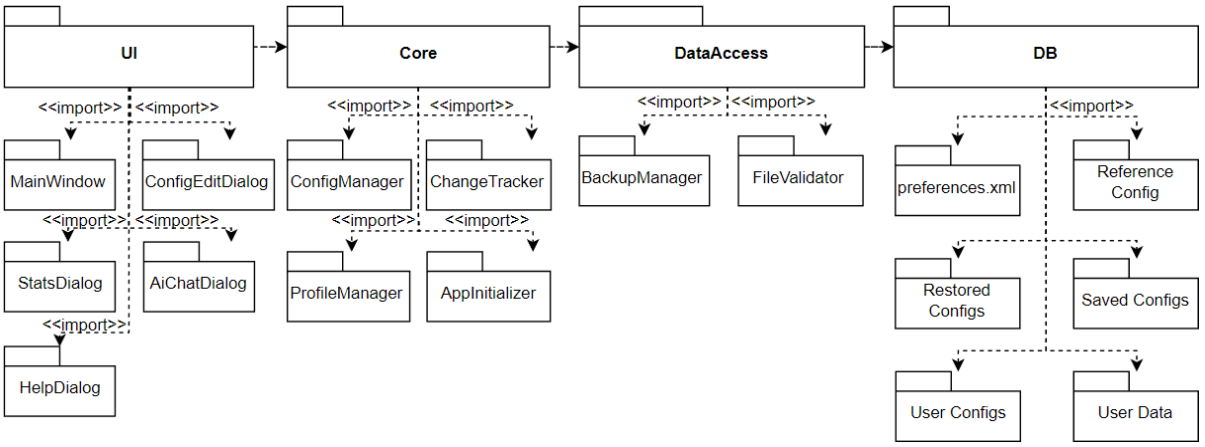


Рис. 3.1 Діаграма логічної архітектури

Вона включає наступні основні шари:

– Шар інтерфейсу користувача (UI): Цей шар є точкою входу для користувача та відповідає за візуалізацію даних, представлення елементів управління та обробку дій користувача. Тут реалізовано головне вікно застосунку (MainWindow) та допоміжні вікна/форми (ConfigEditor, StatsDialog, AiChatDialog, HelpDialog), які забезпечують взаємодію з користувачем та відображають інформацію, отриману від нижчих шарів. Шар UI не містить бізнес-логіки, а лише делегує запити до шару ядра [14].

– Шар ядра (Core): Шар ядра містить основну бізнес-логіку застосунку. Він відповідає за керування процесами, координацію роботи різних модулів та виконання операцій над даними [15]. До цього шару належать такі компоненти, як ConfigManager (керування конфігураціями), ChangeTracker (відстеження змін), ProfileManager (управління профілем) та AppInitializer (ініціалізація застосунку). Шар Core отримує запити від UI, обробляє їх та, за потреби, взаємодіє з шаром доступу до даних.

– Шар доступу до даних (DataAccess): Цей шар є абстракцією над фізичним сховищем даних [16]. Його основне завдання – забезпечити операції читання, запису та управління файлами, що містять конфігурації та резервні копії. До компонентів цього шару належать BackupManager (керування резервними копіями) та FileValidator (валідація файлів). Шар DataAccess отримує запити від шару Core та взаємодіє безпосередньо з файловою системою або іншими механізмами зберігання даних.

– Шар даних (DB/Storage): Хоча застосунок не використовує традиційну систему управління базами даних (СУБД), цей шар представляє фізичне сховище даних, з яким працює шар доступу до даних. Це включає сам файл preferences.xml, каталоги для зберігання резервних копій ("Restored Configs", "Saved Configs"), а також файли, пов'язані з профілями конфігурацій та іншими службовими даними ("User Data", "Reference Config", "User Configs").

Контекстна діаграма ілюструє застосунок "WOT Configurator" як єдину систему та показує його взаємодію із зовнішнім актором – Користувачем, а також з

файлами конфігурації (preferences.xml) та резервними копіями ("Резервні копії") [17]. Ця діаграма надає загальний огляд місця системи в зовнішньому середовищі.

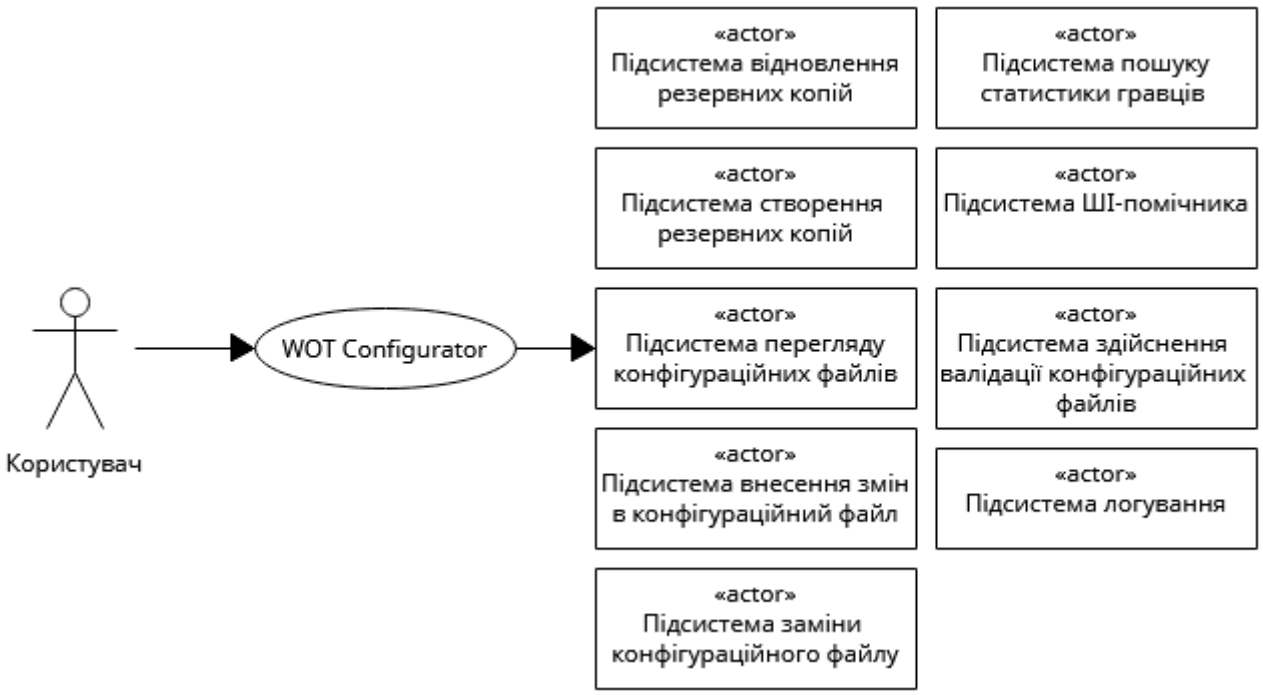


Рис. 3.2 Контекстна діаграма

3.2 Статична модель системи

Статична модель системи розкриває внутрішню структуру застосунку, показуючи, з яких основних "цеглинок" він складається та як ці цеглинки взаємодіють [18]. Це допомагає зрозуміти, як організовані дані та логіка програми для виконання завдань, пов'язаних з управлінням конфігураціями гри World of Tanks.

Для візуалізації ключових сутностей, якими оперує застосунок та які є центральними для процесу налаштування була використана діаграма предметної галузі (Бізнес-модель), що представляє їхню статичну структуру та зв'язки.

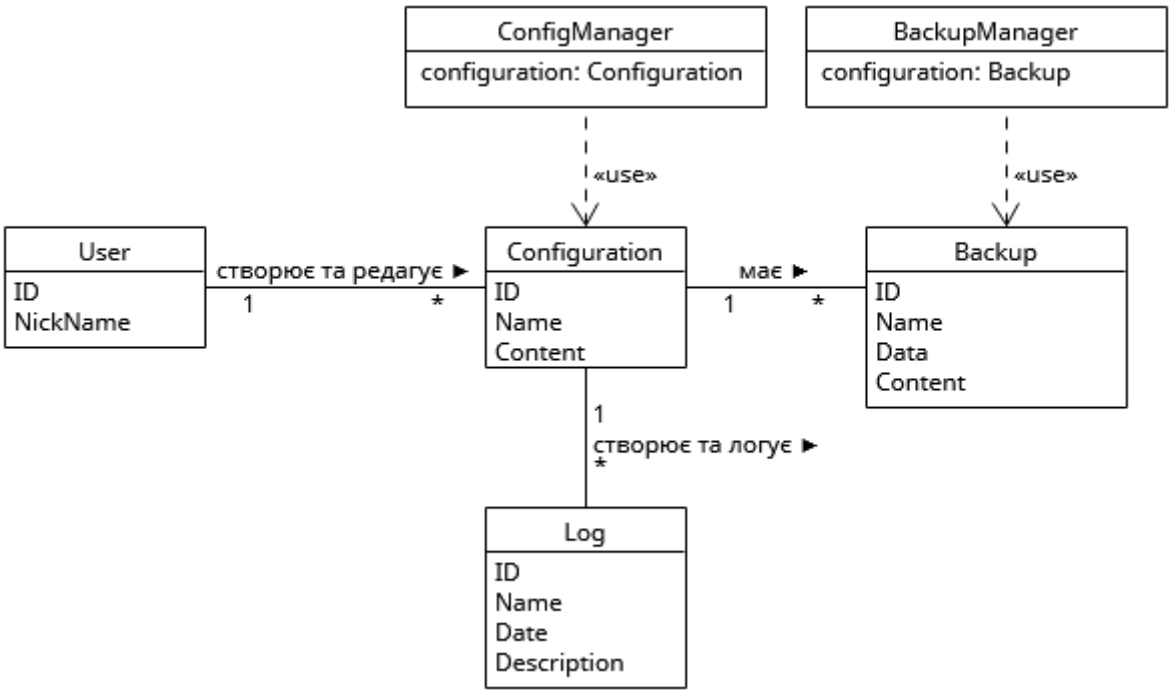


Рис. 3.3 Бізнес-моделі (Діаграма предметної галузі)

На рисунку 3.3 зображені основні елементи, які формують основу даних та логіки застосунку:

ConfigManager: Цей клас є центральною точкою для всіх операцій, пов'язаних із завантаженням та збереженням конфігурацій. Можна уявити його як "адміністратора", який має доступ до файлу preferences.xml і координує роботу з ним. У реалізації він знатиме, де знаходиться файл, та керуватиме процесами його читання та запису.

Configuration: Це сутність, яка представляє власне ігрові налаштування. Кожен об'єкт класу Configuration містить набір параметрів з файлу preferences.xml.

Backup: Цей клас представляє резервну копію конфігурації. Об'єкт Backup зберігає копію налаштувань, має свою ідентифікацію (ID), назву (Name), дату створення (Date) та пов'язаний з тією конфігурацією, з якої він був створений. Кожна Configuration "має" багато Backup, що дозволяє користувачеві створювати декілька точок відновлення для одних і тих самих налаштувань. Це ключовий елемент функціоналу безпеки, який дозволяє користувачеві експериментувати без страху втратити свої налаштування.

Log: Це сутність, яка призначена для фіксації подій та змін, що відбуваються з конфігураціями. Кожен запис у лозі (Log) має ідентифікатор (ID), дату (Date), опис події (Description) та пов'язаний з конкретною конфігурацією, якої стосується ця подія. Одна Configuration "створює та логує" багато записів у лозі. Це надає користувачеві можливість переглядати історію своїх дій, відстежувати, коли і що було змінено, і за потреби, використовувати цю інформацію для відновлення попередніх станів.

Ця діаграма предметної галузі, що фокусується на ключових сутностях, демонструє фундаментальні елементи, навколо яких будується вся логіка управління конфігураціями та резервним копіюванням в застосунку "WOT Configurator". Вона показує, як дані організовані для підтримки таких функцій, як збереження різних наборів налаштувань, їх резервування та відстеження історії змін, що є центральним для зручності та безпеки користувача.

3.3 Динамічна модель системи

Після визначення статичної структури застосунку – тобто з яких частин він складається та як вони організовані – важливо зрозуміти, як ці частини "оживають" і взаємодіють між собою під час роботи програми. Динамічна модель системи показує, як застосунок виконує свої завдання, реагуючи на дії користувача та обробляючи дані в часі [19]. Це дає уявлення про "потоки життя" даних та керування в системі під час виконання ключових функцій.

Одним із головних сценаріїв використання застосунку є зручний перегляд та можливість редагування налаштувань гри World of Tanks. Користувач очікує, що, відкривши програму та вибравши відповідну опцію, він побачить свої поточні ігрові налаштування у зрозумілому вигляді, готовому до внесення змін. За лаштунками, для реалізації цього, застосунок повинен пройти низку кроків: знайти файл preferences.xml, зчитати його вміст, розпарсити складну XML-структуру та представити налаштування у зручному для взаємодії графічному форматі. Цей процес починається з дії користувача в інтерфейсі і

проходить через різні логічні шари застосунку. Діаграма послідовності, що ілюструє цю взаємодію об'єктів у часі, представлена на рисунку 3.4.

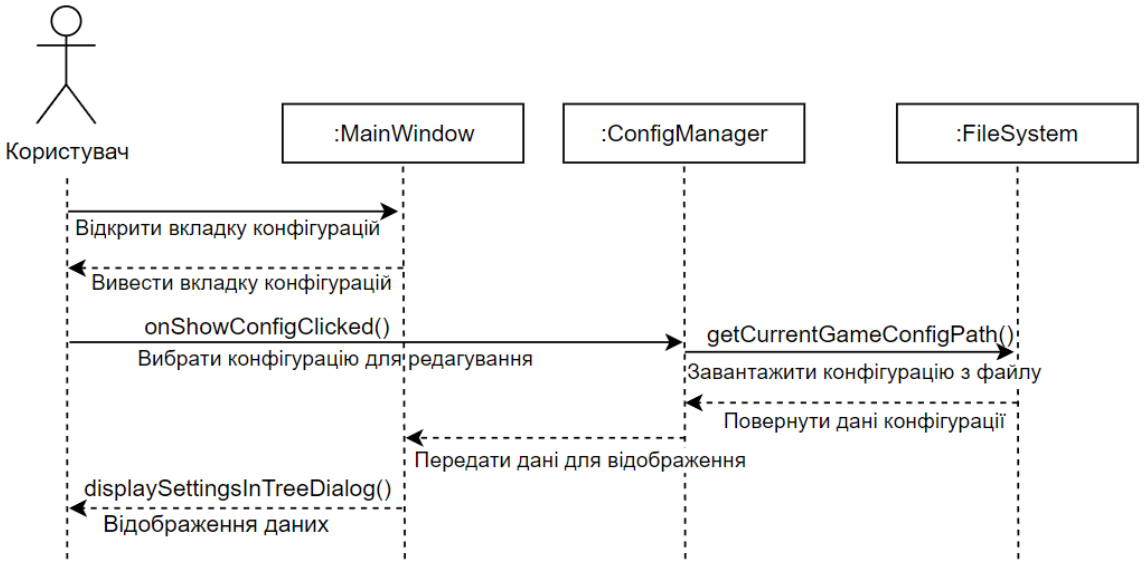


Рис. 3.4 Діаграма послідовності

Як ілюструє рисунок 3.4, все починається з дії Користувача, який через головне вікно застосунку (:MainWindow) вибирає опцію перегляду конфігурації. :MainWindow, як компонент інтерфейсу, не "знає", як безпосередньо працювати з файлом чи парсити XML. Його завдання – лише передати запит далі. Тому він звертається до :ConfigManager (частина шару ядра), який є експертом у всьому, що стосується конфігурацій. :ConfigManager отримує запит "readCurrentSettings()" і, розуміючи, що дані треба зчитати з файлу, делегує цю низькорівневу операцію компоненту :FileSystem (частина шару доступу до даних). :FileSystem виконує Завантажити конфігурацію з файлу, зчитуючи вміст preferences.xml. Отримані дані повертаються через :ConfigManager до :MainWindow, а звідти передаються компоненту :ConfigEditor, який відповідає за їх парсинг, інтерпретацію та візуалізацію у вигляді зрозумілих користувачеві елементів управління (полів вводу, перемикачів). Фінальний крок – :MainWindow відображає ці дані користувачеві.

Критично важливим функціоналом для безпечного використання застосунку є управління резервними копіями налаштувань. Користувачі повинні мати

можливість зберегти поточну "робочу" конфігурацію перед експериментами з новими налаштуваннями або модифікаціями, а також мати змогу швидко відновити стабільну версію у випадку виникнення проблем. Цей процес включає кілька можливих шляхів виконання: створення нової резервної копії або вибір та відновлення однієї з існуючих. Повний потік операцій та рішень в цьому процесі відображає діаграма діяльності "Управління резервними копіями", яка представлена на рисунку 3.5.

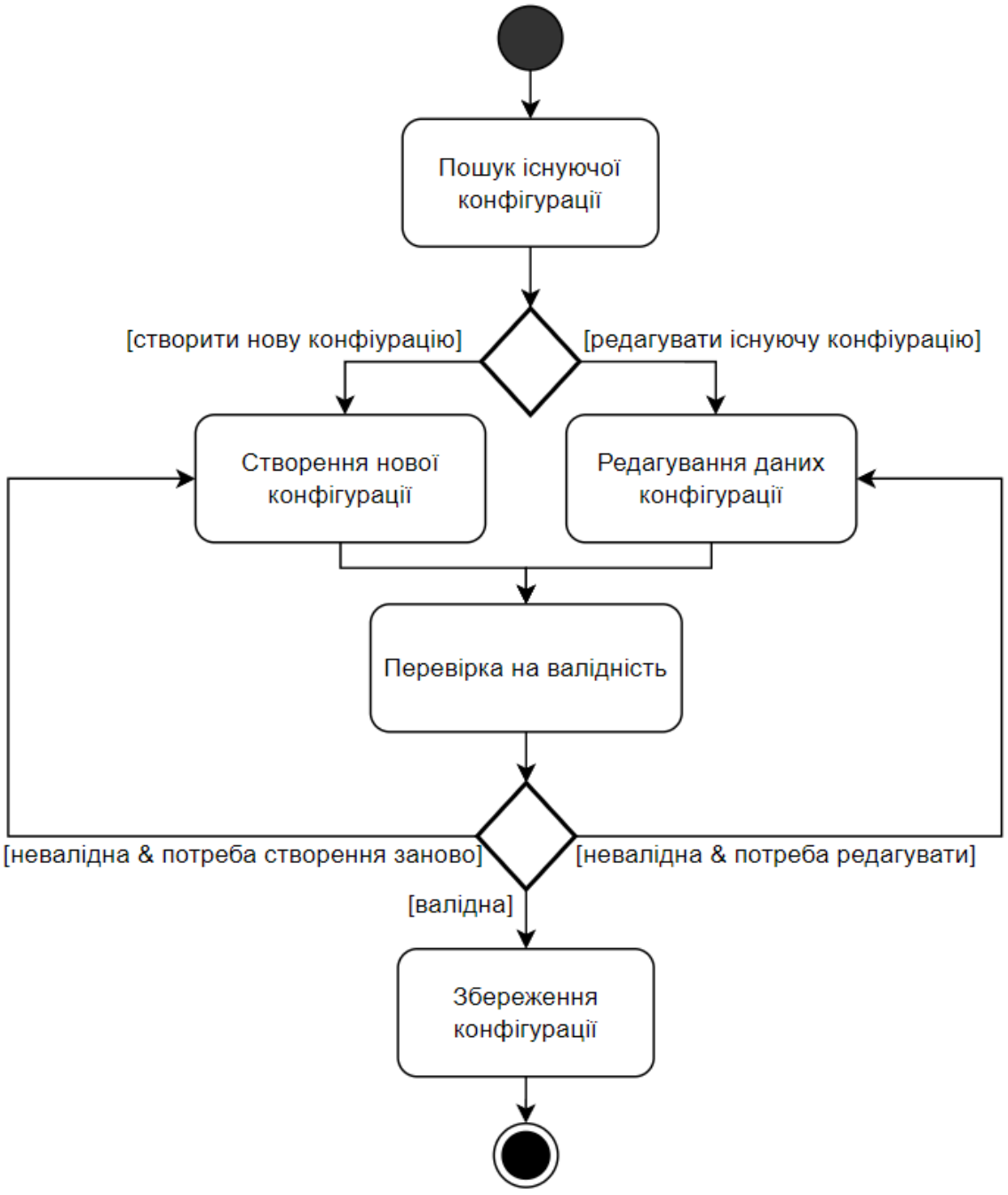


Рис. 3.5 Діаграма діяльності "Управління резервними копіями"

Діаграма діяльності на рисунку 3.5 візуалізує логіку, що стоїть за функціоналом резервного копіювання в застосунку. Процес може розпочатися з перевірки наявності існуючих резервних копій ("Пошук резервної копії"). Далі користувач опиняється у стані вибору: він може або ініціювати "Збереження резервної копії" (що є окремою дією, яка веде до завершення поточного процесу), або перейти до "Вибір резервної копії" з метою подальшого відновлення. Якщо вибір зроблено на користь відновлення, система перевіряє, чи існує резервна копія; якщо ні, користувача повертають до вибору. Якщо копія знайдена, виконується дія "Відновлення конфігурації", яка змінює поточні налаштування гри на ті, що були збережені. Така візуалізація допомагає зрозуміти можливі сценарії та переходи станів користувача в рамках цього функціоналу.

Об'єктна взаємодія під час сценарію створення резервної копії - це пряма дія користувача, спрямована на збереження поточного стану файлу preferences.xml у безпечному місці, щоб мати можливість повернутися до нього пізніше. Діаграма послідовності "Створення резервної копії", яка зображена на рисунку 3.6, деталізує обмін повідомленнями між об'єктами, що відбувається в цей момент.

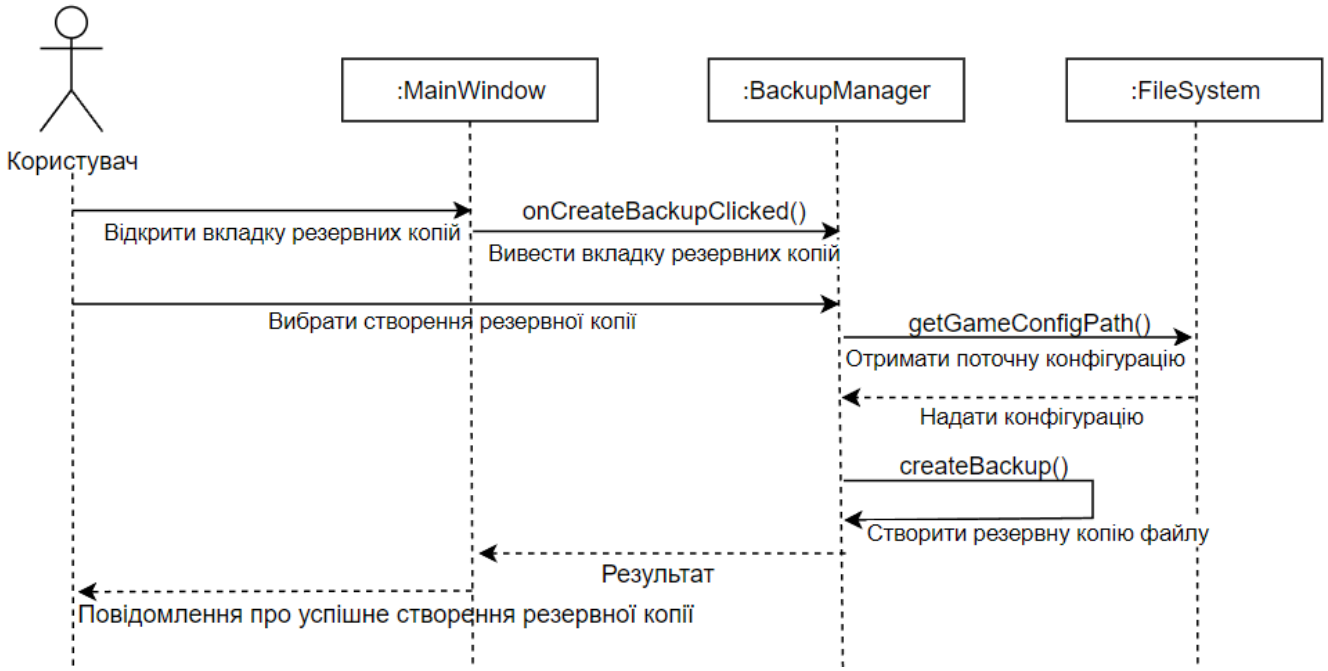


Рис. 3.6 Діаграма послідовності "Створення резервної копії"

На рисунку 3.6 показано, що Користувач натискає кнопку створення резервної копії у :MainWindow.

Головне вікно передає цей запит об'єкту :BackupManager, який є частиною логіки управління резервними копіями. :BackupManager делегує завдання надати поточний файл preferences.xml компоненту :FileSystem, викликаючи метод createBackup() та надаючи необхідні дані (шлях та поточну дату для копії). :FileSystem виконує операцію з пошуку файлу preferences.xml, і надає його компоненту :BackupManager, який в свою чергу створює резервну копію файлу preferences.xml. Після завершення, результат повертається через :BackupManager до :MainWindow.

Головне вікно отримує цей результат (успіх чи помилка) і відображає відповідне повідомлення користувачеві. Це демонструє, як запит на високому рівні (від користувача) трансформується у низькорівневу файлову операцію, керовану спеціалізованими компонентами.

Система також моделює різні стани, в яких може перебувати застосунок або процес роботи з конфігураційним файлом, залежно від дій користувача та внутрішніх операцій. Це важливо для коректного відображення поточного стану програми в інтерфейсі та для ефективного управління її логікою. Діаграма станів застосунку, яка зображена на рисунку 3.7, ілюструє загальні стани програми, такі як "Запущено", "Очікування дії користувача", "Виконання операції" та "Помилка", показуючи можливі переходи між ними при взаємодії користувача та системи. У свою чергу, діаграма станів роботи з конфігураційним файлом, що зображена на рисунку 3.8, більш детально відображає життєвий цикл файлу preferences.xml у контексті застосунку. Вона включає такі стани, як "Файл завантажено", "Файл модифіковано", "Файл збережено" або "Файл некоректний", а також переходи, які запускаються діями користувача (наприклад, редагування, збереження) або системними подіями (наприклад, валідація файлу). Ці діаграми є критично важливими для розуміння поведінки застосунку та забезпечення його стабільності, дозволяючи візуалізувати складні внутрішні процеси.

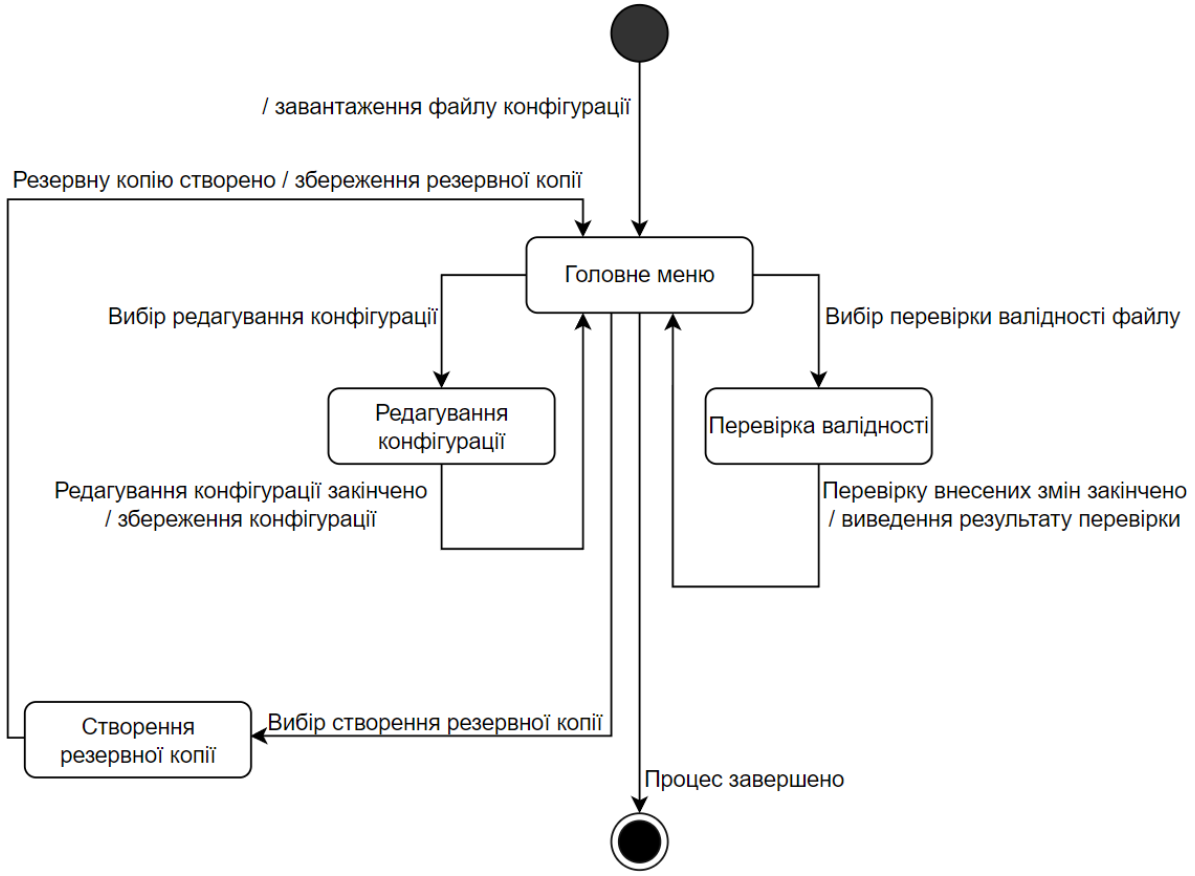


Рис. 3.7 Діаграма станів застосунку

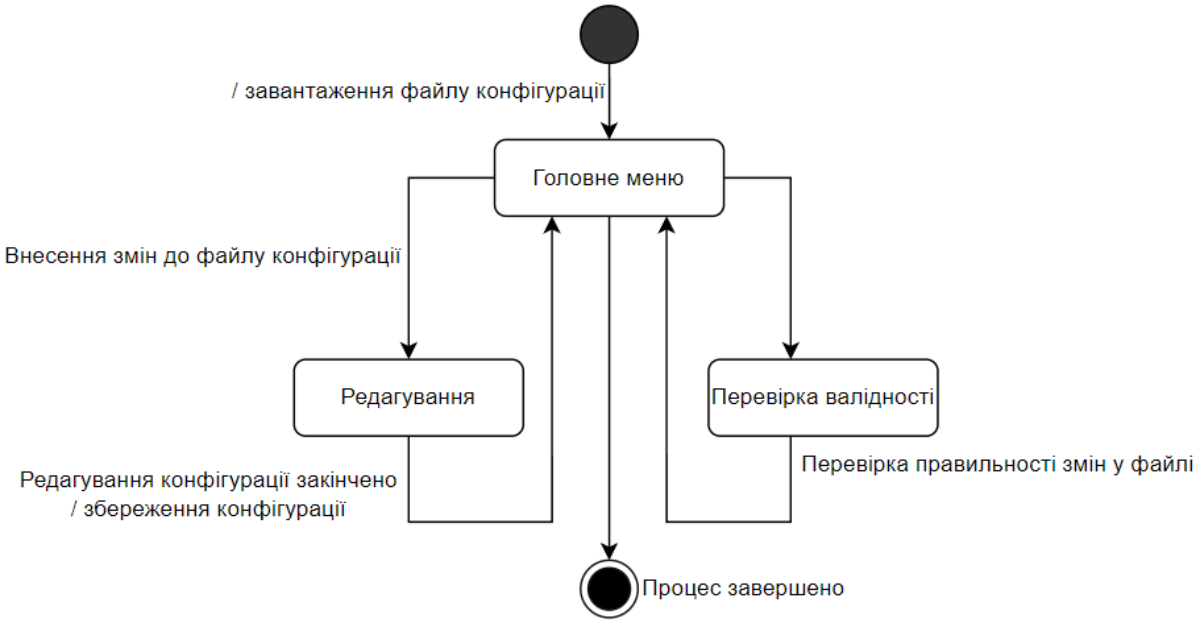


Рис. 3.8 Діаграма станів роботи з конфігураційним файлом

Наприклад, діаграма станів роботи з конфігураційним файлом, яка зображена на рисунку 3.8, показує, що процес може починатися з "Головне меню". При виборі опції редагування відбувається перехід до стану "Редагування". Після внесення змін та спроби збереження, система може перейти до стану "Перевірка валідності" файлу перед фактичним збереженням. Залежно від результату валідації ("Якщо валідна" або "Якщо невалідна"), процес може повернутися до стану "Редагування", або перейти до "Процес завершено" після успішного збереження.

Динамічна модель системи, комплексно представлена цими діаграмами взаємодії та станів, деталізує, як різні компоненти застосунку "WOT Configurator" співпрацюють, щоб надати користувачеві функціонал управління конфігураціями та резервними копіями. Вона показує послідовність дій, розподіл відповідальності під час виконання операцій та можливі зміни станів системи, що є ключовим для реалізації вимог до безпеки, зручності та надійності.

3.4 Діаграма класів

На додаток до бізнес-моделі, яка описує сутності предметної галузі на високому рівні, для розуміння внутрішнього влаштування застосунку була розроблена більш детальна статична модель у вигляді діаграми класів.

Ця діаграма показує програмні класи, які реалізують логіку застосунку, їхні атрибути, методи, а також взаємозв'язки між ними, включаючи використання інтерфейсів. Вона є "кресленням" кодової бази, що показує, як компоненти системи організовані для виконання своїх завдань. Детальна структура класів застосунку представлена на рисунку 3.9.

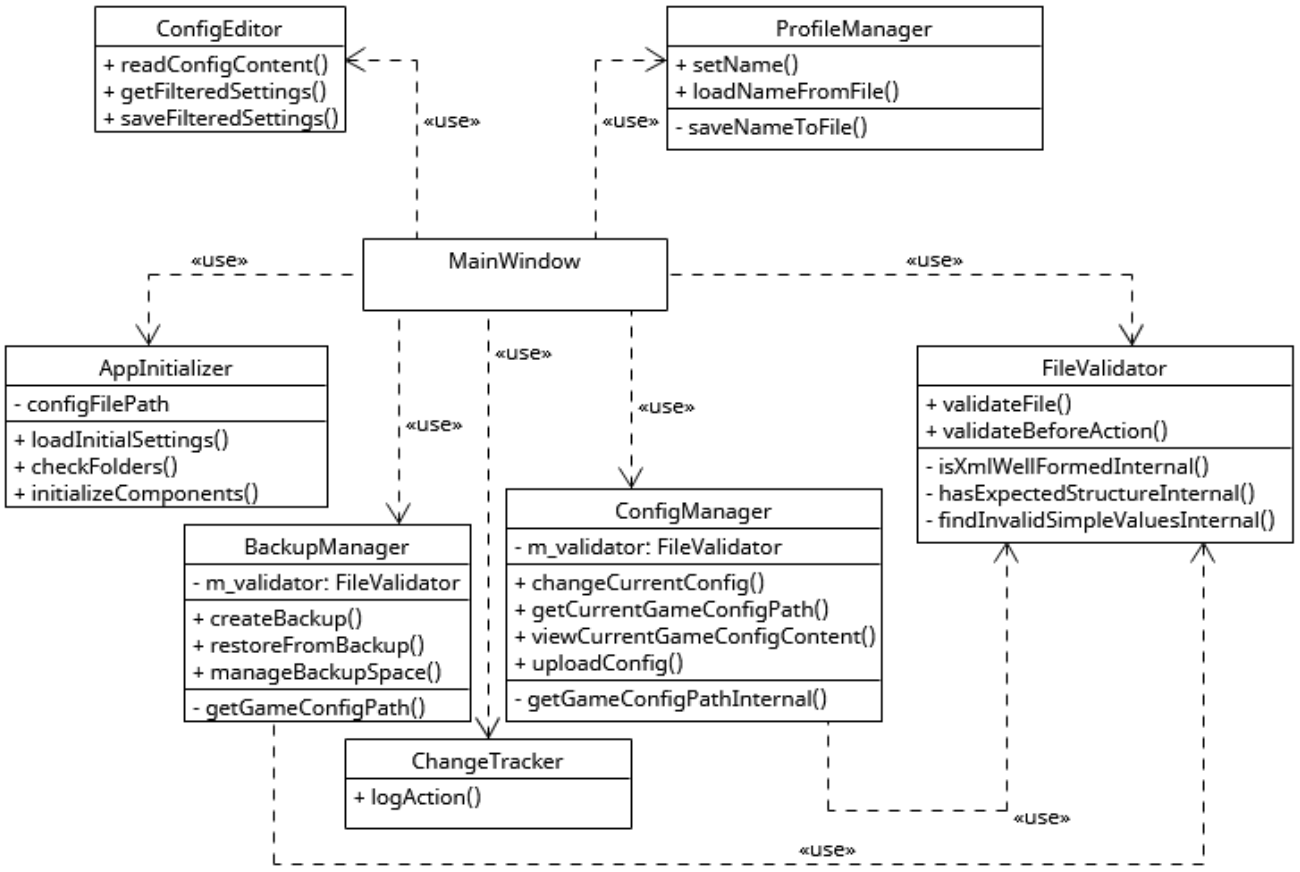


Рис. 3.9 Діаграма класів застосунку

Як ілюструє рисунок 3.9, основними будівельними блоками застосунку на програмному рівні є класи, що відповідають за конкретні функції:

– MainWindow: Центральний компонент інтерфейсу користувача, який взаємодіє з іншими менеджерами для відображення та керування даними.

– ConfigEditor: Відповідає за читання, отримання відфільтрованих налаштувань та збереження відфільтрованих налаштувань.

– ProfileManager: Керує профілями, дозволяючи встановлювати ім'я, завантажувати ім'я з файлу та зберігати ім'я у файл.

– AppInitializer: Відповідає за ініціалізацію застосунку, включаючи завантаження початкових налаштувань, перевірку папок та ініціалізацію компонентів. Зберігає шлях до файлу конфігурації.

– ConfigManager: Є ядром для роботи з файлом preferences.xml. Містить посилання на FileValidator. Відповідає за зміну поточної конфігурації, отримання

поточного шляху до ігрової конфігурації, перегляд вмісту поточної ігрової конфігурації, завантаження конфігурації та отримання внутрішнього шляху до ігрової конфігурації [20].

– FileValidator: Виконує критично важливу функцію перевірки коректності файлів. Має методи для валідації файлу, валідації перед дією, перевірки, чи є XML добре сформованим, та пошуку недійсних простих значень.

– BackupManager: Містить посилання на FileValidator. Реалізує функціонал резервного копіювання та відновлення. Має методи для створення резервної копії, відновлення з резервної копії, керування простором резервних копій та отримання шляху до ігрової конфігурації.

– ChangeTracker: Відповідає за відстеження історії змін, внесених до конфігурації, та реалізацію функціоналу скасування змін (Undo). Має метод для логування дій [21].

Ці детальні діаграми класів показують, як статична структура застосунку спроектована для підтримки ключових функцій управління конфігураціями. Кожен клас має чітко визначену відповідальність, а їх взаємозв'язки забезпечують потік даних та керування, необхідний для виконання операцій з файлом preferences.xml, його валідації, резервного копіювання та відстеження змін.

3.5 Фізична структура та розгортання системи

Окрім логічної та статичної структури, важливим аспектом проектування є визначення фізичної організації системи – з яких виконавчих компонентів вона складається та як ці компоненти розміщуються та взаємодіють у середовищі виконання. Діаграми компонентів та розгортання ілюструють ці аспекти.

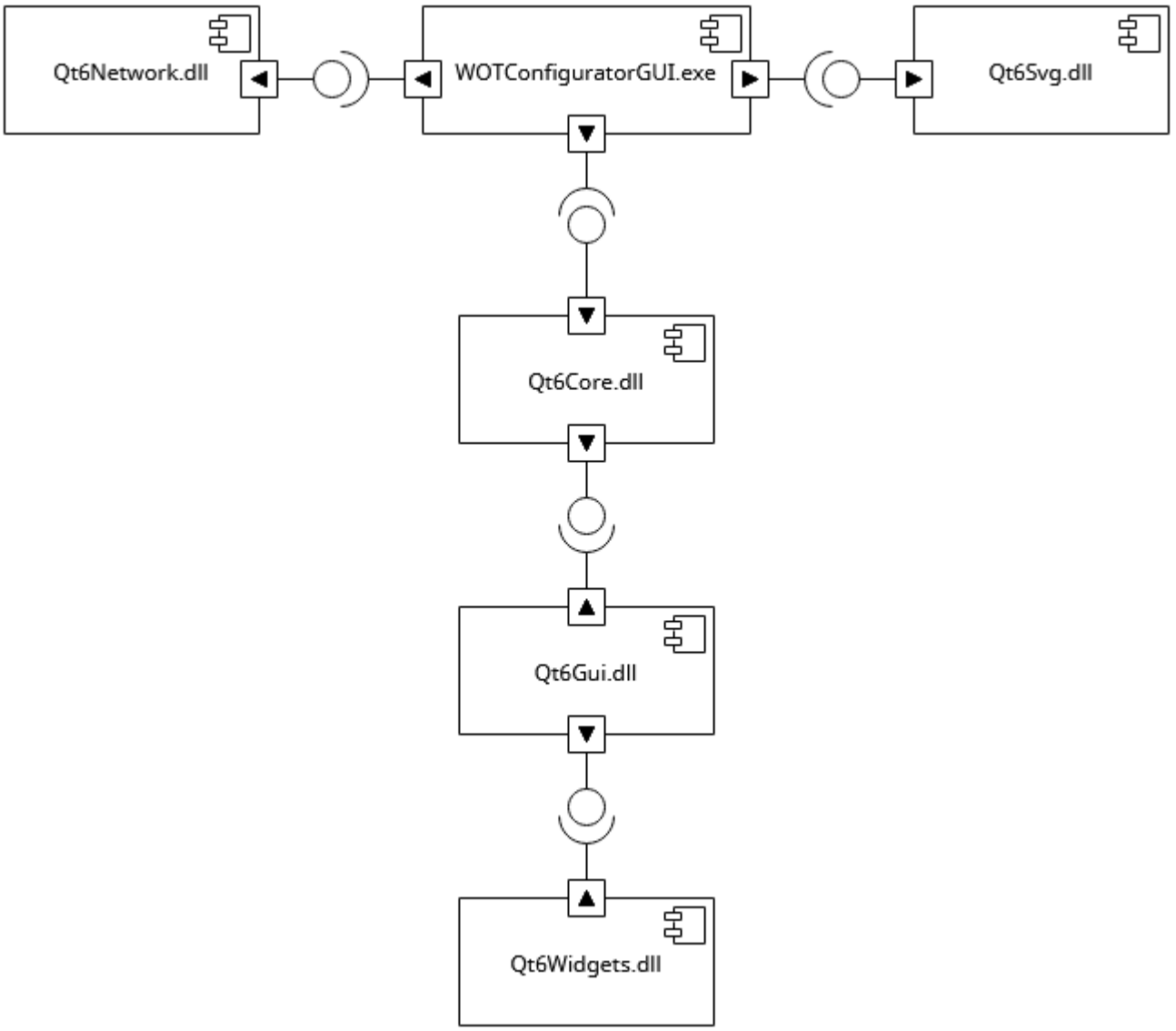


Рис. 3.10 Діаграма компонентів застосунку

Як видно з рисунка 3.10, застосунок складається з кількох ключових компонентів:

– Основний виконуваний файл (WOTConfiguratorGUI.exe): Це головний виконуваний файл застосунку, який містить зібраний код всіх модулів і є точкою входу для користувача.

– Qt6Core.dll: Базова бібліотека Qt, що надає основні неграфічні функціональності, такі як обробка подій, мета-об'єктна система та структури даних.

– Qt6Gui.dll: Бібліотека, що містить основні графічні функціональності, необхідні для роботи з вікнами, подіями введення та 2D-графікою.

– Qt6Widgets.dll: Бібліотека, що надає готові віджети (елементи керування користувацького інтерфейсу), такі як кнопки, текстові поля тощо, які використовуються для побудови GUI застосунку.

– Qt6Network.dll: Бібліотека Qt, що забезпечує функціонал для роботи з мережею, такий як HTTP-запити, TCP/IP сокети тощо, що може бути використано для взаємодії із зовнішніми API.

– Qt6Svg.dll: Бібліотека для роботи з масштабованою векторною графікою (SVG), яка може бути використана для відображення іконок або інших графічних елементів інтерфейсу.

Ця діаграма компонентів показує, як виконуваний файл WOTConfiguratorGUI.exe залежить від ключових бібліотек фреймворку Qt для забезпечення своєї функціональності та графічного інтерфейсу.

Діаграма розгортання ілюструє, як компоненти системи розміщуються та взаємодіють у фізичному середовищі.

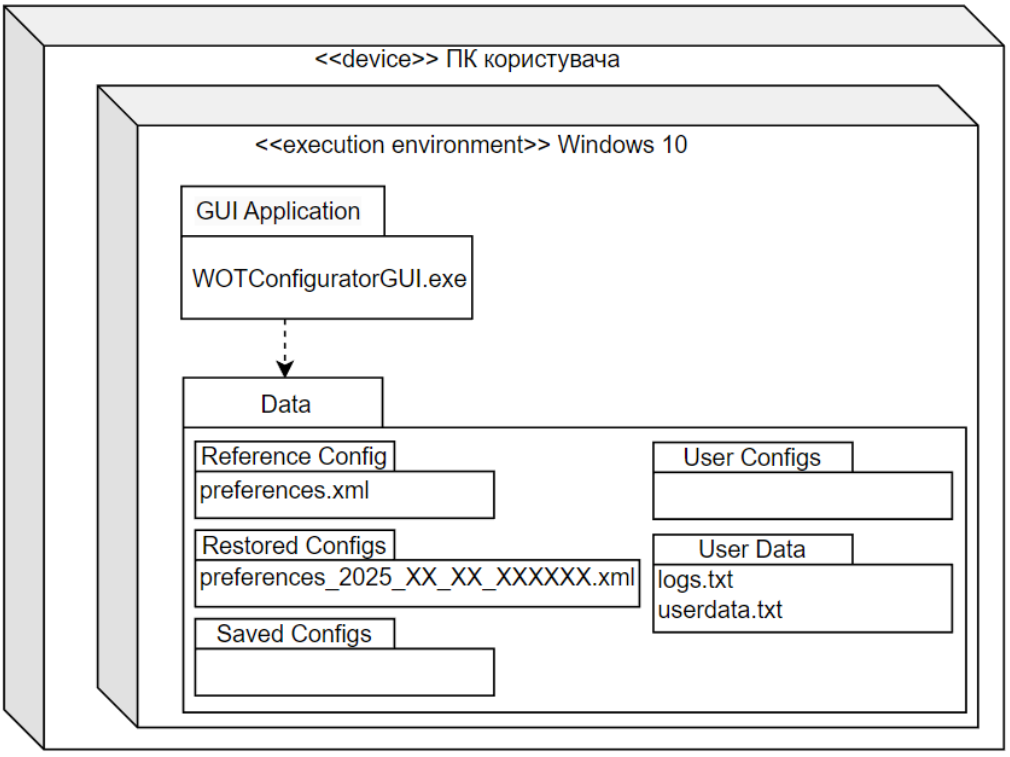


Рис. 3.11 Діаграма розгортання застосунку

Згідно з рисунком 3.11, основним вузлом розгортання є <<device>> ПК користувача з операційним середовищем <<execution environment>> Windows 10.

На цьому вузлі розміщується виконуваний файл застосунку GUI Application (WOTConfiguratorGUI.exe). Цей застосунок безпосередньо взаємодіє з каталогом Data, який містить різні типи конфігураційних файлів та користувацьких даних:

– Reference Config: Сюди входить preferences.xml, який є основним конфігураційним файлом гри.

– Restored Configs: Каталог для збереження відновлених конфігурацій, назви яких можуть включати дату (preferences_2025_XX_XX_XXXXXX.xml).

– Saved Configs: Каталог для збереження інших збережених конфігурацій.

– User Configs: Каталог для користувацьких конфігурацій.

– User Data: Містить файли logs.txt для журналювання подій та userdata.txt для зберігання користувацьких даних.

Такий розподіл компонентів на вузлі розгортання чітко показує, що основна логіка управління конфігураціями та зберігання даних виконується локально на комп'ютері користувача.

Це забезпечує самодостатність та незалежність застосунку від постійного доступу до зовнішніх серверів для базових операцій з файлами.

Незважаючи на інтеграцію із зовнішніми API (як показано на інших діаграмах), основне функціональне ядро "WOT Configurator" функціонує автономно, гарантуючи стабільну роботу навіть за відсутності інтернет-з'єднання для операцій з локальними конфігураціями.

Це підвищує надійність та доступність застосунку для кінцевого користувача.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Архітектура та реалізація ключових компонентів

Реалізація програмного забезпечення є етапом, на якому абстрактні проектні рішення та моделі (описані в розділі 3) перетворюються на функціональний код. В основі цього процесу лежить архітектурна модель, яка визначає, як різні компоненти системи організовані та взаємодіють.

Проект WOT Configurator розроблений на базі об'єктно-орієнтованого підходу з використанням мови програмування C++ та фреймворку Qt, що забезпечує модульність, розширюваність та ефективну розробку графічного інтерфейсу.

Структура файлів та директорій проекту, яка відображає його фізичну організацію в середовищі розробки, представлена на рисунку 4.1.

Як ілюструє рисунок 4.1, фізична організація проекту в середовищі розробки базується на групуванні файлів за їх типом та призначенням.

Ключові групи включають Source Files (файли .cpp з реалізацією), Header Files (файли .h з оголошеннями класів та функцій), Forms (файли .ui з описом графічного інтерфейсу, створені в Qt Designer), Resources (ресурси проекту) та external (зовнішні бібліотеки). Така організація є типовою для проектів на C++ з використанням Qt.

Хоча на фізичному рівні файли згруповані за типом, на логічному рівні реалізація застосунку відповідає багатошаровій архітектурі, описаній у розділі 3. Класи та функції, що реалізують логіку різних шарів та компонентів, розташовані серед файлів у цих фізичних директоріях.

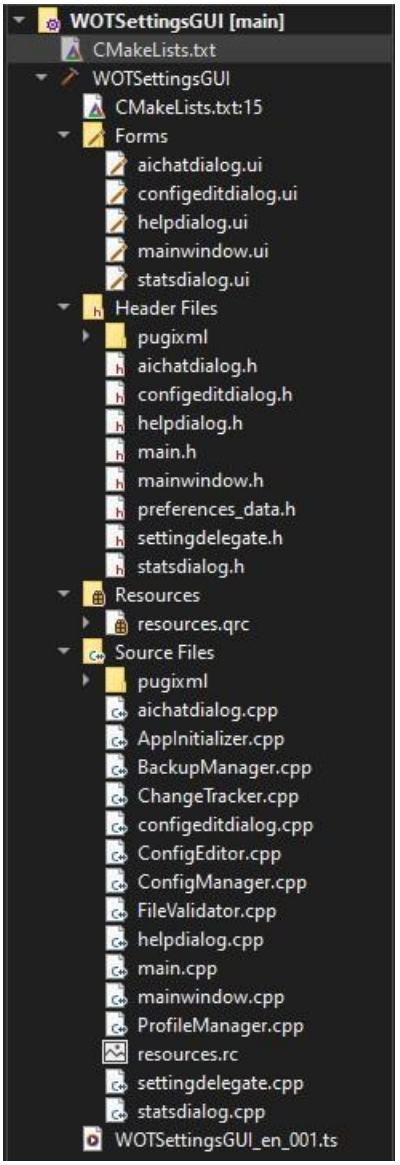


Рис. 4.1 Структура файлів та директорій проекту WOT Configurator

Розглянемо реалізацію ключових компонентів:

Управління конфігурацією та файлова система:

Ядром взаємодії з ігровим конфігураційним файлом preferences.xml є клас ConfigManager.

Він відповідає за визначення шляху до файлу, його читання та запис. Визначення стандартного шляху до файлу в директорії %APPDATA% реалізовано методом getGameConfigPathInternal. Метод getGameConfigPathInternal відображено на рисунку 4.2.

```cpp
fs::path ConfigManager::getGameConfigPathInternal() {
    const char* appDataPath_cstr = getenv("APPDATA");
    if (!appDataPath_cstr) {
        throw std::runtime_error("Не вдалося отримати системний шлях APPDATA.");
    }
    fs::path appDataDir(appDataPath_cstr);
    if (!fs::exists(appDataDir) || !fs::is_directory(appDataDir)) {
        throw std::runtime_error("Шлях APPDATA не існує або не є директорією: " + std::string(appDataPath_cstr));
    }
    return appDataDir / "Wargaming.net" / "WorldOfTanks" / "preferences.xml";
}

// Повертає шлях до поточного конфігу гри
fs::path ConfigManager::getCurrentGameConfigPath() {
    return getGameConfigPathInternal();
}
```

Рис. 4.2 Код методів getGameConfigPathInternal та getCurrentGameConfigPath

Як бачимо з коду, для роботи зі шляхами файлів використовується стандартна бібліотека std::filesystem. Метод getCurrentGameConfigPath надає зовнішнім компонентам актуальний шлях до конфігураційного файлу гри.

Застосування (копіювання) вибраної користувачем конфігурації поверх ігрового файлу preferences.xml реалізовано у методі changeCurrentConfig. Цей метод приймає шлях до файлу-джерела та використовує функції файлової системи для копіювання:

```cpp
void ConfigManager::changeCurrentConfig(const fs::path& sourceConfigPath) {
    if (!fs::exists(sourceConfigPath)) {
        throw std::runtime_error("Обраний файл конфігурації користувача не знайдено: " + sourceConfigPath.string());
    }
    if (!fs::is_regular_file(sourceConfigPath)) {
        throw std::runtime_error("Вибраний шлях не є файлом: " + sourceConfigPath.string());
    }

    fs::path targetPath = getGameConfigPathInternal();
    fs::path targetDir = targetPath.parent_path();

    // Застосування (копіювання з заміною)
    try {
        if (!fs::exists(targetDir)) {
            fs::create_directories(targetDir);
        }
        fs::copy_file(sourceConfigPath, targetPath, fs::copy_options::overwrite_existing);
        // Якщо копіювання пройшло без винятків - успіх
    } catch (const fs::filesystem_error& e) {
        throw std::runtime_error(std::string("Помилка файлової системи при застосуванні конфігу: ") + e.what());
    } catch (...) {
        throw std::runtime_error("Невідома помилка під час застосування конфігу.");
    }
}
```

Рис. 4.3 Код методу changeCurrentConfig

Цей фрагмент демонструє пряму взаємодію з файловою системою для виконання ключової операції із заміни файлу конфігурації, включаючи створення необхідних директорій та обробку можливих помилок.

Читання вмісту поточного конфігураційного файлу гри реалізовано у методі viewCurrentGameConfigContent. Цей метод перед читанням виконує валідацію файлу за допомогою компонента FileValidator та обробляє його вміст як рядок:

```
std::string ConfigManager::viewCurrentGameConfigContent() {
    fs::path gameConfigPath = getGameConfigPathInternal();
    if (!fs::exists(gameConfigPath)) {
        throw std::runtime_error("Файл конфігурації гри (preferences.xml) не знайдено.");
    }

    // Валідація перед читанням (використовуємо внутрішній валідатор)
    ValidationResult valResult = m_validator.validateFile(gameConfigPath);
    if (!valResult.isValid()) {
        throw std::runtime_error("Поточний конфіг гри не є коректним XML: " + valResult.wellFormedError);
    }

    std::ifstream file(gameConfigPath);
    if (!file.is_open()) {
        throw std::runtime_error("Не вдалося відкрити файл конфігурації гри: " + gameConfigPath.string());
    }
    std::stringstream buffer;
    try {
        buffer << file.rdbuf();
    } catch (const std::exception& e) {
        throw std::runtime_error(std::string("Помилка читання файлу конфігурації гри: ") + e.what());
    }
    return buffer.str();
}
```

Рис. 4.4 Код методу viewCurrentGameConfigContent

Тут показано, як ConfigManager делегує частину роботи (валідацію) іншому компоненту (m_validator), що відповідає принципам розподілу відповідальності.

Валідація файлу preferences.xml є критично важливою функцією для запобігання пошкодженню налаштувань гри. Цю логіку реалізовано у класі FileValidator, зокрема у методі validateFile. Валідація включає перевірку на три рівні: коректність XML-формату (well-formed), відповідність очікуваній структурі та коректність значень окремих елементів. Для парсингу XML використовується стороння бібліотека Pugixml.

Фрагмент методу isXmlWellFormedInternal демонструє завантаження файлу та базову перевірку XML-формату за допомогою Pugixml:

```
bool FileValidator::isXmlWellFormedInternal(const fs::path& filePath, pugi::xml_document& doc, std::string& errorMsg) {
    pugi::xml_parse_result result = doc.load_file(filePath.c_str());
    if (result.status != pugi::status_ok) {
        // Формуємо повідомлення про помилку
        std::stringstream ss;
        ss << "Помилка XML: " << result.description() << " (позиція " << result.offset << ")";
        errorMsg = ss.str();
        return false;
    }
    errorMsg = "OK";
    return true;
}
```

Рис. 4.5 Код методу isXmlWellFormedInternal

Цей код прямо показує використання ключової функції doc.load_file() бібліотеки Pugixml для спроби завантажити XML-документ та отримати результат парсингу.

Перевірка структури (hasExpectedStructureInternal) та значень (findInvalidSimpleValuesInternal) також використовує Pugixml для навігації по XML-дереву (наприклад, doc.child(), node.text().as_string()) та стандартні функції C++ для перевірки типів та діапазонів значень, як видно з наданого коду методу findInvalidSimpleValuesInternal.

```
bool FileValidator::validateBeforeAction(const fs::path& filePath, const std::string& actionNameStd, bool showSuccess) {
    QString actionName = QString::fromStdString(actionNameStd);
    ValidationResult result = validateFile(filePath);

    if (result.isValid()) {
        QString summary;
        bool hasWarnings = false;

        summary += "XML: OK.\n";
        summary += QString("Структура: %1\n").arg(QString::fromStdString(result.structureInfo));
        if (!result.hasStructure) hasWarnings = true;

        if (result.valueErrors.empty()) {
            summary += "Значення: OK.";
        } else {
            summary += QString("Значення: Знайдено %1 потенційних проблем.").arg(result.valueErrors.size());
            hasWarnings = true;
            summary += "\nПриклади помилок значень:\n";
            int count = 0;
            for(const auto& err : result.valueErrors) {
                if (++count > 3) {
                    summary += "- ...\n";
                    break;
                }
                summary += QString("- %1\n").arg(QString::fromStdString(err));
            }
        }

        if (hasWarnings) {
            QMessageBox::StandardButton reply;
            QString warningText = QString("Увага! Файл '%1' містить попередження:\n\n%2\n\nПродовжити дію '%3'?")
                                      .arg(QString::fromStdWString(filePath.filename().wstring()))
                                      .arg(summary)
                                      .arg(actionName);
            reply = QMessageBox::warning(nullptr, actionName + " - Попередження валідації",
                                         warningText,
                                         QMessageBox::Yes | QMessageBox::No,
                                         QMessageBox::No);
            return (reply == QMessageBox::Yes);
        } else {
            if (showSuccess) {
                QMessageBox::information(nullptr, actionName + " - Валідація успішна",
                                         QString("Файл '%1' успішно пройшов валідацію.")
                                             .arg(QString::fromStdWString(filePath.filename().wstring())));
            }
            return true;
        }
    } else { // Критична помилка XML
        QMessageBox::critical(nullptr, actionName + " - Помилка валідації",
                              QString("Не вдалося виконати дію '%1'.\nФайл '%2' не пройшов валідацію:\n%3")
                                  .arg(actionName)
                                  .arg(QString::fromStdWString(filePath.filename().wstring()))
                                  .arg(QString::fromStdString(result.wellFormedError)));
        return false;
    }
}
```

Рис. 4.6 Код методу validateBeforeAction

Цей фрагмент ілюструє зв'язок шару доступу до даних/валідації з шаром інтерфейсу користувача через використання компонентів Qt для відображення повідомлень.

Управління резервними копіями:

Клас BackupManager відповідає за створення та відновлення резервних копій. Створення резервної копії в методі createBackup включає валідацію файлу-джерела, генерацію унікального імені файлу на основі дати та часу, створення директорії для бекапів (якщо необхідно) та копіювання файлу за допомогою std::filesystem:

```
fs::path BackupManager::createBackup() {
    fs::path sourcePath = getGameConfigPath();

    if (!fs::exists(sourcePath)) {
        throw std::runtime_error("Файл конфігурації гри для резервного копіювання не знайдено: " + sourcePath.string());
    }

    if (!m_validator.validateBeforeAction(sourcePath, "Створення резервної копії (перевірка джерела)", false)) {
        throw std::runtime_error("Перевірка файлу-джерела перед резервним копіюванням не пройдена або скасована.");
    }

    auto now = std::chrono::system_clock::now();
    auto in_time_t = std::chrono::system_clock::to_time_t(now);
    std::tm local_tm;
#ifdef _WIN32
    localtime_s(&local_tm, &in_time_t);
#else
    localtime_r(&in_time_t, &local_tm); // POSIX
#endif
    std::stringstream ss;
    ss << "preferences_" << std::put_time(&local_tm, "%Y_%m_%d_%H%M%S") << ".xml";

    fs::path backupDir = "Restored Configs";
    fs::path backupPath = backupDir / ss.str();

    try {
        if (!fs::exists(backupDir)) {
            fs::create_directories(backupDir);
        }
        fs::copy_file(sourcePath, backupPath, fs::copy_options::overwrite_existing);
        return backupPath;
    } catch (const fs::filesystem_error& e) {
        throw std::runtime_error(std::string("Помилка файлової системи при створенні резервної копії: ") + e.what());
    } catch (...) {
        throw std::runtime_error("Невідома помилка під час створення резервної копії.");
    }
}
```

Рис. 4.7 Код методу createBackup

Відновлення з резервної копії у методі restoreFromBackup також використовує файлові операції для заміни поточного файлу конфігурації на вибрану резервну копію:

```cpp
void BackupManager::restoreFromBackup(const fs::path& backupPath) {
    if (!fs::exists(backupPath)) {
        throw std::runtime_error("Обраний файл резервної копії не знайдено: " + backupPath.string());
    }

    fs::path targetPath = getGameConfigPath();
    fs::path targetDir = targetPath.parent_path();

    try {
        if (!fs::exists(targetDir)) {
            fs::create_directories(targetDir);
        }
        fs::copy_file(backupPath, targetPath, fs::copy_options::overwrite_existing);
    } catch (const fs::filesystem_error& e) {
        throw std::runtime_error(std::string("Помилка файлової системи при відновленні з копії: ") + e.what());
    } catch (...) {
        throw std::runtime_error("Невідома помилка під час відновлення з копії.");
    }
}
```

Рис. 4.8 Код методу restoreFromBackup

Відстеження змін та логування:

Клас ChangeTracker реалізує функціональність логування дій до текстового файлу. Метод logAction форматує повідомлення з часовою міткою, назвою функції, результатом та деталями, і записує його у файл logs.txt у директорії "User Data", створюючи її за необхідності:

```cpp
void ChangeTracker::logAction(const std::string& functionName, bool success, const std::string& details) {
    const fs::path logDir = "User Data";
    const fs::path logFilePath = logDir / "logs.txt";

    try {
        if (!fs::exists(logDir)) {
            try {
                fs::create_directory(logDir);
            } catch(const fs::filesystem_error& e) {
                std::cerr << "CRITICAL ERROR: Could not create log directory '" << logDir.string() << "'. Error: " << e.what() << std::endl;
                return;
            }
        }

        std::ofstream logFile(logFilePath, std::ios::app);

        if (!logFile.is_open()) {
            std::cerr << "ERROR: Could not open log file for writing: " << logFilePath.string() << std::endl;
            return;
        }

        auto now = std::chrono::system_clock::now();
        auto now_c = std::chrono::system_clock::to_time_t(now);
        std::tm now_tm;
#ifdef _WIN32
        localtime_s(&now_tm, &now_c);
#else
        localtime_r(&now_c, &now_tm);
#endif

        logFile << std::put_time(&now_tm, "%Y-%m-%d_%H:%M:%S") << " | "
                << functionName << " | Result: [" << (success ? "OK" : "NOK") << "]";

        if (!details.empty()) {
            logFile << " | Details: " << details;
        }

        logFile << std::endl;

    } catch (const fs::filesystem_error& e) {
        std::cerr << "Filesystem error during logging: " << e.what() << std::endl;
    } catch (const std::exception& e) {
        std::cerr << "Standard exception during logging: " << e.what() << std::endl;
    } catch (...) {
        std::cerr << "Unknown error during logging." << std::endl;
    }
}
```

Рис. 4.9 Код методу logAction

Наданий фрагмент коду демонструє реалізацію механізму запису подій.

Таким чином, аналіз наданих фрагментів коду підтверджує, що ключові функціональні модулі застосунку "WOT Configurator" (управління конфігурацією, валідація, резервне копіювання, логування) реалізовані з використанням C++, стандартної бібліотеки std::filesystem для роботи з файлами, бібліотеки Pugixml для парсингу XML та фреймворку Qt для інтеграції з інтерфейсом користувача.

4.2 Проектування та реалізація користувацького інтерфейсу

Користувацький інтерфейс (UI) є обличчям будь-якого застосунку – це саме те, з чим безпосередньо взаємодіє користувач. Від його зручності, інтуїтивності та інформативності значною мірою залежить загальне сприйняття програми.

При проектуванні та реалізації інтерфейсу застосунку "WOT Configurator" ключовим принципом було створення простого, зрозумілого та функціонального інструменту, який дозволить гравцям World of Tanks легко керувати своїми налаштуваннями без необхідності вручну редагувати складний XML-файл.

Для реалізації графічного інтерфейсу був використаний кросплатформовий фреймворк Qt, зокрема його модулі для створення віджетів (елементів інтерфейсу) та дизайну вікон. Візуальне макетування основних екранних форм здійснювалося за допомогою Qt Designer – інструменту, який дозволяє швидко розміщувати елементи, налагоджувати їх властивості та зв'язки, генеруючи .ui файли. Потім ці .ui файли інтегрувалися з C++ кодом, де реалізовувалася логіка взаємодії елементів інтерфейсу з ядром застосунку.

Основним елементом інтерфейсу є головне вікно застосунку, зовнішній вигляд якого представлений на рисунку 4.10.

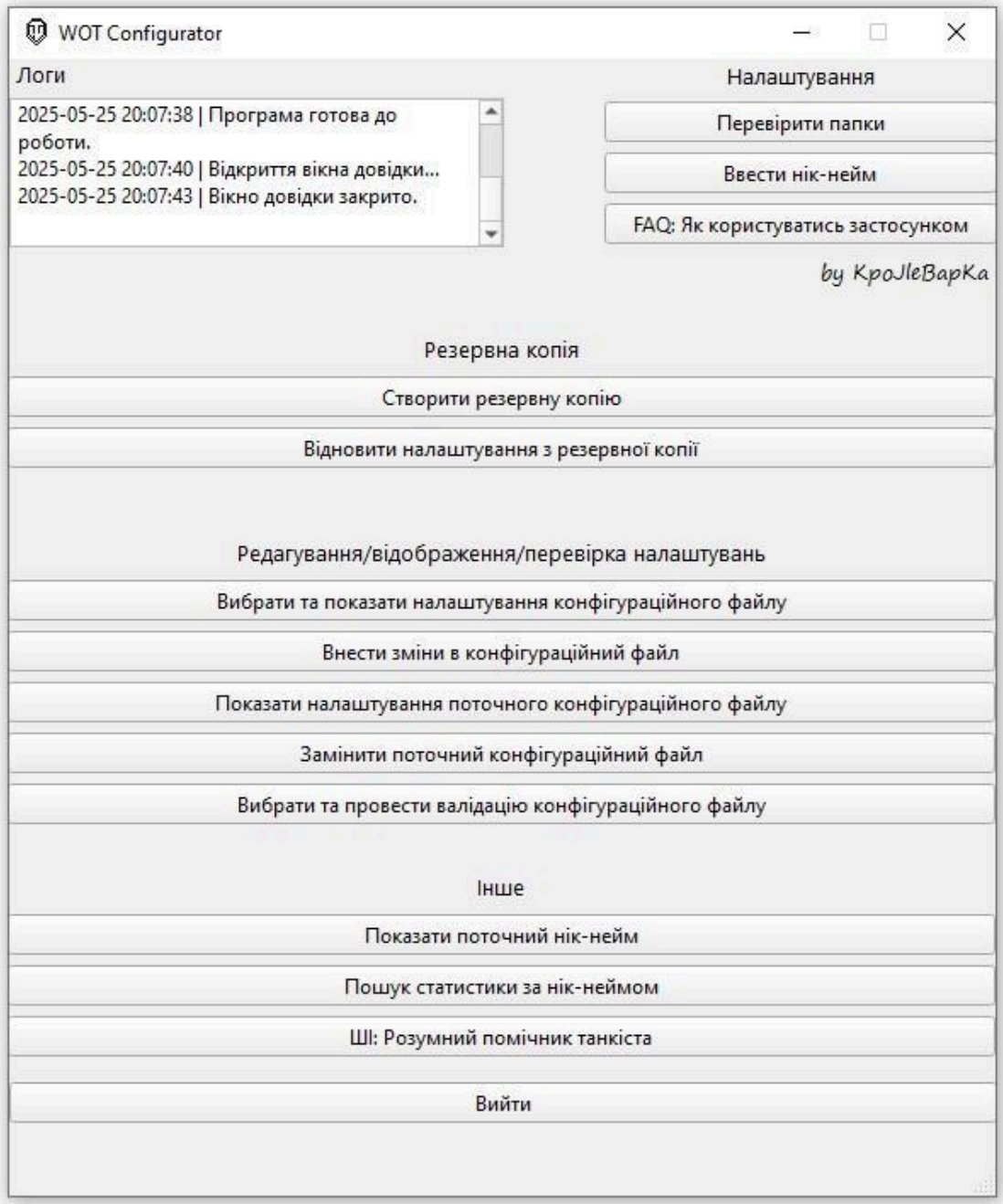


Рис. 4.10 Головне вікно застосунку WOT Configurator

Головне вікно (MainWindow) слугує центральним хабом для доступу до всього функціоналу програми. Воно надає користувачеві можливість:

– Переглядати лог-повідомлення: У верхній частині вікна розташована область для відображення системних повідомлень та логів (Logs), що дозволяє відстежувати дії програми та можливі помилки.

– Виконувати базові налаштування та перевірки: Група кнопок у правій верхній частині вікна ("Налаштування") дозволяє виконувати такі дії, як "Перевірити папки" (створення необхідних директорій) та "Ввести нікнейм" (виклик вікна для введення ігрового нікнейму).

– Працювати з резервними копіями: Група кнопок "Резервна копія" надає доступ до функціоналу збереження поточної конфігурації ("Створити резервну копію") та відновлення з раніше створених копій ("Відновити налаштування з резервної копії").

– Редагувати та переглядати налаштування: Найбільша група кнопок присвячена роботі безпосередньо з файлом preferences.xml. Тут користувач може "Вибрати та показати налаштування конфігураційного файлу" (завантажити файл для перегляду), "Показати налаштування поточної конфігураційного файлу" (зчитати актуальний файл гри), "Замінити поточний конфігураційний файл" (застосувати вибрану конфігурацію), "Вибрати та провести валідацію конфігураційного файлу" (перевірити коректність вибраного файлу) та інші опції для взаємодії з конфігураціями.

– Отримувати довідкову інформацію та використовувати додаткові функції: Група "Інше" включає кнопки для "Показати поточний нікнейм", "Пошук статистики за нікнеймом" (виклик вікна статистики) та "ШІ: Розумний помічник танкіста" (виклик вікна AI-асистента). Кнопка "FAQ: Як користуватись застосунком" у правій частині вікна відкриває вікно довідки.

– Вийти з програми: Кнопка "Вийти" завершує роботу застосунку.

Кожна кнопка на головному вікні пов'язана з відповідним методом у C++ коді (класу MainWindow), який ініціює виконання необхідної логіки в шарах Core або DataAccess, або відкриває нове діалогове вікно.

Розглянемо деякі з цих діалогових вікон, які є ключовими частинами інтерфейсу для виконання певних функцій:

Вікно введення нікнейму:

При першому запуску або при натисканні відповідної кнопки на головному вікні, застосунок пропонує користувачеві ввести свій нікнейм World of Tanks. Це реалізовано у вигляді модального діалогового вікна, як показано на рисунку 4.11.

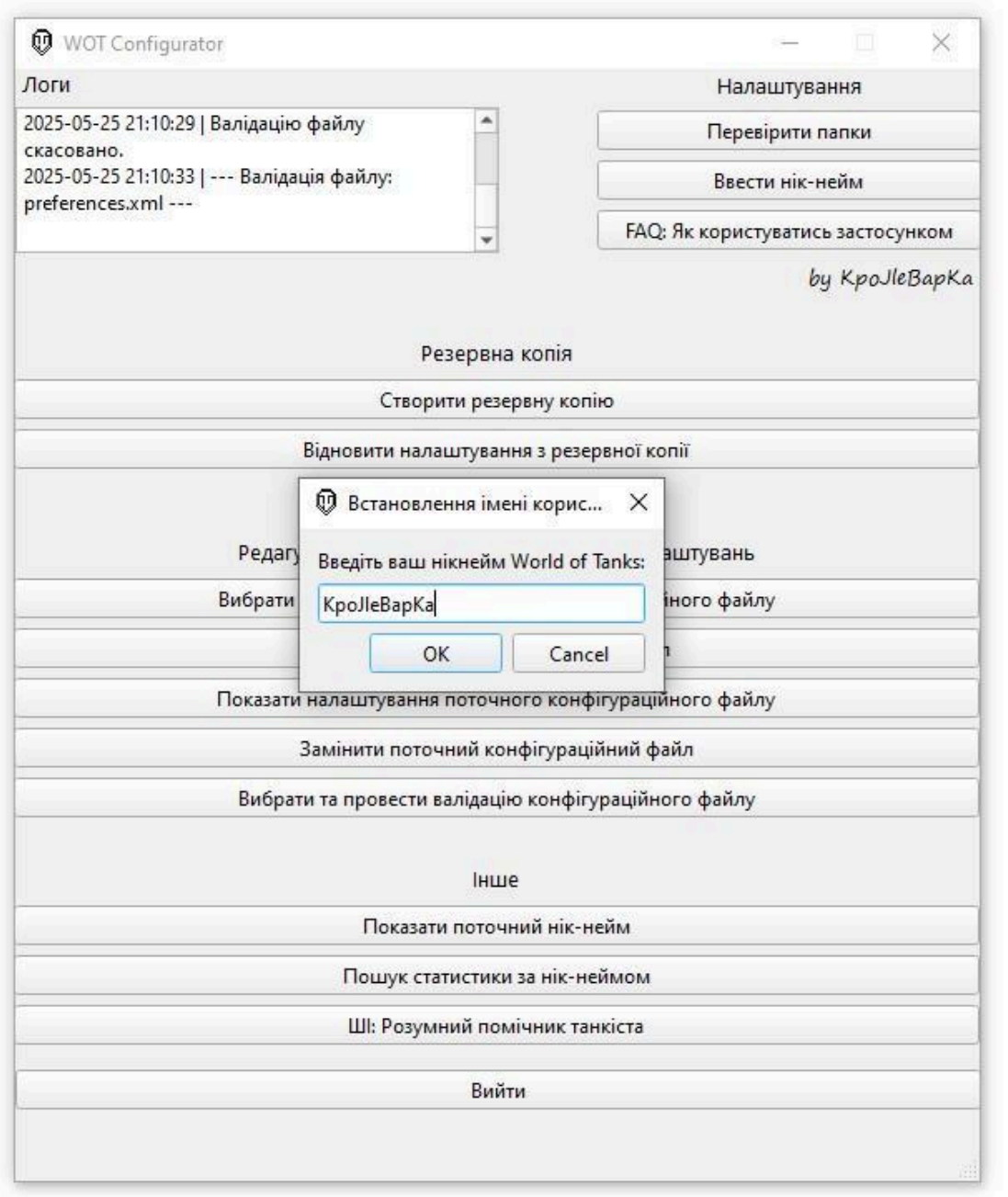


Рис. 4.11 Вікно введення нікнейму

Це просте вікно з полем вводу та кнопками "OK" і "Cancel". Введений користувачем нікнейм далі використовується для ідентифікації.

Вікно Довідки та Частих Питань (FAQ):

Для надання користувачеві допомоги та відповідей на поширені запитання було реалізовано окреме вікно Довідки, яке показано на рисунку 4.12.

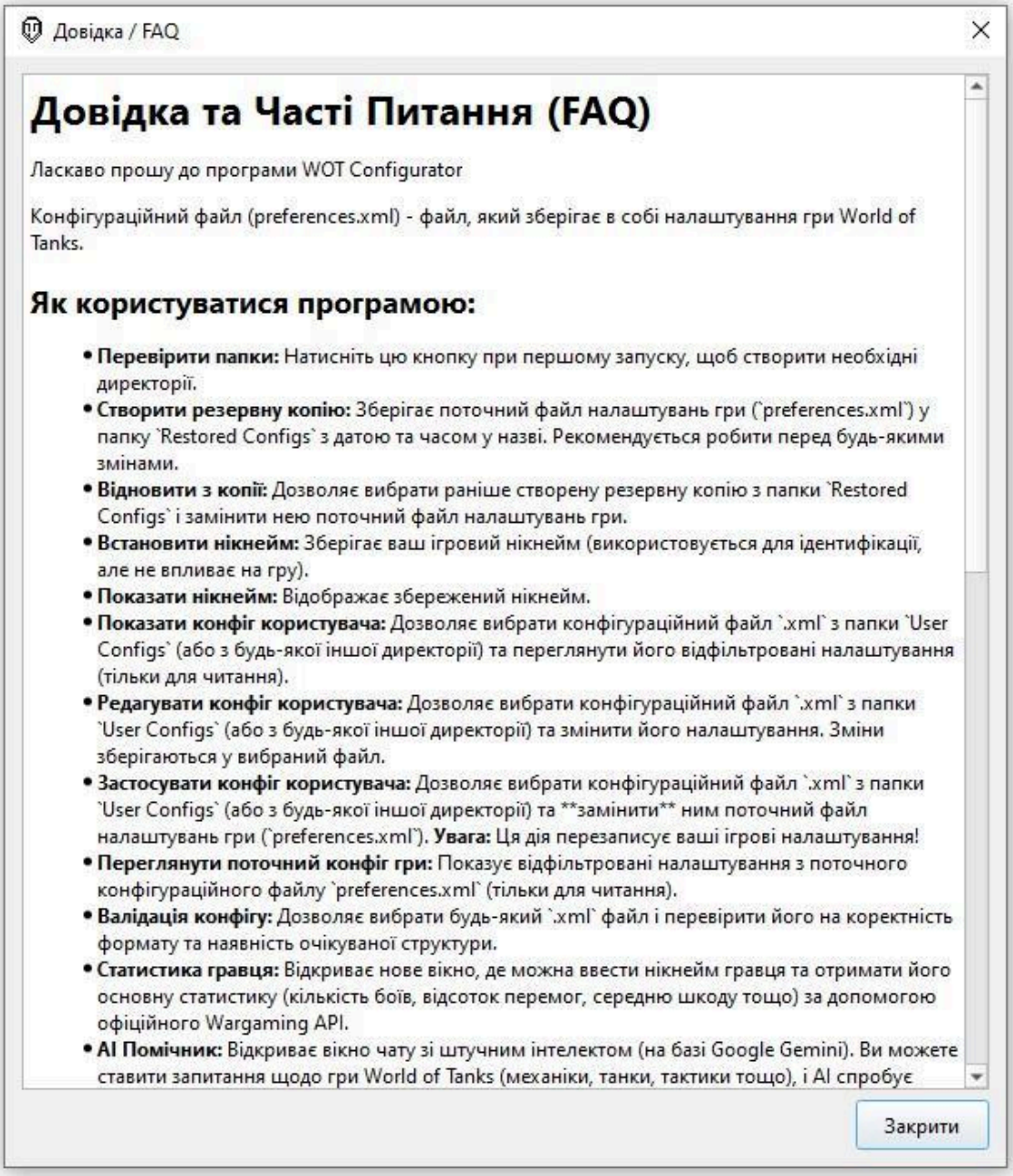


Рис. 4.12 Вікно Довідки та Частих Питань (FAQ)

Це вікно містить текстову інформацію про призначення програми, файл конфігурації та покрокові інструкції щодо використання основних функцій.

Реалізація цього вікна включає відображення форматованого тексту та кнопку "Закрити".

Вікно перегляду налаштувань:

Для візуалізації вмісту файлу preferences.xml у зручному для читання форматі реалізовано окреме вікно перегляду, яке показано на рисунку 4.13.

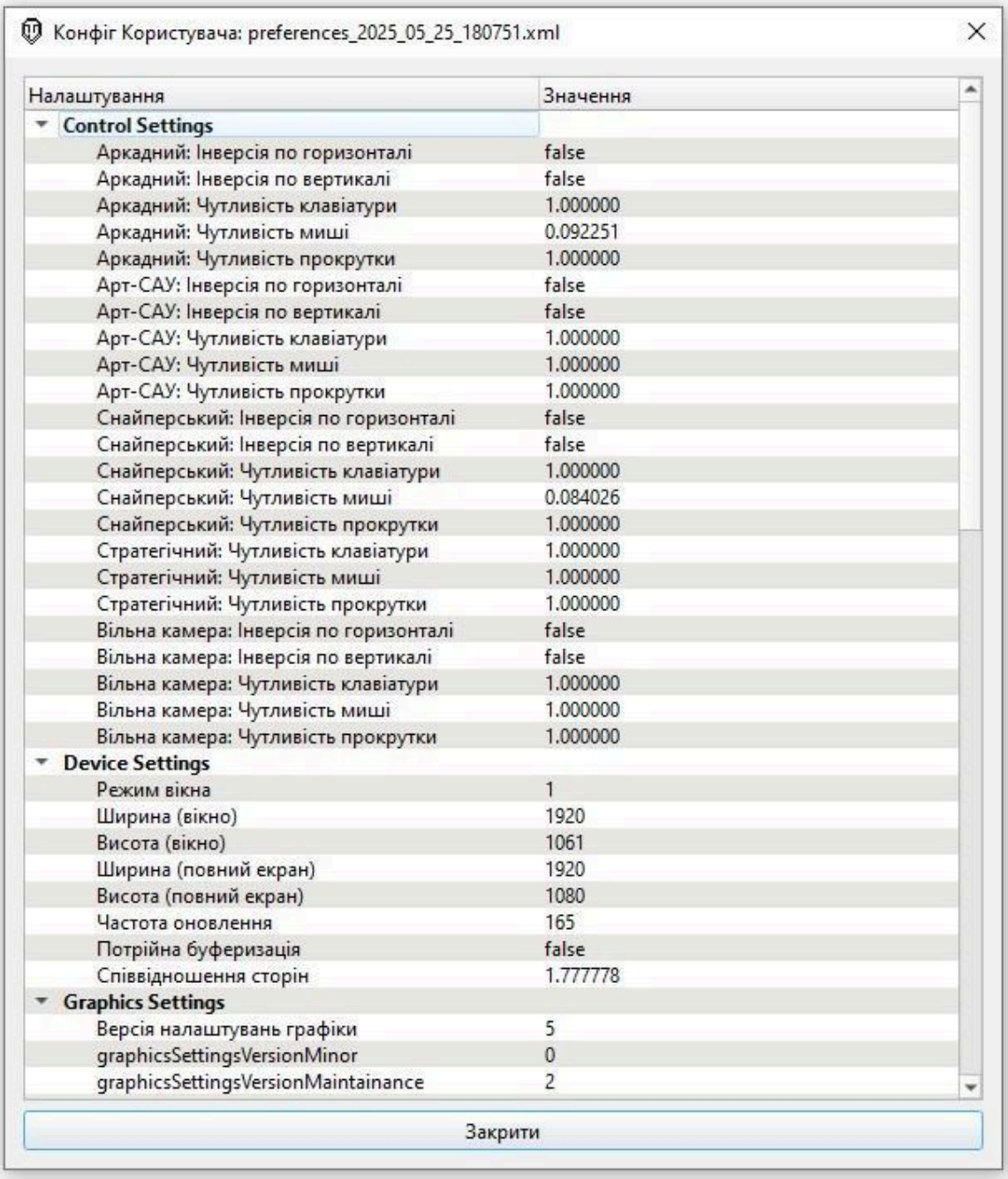


Рис. 4.13 Вікно перегляду налаштувань

Це вікно використовує табличний або деревоподібний віджет Qt для відображення ієрархічної структури XML-налаштувань. Налаштування згруповані за секціями ("Control Settings", "Device Settings", "Graphics Settings" тощо). Кожен рядок відображає назву налаштування та його поточне значення. Це вікно призначене лише для читання і допомагає користувачеві швидко оглянути всі параметри. Дані для цього вікна отримуються від ConfigManager після парсингу preferences.xml.

Вікно редагування налаштувань:

Ключовим елементом інтерфейсу є вікно редагування налаштувань, яке дозволяє користувачеві безпосередньо змінювати значення параметрів конфігурації. Його зовнішній вигляд представлений на рисунку 4.14.1 та рисунку 4.14.2.

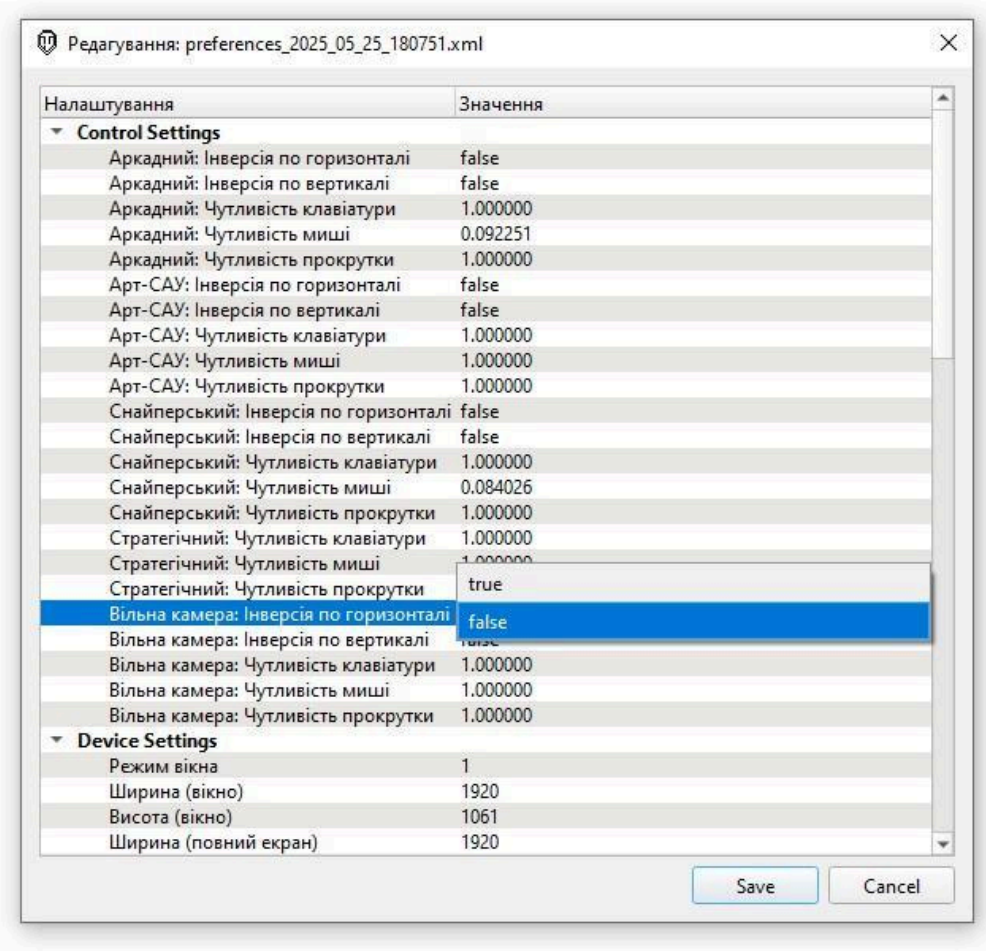


Рис. 4.14.1 Вікно редагування налаштувань

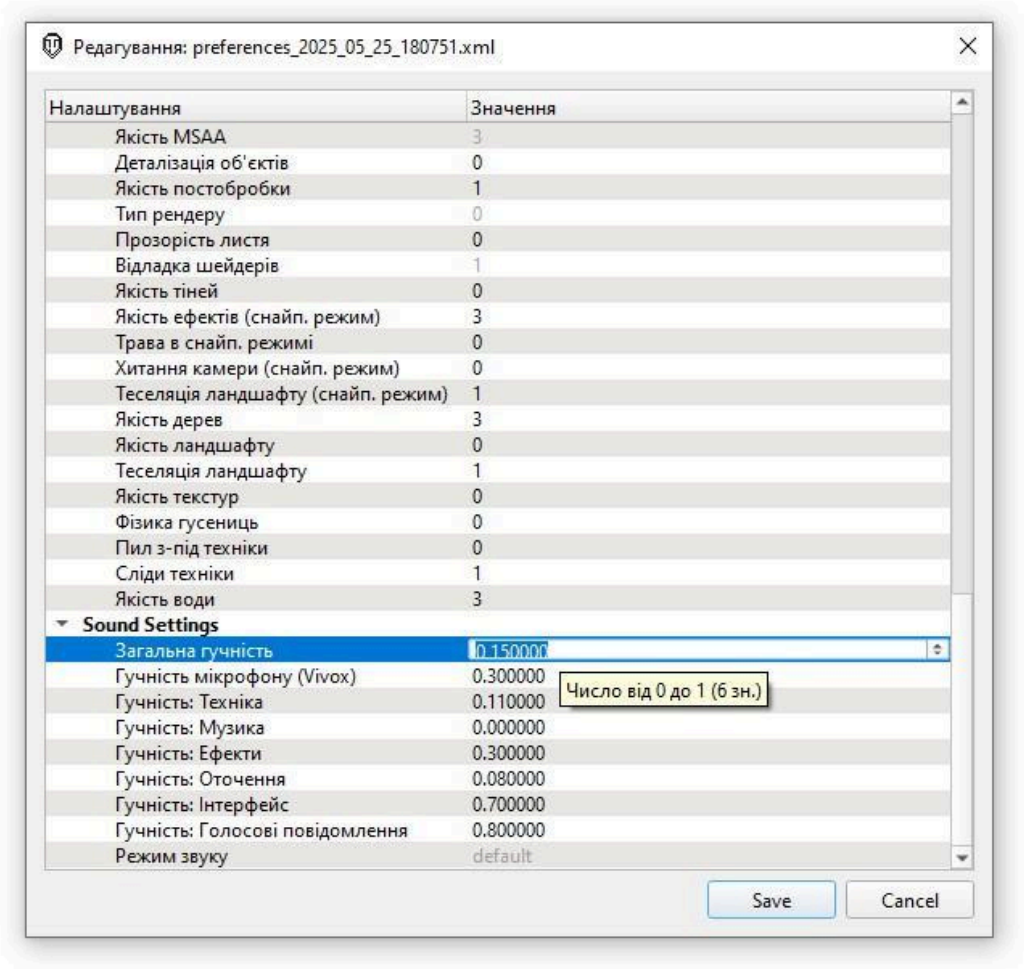


Рис. 4.14.2 Вікно редагування налаштувань

Ці вікна мають схожу структуру з вікном перегляду – налаштування також відображаються у вигляді таблиці або дерева. Однак, у стовпці "Значення" замість простого тексту використовуються відповідні елементи керування (віджети) Qt, які дозволяють змінювати значення:

– Для логічних значень (true/false) можуть використовуватися чекбокси або випадаючі списки.

– Для числових значень (цілих або з плаваючою точкою) можуть використовуватися поля вводу, валідатори яких обмежують тип даних, або повзунки для значень у певному діапазоні.

– Для значень з обмеженого набору варіантів можуть використовуватися випадаючі списки.

На Рисунку 4.14.2 видно спливаючу підказку "Число від 0,01 до 1 (2 зн.)", що вказує на використання валідаторів або специфічних делегатів редагування Qt (settingdelegate), які забезпечують введення даних у правильному форматі та діапазоні.

Кнопки "Save" та "Cancel" дозволяють зберегти внесені зміни або відхилити їх відповідно. При натисканні "Save", зміни з елементів керування зчитуються, передаються в структуру даних конфігурації та, через ConfigManager, записуються назад у файл preferences.xml (після повторної валідації).

Вікно статистики гравця:

Для доступу до ігрової статистики гравця через Wargaming.net Public API реалізовано окреме діалогове вікно, яке показане на рисунку 4.15.

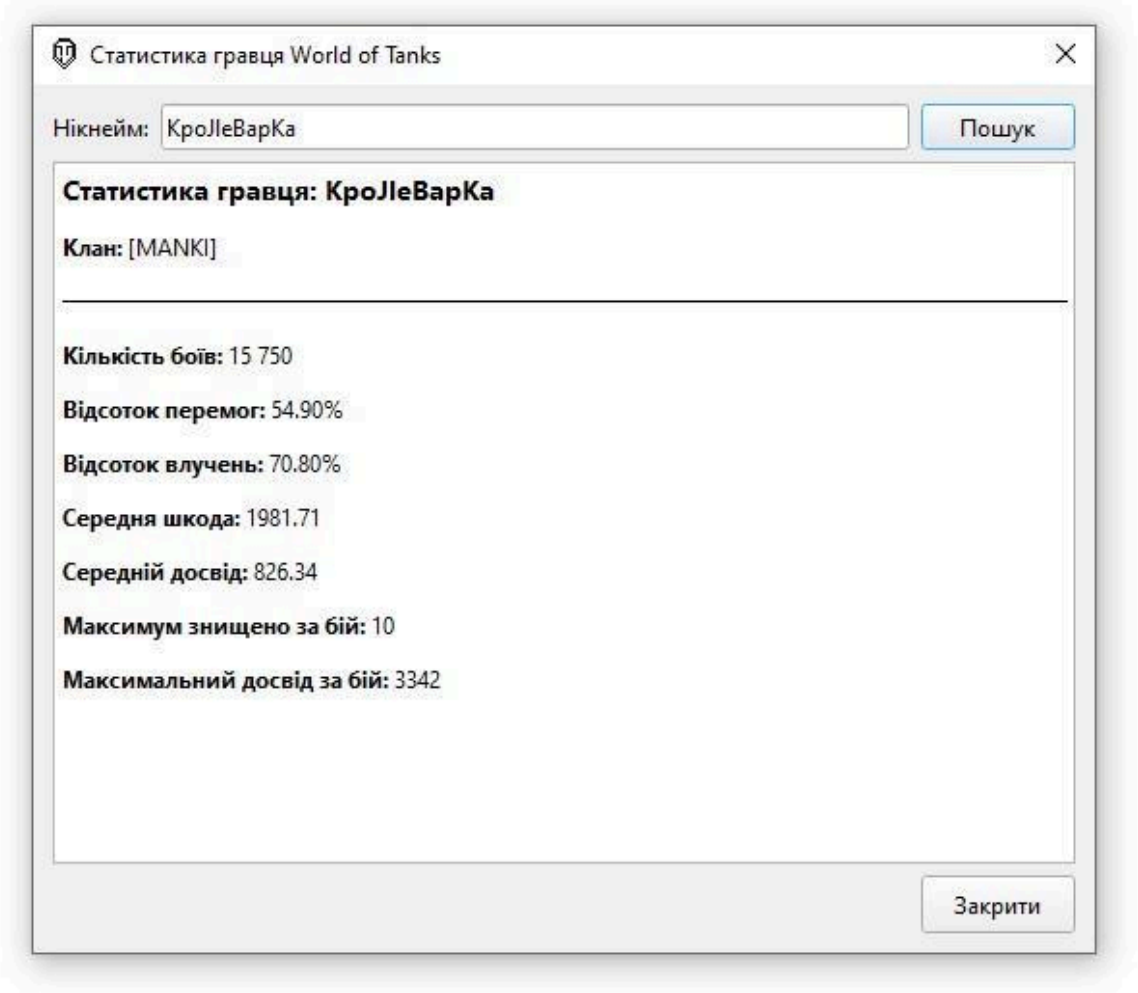


Рис. 4.15 Вікно статистики гравця

Це вікно містить поле вводу для нікнейму гравця та кнопку "Пошук". Після натискання "Пошук", застосунок відправляє запит до Wargaming.net API, отримує дані статистики та відображає їх у відповідних полях вікна (клан, кількість боїв, відсоток перемог, середня шкода тощо). Реалізація цього функціоналу включає використання бібліотек Qt для мережевих запитів (QtNetwork) та обробки отриманої відповіді (у форматі JSON).

Вікно AI Помічника Танкіста:

Для взаємодії з AI-асистентом на базі Google Gemini API реалізовано діалогове вікно чату, яке показане на рисунку 4.16.

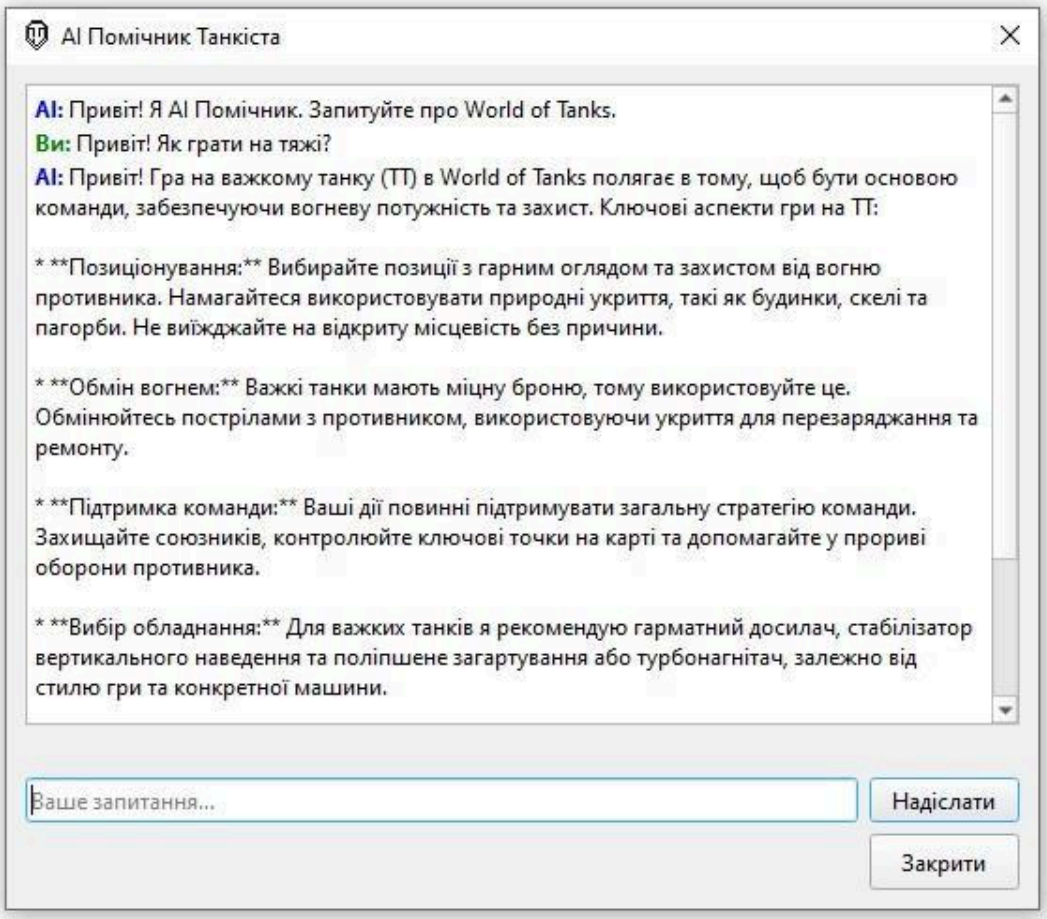


Рис. 4.16 Вікно AI Помічника Танкіста

Це вікно надає інтерфейс чату, де користувач може вводити запитання у поле вводу та відправляти їх кнопкою "Надіслати". Відповіді від AI відображаються в історії чату. Реалізація цього вікна включає відправку запитів до Google Gemini API з використанням мережевих можливостей Qt та відображення отриманих відповідей.

Зв'язок UI з логікою:

Реалізація інтерфейсу користувача тісно пов'язана з логікою застосунку, реалізованою в шарах Core та DataAccess. Обробники подій (слоти) в класах вікон (MainWindow, ConfigEditDialog тощо) викликають відповідні методи у класах ConfigManager, BackupManager, ProfileManager, FileValidator для виконання операцій з даними та файлами. Наприклад, натискання кнопки "Створити резервну копію" викликає метод у BackupManager, а натискання "Save" у вікні редагування викликає методи у ConfigEditor та ConfigManager.

Такий підхід, де UI відповідає лише за візуалізацію та збір введення, а основна логіка знаходиться в окремих класах, відповідає принципам багатошарової архітектури та забезпечує чистоту коду, полегшує тестування (можна тестувати логіку окремо від UI) та подальшу підтримку.

4.3 Тестування застосунку

Процес тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення, що дозволяє переконатися у відповідності реалізованої системи поставленим вимогам, виявити та усунути дефекти, а також гарантувати стабільність та надійність роботи застосунку. Для застосунку "WOT Configurator" тестування відіграло ключову роль у забезпеченні коректної взаємодії з файлами конфігурації гри World of Tanks та зовнішніми сервісами, а також у наданні користувачеві інтуїтивно зрозумілого та безпомилкового інтерфейсу.

Тестування проводилось в умовах, що максимально наближені до типового середовища використання застосунку кінцевим користувачем. Робочим середовищем виступав персональний комп'ютер під управлінням операційної

системи Windows 10, оснащений процесором Intel Core i5-11400f та 32GB DDR4 оперативної пам'яті. Це конфігурація, що є достатньо потужною для комфортного запуску як самої гри World of Tanks, так і супутніх інструментів, подібних до "WOT Configurator". Наявність встановленої гри World of Tanks була обов'язковою умовою, оскільки застосунок безпосередньо взаємодіє з її конфігураційними файлами та, у випадку запитів статистики, з її сервісами.

4.3.1 Модульне тестування

Модульне тестування (Unit Testing) є першим етапом у піраміді тестування, який фокусується на перевірці окремих, ізольованих компонентів або "модулів" вихідного коду. Метою модульного тестування є підтвердження того, що кожен функціональний блок програми працює коректно та відповідно до свого призначення перед інтеграцією з іншими частинами системи.

Для розробки "WOT Configurator" модульне тестування відіграло ключову роль у забезпеченні якості та надійності низькорівневих компонентів. Оскільки застосунок написаний на C++ та використовує бібліотеку Qt, для модульного тестування було обрано фреймворк Qt Test. Тестування було проведено в рамках основного проекту, що спрощує процес розробки та інтеграції тестів.

Ключові аспекти та результати модульного тестування:

Модульні тести були розроблені для перевірки критичних функціональних блоків застосунку, забезпечуючи їх коректну роботу в ізоляції. Нижче наведено перелік протестованих модулів та досягнуті результати:

– TestAppInitializer: Цей тестовий клас перевіряв коректність ініціалізації додатку, зокрема функціонал checkFolders(), який відповідає за перевірку та створення необхідних директорій.

– TestFileValidator: Один із найважливіших модулів, відповідальний за валідацію preferences.xml. Тести фокусувалися на методі validateFile(), перевіряючи коректність визначення синтаксичних та структурних помилок в XML-файлах.

– TestBackupManager: Тестовий клас для модуля, що керує створенням резервних копій та їх відновленням. Перевірялася коректність збереження та завантаження конфігурацій.

– TestConfigManager: Цей модуль відповідає за управління конфігураційними файлами гри World of Tanks. Тести включали перевірку функціоналу getCurrentGameConfigPath() (визначення шляху до конфігурації), viewCurrentGameConfigContent() (перегляд вмісту конфігурації) та uploadConfig() (завантаження конфігурації).

Загальний результат модульного тестування:

Було розроблено та успішно виконано 13 модульних тестів, результат модульного тестування відображений на рисунку 4.3.1, що підтверджує стабільність та надійність базових компонентів застосунку. Усі тести пройшли без єдиної помилки, що свідчить про високу якість реалізації ключових функціональних модулів.

Автоматизоване модульне тестування забезпечило швидке виявлення та виправлення дефектів на ранніх стадіях розробки, що значно покращило загальну якість коду та зменшило ризики виникнення проблем на подальших етапах.

```
21:05:39: Starting C:\Users\Kroll\Desktop\Диплом\TESTS\WOTConfiguratorGUITests\build\Desktop_Qt_6_8_2_MinGW_64_bit-Debug\all_tests.exe...
********* Start testing of TestAppInitializer *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestAppInitializer::initTestCase()
PASS   : TestAppInitializer::testCheckFolders()
PASS   : TestAppInitializer::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped, 0 blacklisted, 19ms
********* Finished testing of TestAppInitializer *********
********* Start testing of TestFileValidator *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestFileValidator::initTestCase()
PASS   : TestFileValidator::testValidateFile()
PASS   : TestFileValidator::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped, 0 blacklisted, 3ms
********* Finished testing of TestFileValidator *********
********* Start testing of TestBackupManager *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestBackupManager::initTestCase()
PASS   : TestBackupManager::cleanupTestCase()
Totals: 2 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms
********* Finished testing of TestBackupManager *********
********* Start testing of TestConfigManager *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestConfigManager::initTestCase()
PASS   : TestConfigManager::testGetCurrentGameConfigPath()
PASS   : TestConfigManager::testViewCurrentGameConfigContent()
PASS   : TestConfigManager::testUploadConfig()
PASS   : TestConfigManager::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms
********* Finished testing of TestConfigManager *********
21:05:39: C:\Users\Kroll\Desktop\Диплом\TESTS\WOTConfiguratorGUITests\build\Desktop_Qt_6_8_2_MinGW_64_bit-Debug\all_tests.exe exited with code 0
```

Рис. 4.3.1 Успішне проходження всіх модульних тестів.

4.3.2 Ручне функціональне тестування

Основним методом контролю якості на рівні користувацького інтерфейсу та взаємодії стало ручне функціональне тестування. Цей підхід передбачав неперервну перевірку працездатності застосунку безпосередньо під час процесу реалізації. Таке ітеративне тестування виконувалося після кожного значного внесення змін у кодову базу або після завершення реалізації окремого функціонального блоку. Це дозволило оперативно виявляти помилки та інтеграційні проблеми ще на ранніх етапах, забезпечуючи швидке їх виправлення та мінімізуючи ризик накопичення дефектів до кінця розробки. По суті, кожен етап кодування завершувався етапом тестування реалізованої функціональності, що створювало тісний цикл зворотного зв'язку.

Спектр тестових сценаріїв охоплював увесь реалізований функціонал застосунку. Кожна функція, доступна через головне вікно та діалогові вікна, була ретельно перевірена. Ключові напрямки тестування включали:

– Перевірка коректності визначення шляху до файлу preferences.xml: Тестувалося, чи правильно застосунок знаходить розташування стандартного файлу налаштувань гри в директорії %APPDATA% на різних системах або при різних налаштуваннях середовища.

– Тестування функцій читання та відображення конфігурації: Перевірялося, чи застосунок успішно зчитує вміст файлу, чи коректно парситься XML-структура за допомогою Pugixml, та чи правильно відображаються всі налаштування у вікні перегляду та редагування.

– Тестування функцій редагування та збереження конфігурації: Це один з найважливіших блоків. Перевірялося, чи користувач може змінювати значення різних типів налаштувань (числа, булеві значення) через відповідні елементи керування в вікні редагування. Особлива увага приділялася перевірці граничних та некоректних значень (наприклад, введення тексту замість числа, числа поза допустимим діапазоном), щоб переконатися, що валідація працює належним чином і застосунок не дозволяє зберегти дані, які можуть пошкодити файл або спричинити помилки в грі. Перевірялося, чи зміни зберігаються у файл

preferences.xml у коректному XML-форматі та чи гра "бачить" ці зміни при наступному запуску.

– Тестування застосування користувацького конфігу: Перевірялося, чи функція "Замінити поточний конфігураційний файл" коректно копіює вибраний користувачем файл на місце ігрового preferences.xml та чи оновлюються налаштування в грі.

– Перевірка створення резервної копії: Тестувалося, чи при натисканні відповідної кнопки створюється нова резервна копія поточного preferences.xml у вказаній директорії, чи генерується коректне ім'я файлу з часовою міткою, та чи збережений файл є валідним.

– Перевірка відновлення з резервної копії: Тестувалося, чи функція відновлення правильно замінює поточний preferences.xml на вибрану резервну копію з каталогу "Restored Configs" та чи коректно завантажуються відновлені налаштування в грі.

– Перевірка валідації коректних файлів: Тестувалося, чи застосунок успішно валідує стандартний preferences.xml або інші відомо коректні файли конфігурації, повідомляючи про успіх або лише про некритичні попередження (структури/значень), і чи дозволяє продовжувати операції.

– Перевірка валідації некоректних файлів: Створювалися та модифікувалися тестові файли preferences.xml з синтаксичними помилками XML, відсутніми ключовими секціями або некоректними значеннями. Тестувалося, чи FileValidator коректно виявляє ці помилки, надає зрозуміле повідомлення про причину і чи застосунок блокує потенційно небезпечні операції (наприклад, збереження або застосування такого файлу).

– Тестування функції "Пошук статистики за нік-неймом": Перевірялася коректність відправки запиту до Wargaming.net Public API з введеним нікнеймом та правильність відображення отриманих даних статистики у відповідному вікні. Тестувалася поведінка при введенні неіснуючого нікнейму або при проблемах з інтернет-з'єднанням.

– Тестування функції "ШІ: Розумний помічник танкіста": Перевірялась коректність відправки запитів до Google Gemini API з введеними запитаннями та відображення отриманих відповідей AI у вікні чату.

– Тестування вікна Довідки (FAQ): Перевірялось коректне відкриття та відображення інформації у вікні.

– Перевірка логування: Відстежувалося, чи коректно записуються важливі системні події та дії користувача у лог-файл (logs.txt).

Загальним результатом проведеного ручного функціонального тестування стало підтвердження того, що реалізований застосунок "WOT Configurator" повністю відповідає поставленим вимогам та працює стабільно.

Усі ключові функції, включаючи управління конфігурацією, резервне копіювання, валідацію, взаємодію із зовнішніми API та роботу користувацького інтерфейсу, функціонують належним чином.

В процесі розробки було виявлено та виправлено низку дрібних недоліків, що дозволило представити користувачеві готовий до використання, надійний інструмент.

4.3.3 Користувацьке тестування (опитування)

Для отримання об'єктивної оцінки користувацького досвіду, зручності використання, стабільності та функціональності застосунку "WOT Configurator" було проведено онлайн-опитування серед 12-и користувачів.

Опитування включало кількісні питання з оцінкою за 10-бальною шкалою (де 1 – дуже погано, 10 – відмінно) та відкриті питання для збору якісних відгуків.

Результати кількісного оцінювання:

Середні бали за ключовими критеріями представлені в таблиці 4.1.

Таблиця 4.1

Середні оцінки користувачів застосунку "WOT Configurator"

Критерій оцінювання	**Середній бал (з 10)**
Автоматичне виявлення шляху до гри World of Tanks	10.0
Зручність створення резервних копій конфігурацій	9.83
Зручність відновлення конфігурацій з резервних копій	9.83
Зручність перегляду налаштувань через графічний інтерфейс	9.0
Зручність редагування налаштувань через графічний інтерфейс (зрозумілість опцій, легкість зміни значень)	9.17
Ефективність валідації XML-файлу (запобігання помилкам, коректність перевірки)	10.0
Функціонал керування нік-неймами (додавання, редагування, видалення, відображення)	10.0
Зручність пошуку статистики гравців	9.92
Зручність та користь ШІ-помічника	9.92
Загальна стабільність роботи застосунку (відсутність збоїв, зависань)	9.67
Швидкодія застосунку (швидкість завантаження, збереження, виконання операцій)	10.0
Інтуїтивність та зрозумілість користувацького інтерфейсу	9.75
Привабливість візуального дизайну застосунку	9.0
Наскільки застосунок "WOT Configurator" спростив Вам управління налаштуваннями гри World of Tanks?	10.0
Ймовірність рекомендації "WOT Configurator" іншим гравцям World of Tanks	10.0

Аналіз кількісних даних демонструє високий рівень задоволеності користувачів. Застосунок отримав максимальні оцінки (10.0) за такі ключові аспекти, як автоматичне виявлення шляху до гри, ефективність валідації XML-файлу, функціонал керування нік-неймами, швидкість завантаження та збереження операцій, спрощення управління конфігураціями та ймовірність рекомендації. Це підтверджує, що основні функціональні вимоги були успішно реалізовані та відповідають очікуванням цільової аудиторії.

Дещо нижчі, але все ще високі оцінки були отримані за "Привабливість візуального дизайну застосунку" (9.0), "Зручність перегляду налаштувань через графічний інтерфейс" (9.0), "Зручність редагування налаштувань через графічний інтерфейс" (9.17) та "Загальна стабільність роботи застосунку" (9.67). Це вказує на потенційні напрямки для подальших покращень, хоча загалом ці аспекти також оцінені дуже позитивно.

Аналіз якісних відгуків:

Відкриті питання дозволили зібрати цінні коментарі та пропозиції. Серед них виділяються наступні категорії:

Запитувані функції: Користувачі висловили зацікавленість у додаванні:

– готових конфігурацій обладнання від відомих/найкращих гравців або найпопулярніших конфігурацій відповідно до класу техніки;

– можливості прямого скачування з офіційного сайту WOT;

– додання модів, як інноваційної продукції для управління грою;

– більш повної статистики гравців (наприклад, WN8, EFF, WTR тощо);

– можливості керування своїм ігровим акаунтом та навіть кланом (що є значним розширенням функціоналу).

Виявлені проблеми та труднощі:

Один з користувачів вказав на труднощі з навігацією та пошуком функцій при першому запуску, запропонувавши змінити інтерфейс та додати більше кольорів до налаштувань та іконок, щоб зробити його більш зручним та комфортним.

Були також коментарі щодо візуального дизайну, зокрема, що він "трохи старувато виглядає, як Windows XP".

Переважна більшість користувачів не стикалися з труднощами або були задоволені роботою, що підкреслює загальну зручність.

Загальні коментарі та пропозиції: Переважна більшість відгуків була позитивною, користувачі висловлювали задоволення роботою застосунку, відзначали його естетичність та ефективність. Деякі респонденти відзначили, що "застосунок повноцінний".

Висновки за результатами опитування:

Проведене користувацьке тестування за участі 12-и респондентів підтвердило високу затребуваність та ефективність застосунку "WOT Configurator". Застосунок успішно вирішує поставлені задачі, значно спрощуючи управління конфігураціями гри World of Tanks та підвищуючи безпеку цього процесу. Отримані якісні відгуки надають чіткі напрямки для майбутнього розвитку, зокрема щодо розширення функціоналу, покращення візуального дизайну та зручності першого знайомства з інтерфейсом. Загалом, застосунок отримав дуже високу оцінку та демонструє великий потенціал для подальшого вдосконалення..

4.4 Управління версіями та розміщення проекту на GitHub

В процесі розробки програмного забезпечення, особливо для проектів високої складності, використання систем контролю версій (Version Control Systems, VCS) є критично важливим. Вони дозволяють ефективно відстежувати зміни у коді, керувати різними версіями проекту, полегшують спільну роботу (навіть якщо над проектом працює одна людина) та забезпечують можливість відкату до попередніх станів кодової бази.

Для проекту "WOT Configurator" як система контролю версій був обраний Git – одна з найпопулярніших розподілених VCS у світі. Його гнучкість та

потужний функціонал ідеально підходять для індивідуальної розробки з необхідністю фіксації етапів роботи.

Для централізованого зберігання репозиторію та зручного доступу до нього був використаний веб-сервіс GitHub. GitHub є провідною платформою для хостингу Git-репозиторіїв, що надає додаткові можливості, такі як відстеження завдань, запити на злиття (pull requests) та інструменти для співпраці.

Процес управління проектом з використанням Git та GitHub включав наступні ключові етапи:

– Створення віддаленого репозиторію на GitHub: На початку роботи над проектом був створений новий репозиторій на платформі GitHub, який став центральним сховищем кодової бази.

– Ініціалізація локального Git-репозиторію: У кореневій директорії проекту "WOT Configurator" був ініціалізований локальний Git-репозиторій. Це дозволило Git почати відстежувати зміни у файлах проекту.

– Фіксація змін (Commit): Впродовж усього періоду розробки, після реалізації певного функціоналу, виправлення помилки або внесення інших логічно завершених змін, виконувалася фіксація цих змін у локальному репозиторії за допомогою команди git commit. Кожен коміт супроводжувався змістовним повідомленням, що коротко описувало суть внесених змін (наприклад, "v0.15 AI integration & UI fix", "v0.7 current cfg replacement & check"). Це дозволило створити детальну історію розвитку проекту.

– Синхронізація з віддаленим репозиторієм (Push): Регулярно, після виконання кожного коміту, всі зафіксовані зміни надсилалися до віддаленого репозиторію на GitHub за допомогою команди git push. Це забезпечувало централізоване зберігання актуальної версії коду та його доступність.

Таким чином, використання Git та GitHub дозволило ефективно управляти процесом розробки застосунку "WOT Configurator", мати повну історію змін, забезпечити безпеку кодової бази завдяки віддаленому сховищу та продемонструвати використання стандартних інструментів розробки.

ВИСНОВКИ

В рамках даної дипломної роботи було успішно розроблено та реалізовано програмне забезпечення – застосунок "WOT Configurator" – призначений для спрощення процесу управління файлом конфігурації гри World of Tanks (preferences.xml). Реалізація застосунку дозволила вирішити проблему незручного ручного редагування XML-файлу, надавши користувачеві інтуїтивно зрозумілий графічний інтерфейс та додаткові функції, що підвищують безпеку та зручність налаштування гри.

В ході виконання роботи було успішно вирішено наступні ключові завдання:

– Проаналізовано предметну область та визначено вимоги: Була детально проаналізована структура файлу preferences.xml та специфіка управління налаштуваннями гри. Визначено потреби користувачів у зручному інструменті для зміни параметрів, створення резервних копій та валідації файлу. Проведено аналіз існуючих рішень та виявлено їх переваги та недоліки. На основі аналізу сформовано функціональні та нефункціональні вимоги до майбутнього програмного забезпечення, включаючи вимоги до інтерфейсу, безпеки даних (резервне копіювання, валідація) та інтеграції із зовнішніми сервісами (Wargaming.net API, Google Gemini API).

– Обрано засоби реалізації: Виходячи з вимог до кросплатформності, продуктивності та зручності розробки графічного інтерфейсу, було обрано мову програмування C++ та фреймворк Qt як основні засоби реалізації. Для роботи з XML-файлами обрано бібліотеку Pugixml. Для файлових операцій використовувалася стандартна бібліотека <filesystem>. Середовищем розробки виступало Visual Studio. Для управління версіями коду використовувався Git, а для централізованого зберігання – GitHub.

– Виконано проектування програмного забезпечення: Розроблено архітектуру застосунку на основі багатошарового підходу, що включає шари інтерфейсу користувача, ядра та доступу до даних. Створено комплект діаграм з

використанням нотації UML, що відображають різні аспекти системи: контекстна діаграма (що ілюструє взаємодію із зовнішнім середовищем, включаючи користувача, файли та зовнішні API), діаграма прецедентів (що описує функціональні вимоги з точки зору користувача), діаграма класів (що деталізує структуру програмних класів та їх зв'язки), діаграми послідовності та діяльності (що моделюють динамічну поведінку системи при виконанні ключових операцій), а також діаграми компонентів та розгортання (що показують фізичну структуру застосунку та його розміщення).

– Реалізовано програмне забезпечення: Згідно з розробленою архітектурою та проектом, реалізовано ключові функціональні модулі застосунку на мові C++ з використанням фреймворку Qt та бібліотеки Pugixml. Розроблено користувацький інтерфейс з використанням Qt Designer та C++ коду, що забезпечує зручну взаємодію користувача з налаштуваннями. Реалізовано функції читання, парсингу, редагування та збереження XML-файлу конфігурації, механізми резервного копіювання та відновлення, а також систему валідації файлів. Виконано інтеграцію із зовнішніми сервісами – Wargaming.net Public API (для отримання ігрової статистики) та Google Gemini API (для функції AI-асистента). Проект розміщено та підтримувався на платформі GitHub з використанням системи контролю версій Git.

– Проведено тестування програмного забезпечення: Виконано ручне функціональне тестування застосунку після реалізації кожного функціонального блоку та внесення змін. Тестування охоплювало всі ключові функції та сценарії використання, включаючи роботу з файлами, резервними копіями, валідацію, редагування налаштувань та взаємодію з API. Перевірялась коректність обробки даних, реакція на некоректний ввід та граничні умови. Тестування проводилось в умовах, наближених до реального використання, на типовій апаратній конфігурації.

За результатами виконання дипломної роботи було створено повністю функціональний застосунок "WOT Configurator", який відповідає поставленим вимогам та готовий до використання. Досягнуті результати можуть бути

використані гравцями World of Tanks для спрощення процесу управління ігровими налаштуваннями.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей та стаття:

1. Солов’ян А.О., Гаманюк І.М. Забезпечення безпеки даних у застосунку WOT Configurator. // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, С.327-328.

2. Солов’ян А.О., Гаманюк І.М. Обробка конфігураційних даних для оптимізації налаштувань гри World of Tanks. // VI Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24 квітня 2025 року, ДУІКТ, м.Київ, С.563-565.

3. Солов’ян А.О., Гаманюк І.М. Розробка та реалізація програмного застосунку для ефективного управління конфігураціями гри World of Tanks. Матеріали №21 студентського наукового журналу «UNIVERSUM», 20.06.2025, подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Wargaming. Як вручну змінити налаштування графіки? 25.07.2024. URL: https://wargaming.net/support/uk/products/wot/article/15735/ (дата звернення 10.01.2025)

2. Fawcett J., Ayers D., Quin L., Beginning XML, Indianapolis, 2001. 800 с.

3. Wgmods. KorbenTeam Modpack. 16.04.2025. URL: https://wgmods.net/4963/ (дата звернення 20.05.2025)

4. Wotmods. WOT Tweaker Plus для World of Tanks 1.28.1.0. 16.04.2025. URL: https://wotmods.net.ua/prohramy-wot/11-wot-tweaker-plus (дата звернення 20.05.2025)

5. Kessel R., Learning Visual Studio Code, Birmingham, 2017. 296 с.

6. Summerfield M., Advanced Qt Programming: Creating Robust and Reusable Software, Boston, 2020. 608 с.

7. Martin B., Hoffman B., Professional CMake: A Practical Guide, Chapel Hill, NC, 2024. 640 с.

8. Stroustrup B., The C++ Programming Language, Boston, 2013, 1376 с.

9. Blanchette J., Summerfield M., C++ GUI Programming with Qt 6, Upper Saddle River, 2023. 608 с.

10. Pugixml. pugixml 1.15 manual. 10.01.2025. URL: https://pugixml.org/docs/manual.html (дата звернення 20.05.2025)

11. Chacon S., Straub B., Pro Git, Apress, Berkeley, CA, 2024. 496 с.

12. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston: Addison-Wesley Professional, 2021. 608 с.

13. Shalloway A., Trott J. R., et al. Design Patterns Explained: A New Perspective on Object-Oriented Design. 3rd ed. Boston: Addison-Wesley Professional, 2024. 528 с.

14. Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 4th ed. New York: Pearson, 2022. 752

c.

15. Freeman E., Robson E., et al. Head First Design Patterns. 2nd ed. Sebastopol, CA: O'Reilly Media, 2021. 650 c.

16. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 4th ed. Boston: Addison-Wesley Professional, 2023. 224 c.

17. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 3rd ed. Boston: Addison-Wesley Professional, 2022. 528 c.

18. Sommerville I. Software Engineering. 12th ed. Harlow: Pearson, 2021. 832 c.

19. Cockburn A. Writing Effective Use Cases. 2nd ed. Boston: Addison-Wesley Professional, 2024. 400 c.

20. Kerievsky J. Refactoring to Patterns. Boston: Addison-Wesley Professional, 2024. 400 c.

21. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. New York: Pearson, 2023. 432 c.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмного забезпечення для управління конфігураціями гри World of Tanks

Виконав студент 4 курсу
групи ПД-44
Солов'ян Арсен Олександрович
Керівник роботи
ст. викладач кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення та підвищення надійності процесу керування конфігураційними файлами гри World of Tanks, а також розширення можливостей гравців шляхом інтеграції доступу до актуальної ігрової статистики та довідкової інформації.

Об'єкт дослідження: процес управління користувацькими конфігураціями та доступу до супутньої ігрової інформації у грі World of Tanks.

Предмет дослідження: методи та програмний застосунок для автоматизації та спрощення процесу керування конфігураційними файлами гри World of Tanks та інтеграції додаткових інформаційних сервісів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Дослідити існуючі проблеми керування конфігураціями гри World of Tanks.
2. Здійснити огляд та проаналізувати існуючі застосунки, які використовуються для керування конфігураціями гри World of Tanks.
3. Здійснити вибір технологій та інструментів для реалізації десктопного застосунку.
4. Визначити функціональні та нефункціональні вимоги до застосунку.
5. Реалізувати запланований функціонал та інтерфейс користувача застосунку.
6. Провести тестування працездатності, стабільності та зручності використання розробленого застосунку.

3

АНАЛІЗ АНАЛОГІВ

	WotStarter	KorbenTeam Modpack	Wot Tweaker Plus	WoT Settings & Fixes	WOT Configurator
Заміна на існуючий cfg файл	-	+	-	-	+
Створення резервної копії cfg файлу	-	-	-	-	+
Відновлення налаштувань з резервної копії cfg файлу	-	-	-	-	+
Відображення поточних налаштувань гри	-	-	-	+	+
Зміна налаштувань гри	+	-	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

Застосунок повинен:

1. Надавати можливість зберігати поточну конфігурацію гри World of Tanks (файл preferences.xml) у вигляді резервної копії.
2. Надавати можливість завантажувати вибрану конфігурацію, замінюючи нею активний файл preferences.xml.
3. Автоматично визначати шлях до каталогу встановленої гри World of Tanks.
4. Надавати можливість перегляду вмісту файлу конфігурації preferences.xml у структурованому вигляді через графічний інтерфейс.
5. Надавати можливість редагування ключових параметрів конфігурації через графічний інтерфейс (без необхідності прямого редагування XML).
6. Зберігати зміни, внесені користувачем, у файл preferences.xml.
7. Надавати можливість відновлення конфігурації з раніше створених резервних копій.
8. Дозволяти користувачеві зберігати різні конфігурації під унікальними іменами (профілями).
9. Перевіряти коректність структури XML файлу preferences.xml перед збереженням змін або завантаженням.

Нефункціональні вимоги:

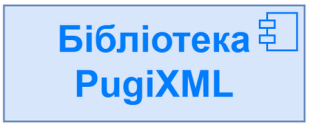
1. Операції збереження, завантаження та валідації конфігураційного файлу повинні виконуватися не довше 3-х секунд.
2. Застосунок повинен гарантувати цілісність конфігураційного файлу під час операцій збереження та редагування.
3. Наявність механізму резервного копіювання.
4. Застосунок повинен бути сумісним з операційною системою Windows 10.
5. Код програми повинен бути прокоментованим для полегшення подальших модифікацій та виправлення помилок.
6. Мати графічний інтерфейс користувача (GUI) для доступу до всіх перелічених функцій.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

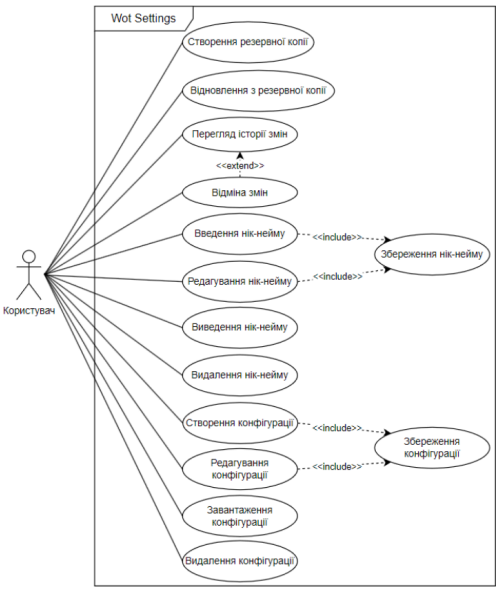
Фреймворк Qt

Visual Studio Code

Inno Setup

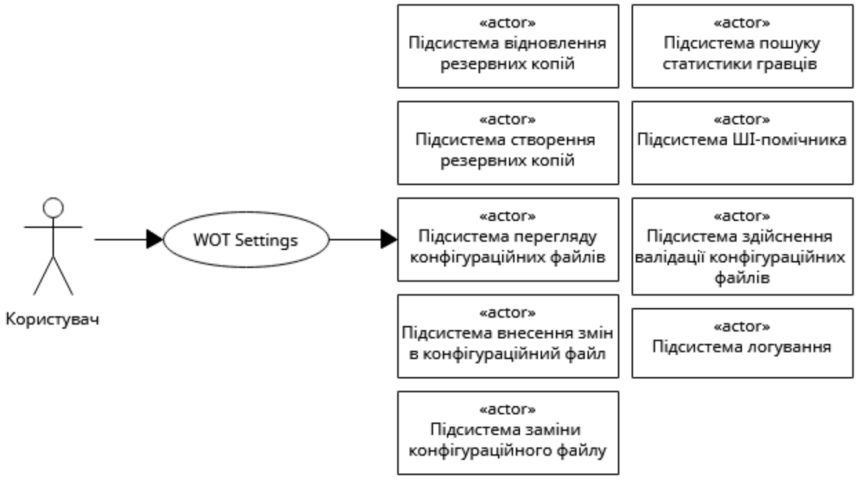


МОДЕЛЮВАННЯ ФУНКЦІОНАЛЬНОСТІ



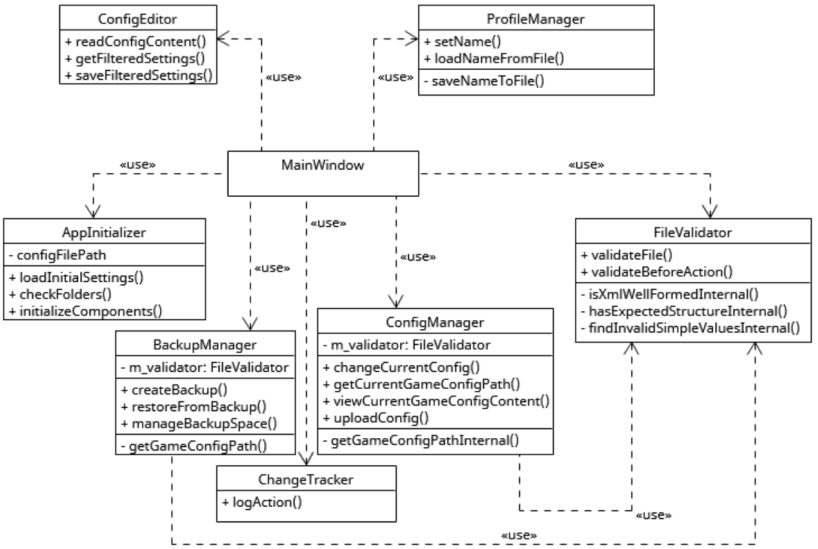
7

КОНТЕКСТНА ДІАГРАМА



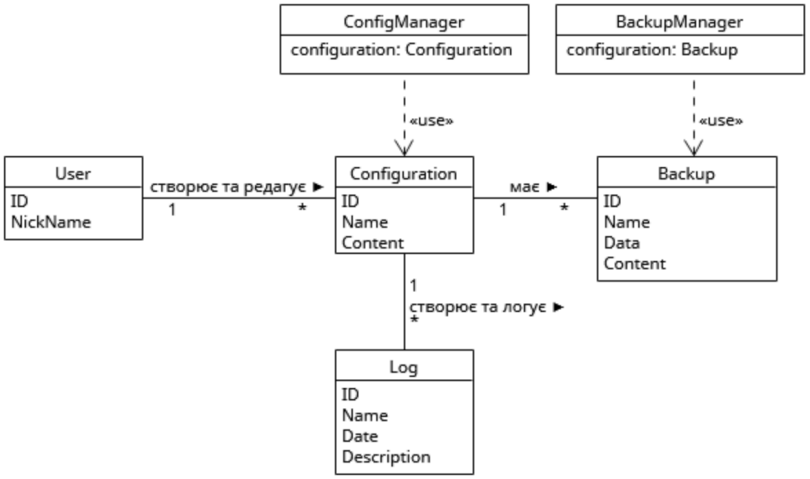
8

ДІАГРАМА КЛАСІВ



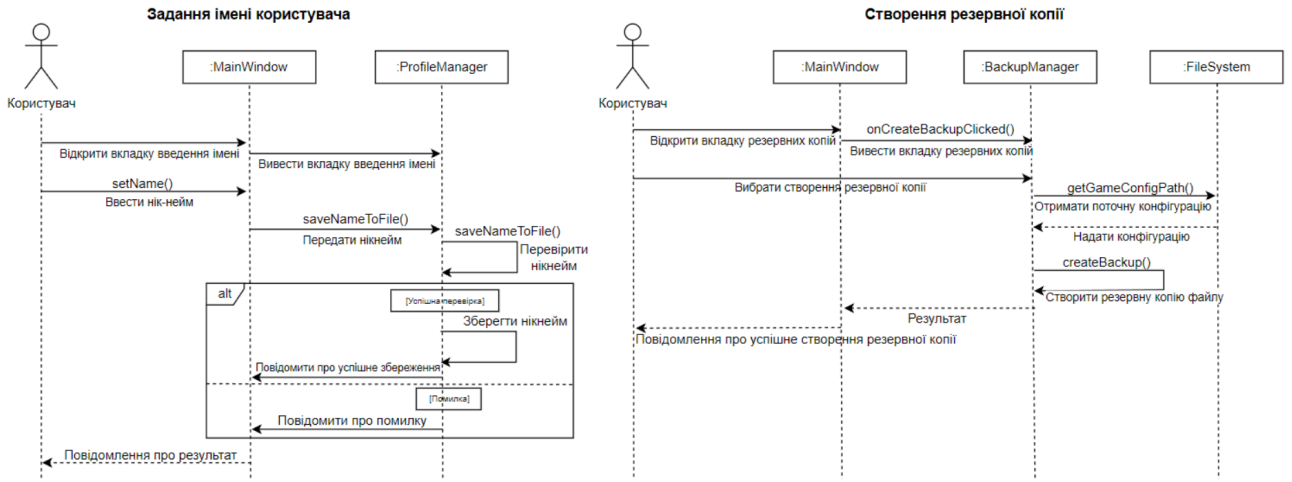
9

Діаграма предметної галузі



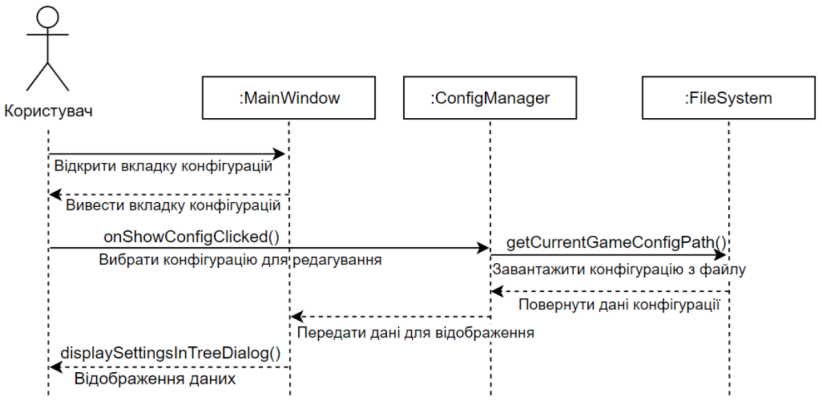
10

ДІАГРАМИ ПОСЛІДОВНОСТІ



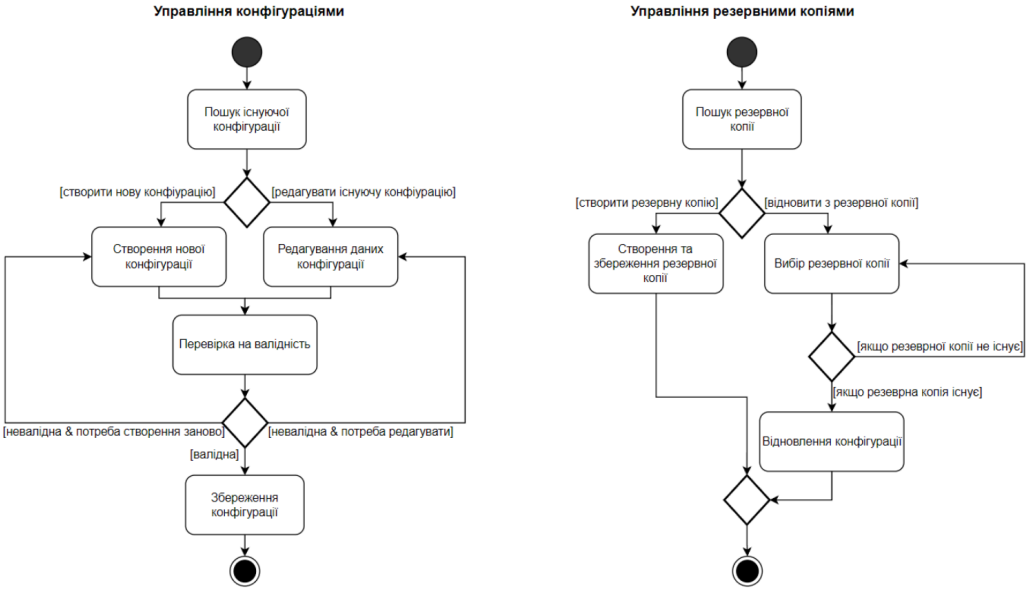
11

ДІАГРАМА ПОСЛІДОВНОСТІ ПЕРЕГЛЯДУ КОНФІГУРАЦІЇ



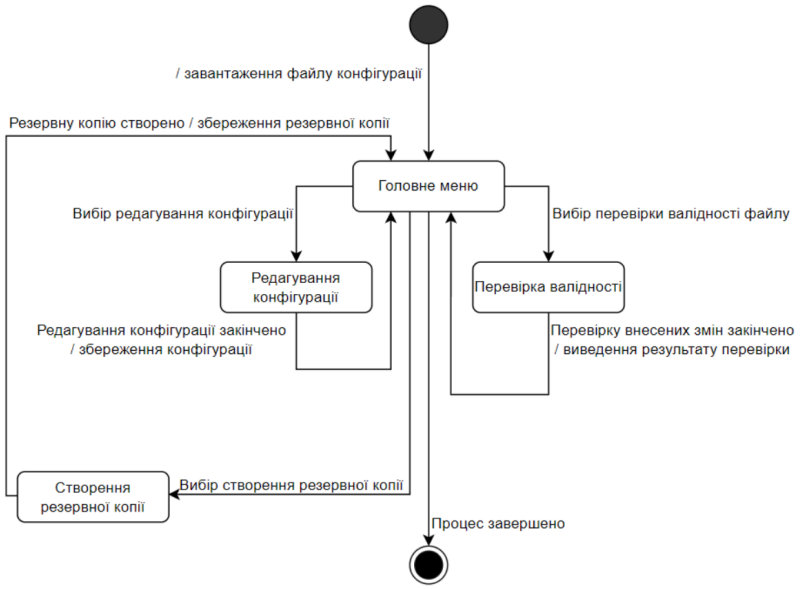
12

ДІАГРАМИ ДІЯЛЬНОСТІ



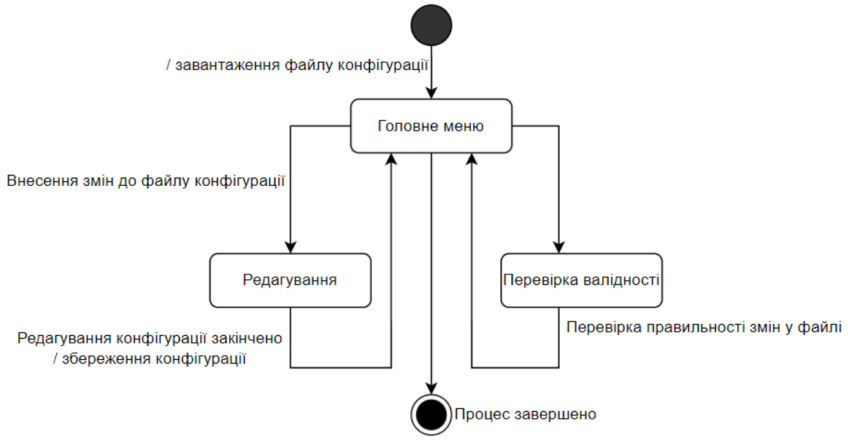
13

ДІАГРАМА СТАНІВ ЗАСТОСУНКУ



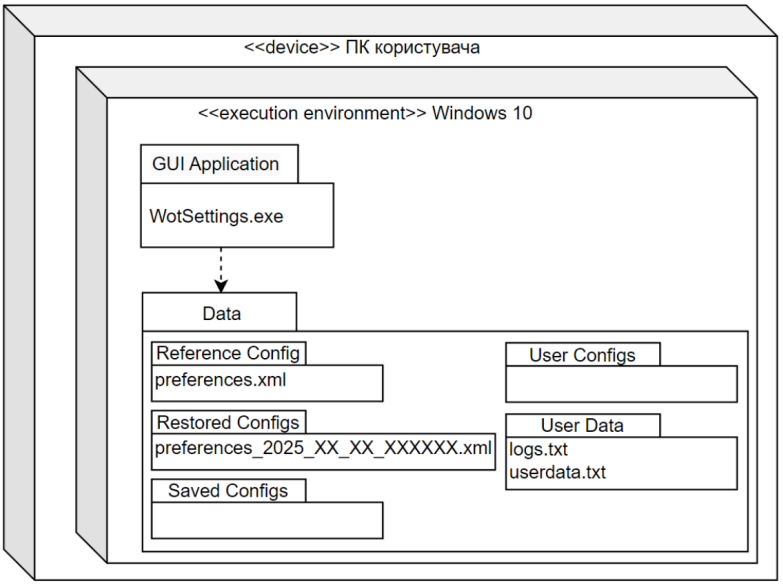
14

ДІАГРАМА СТАНІВ РОБОТИ З КОНФІГУРАЦІЙНИМ ФАЙЛОМ



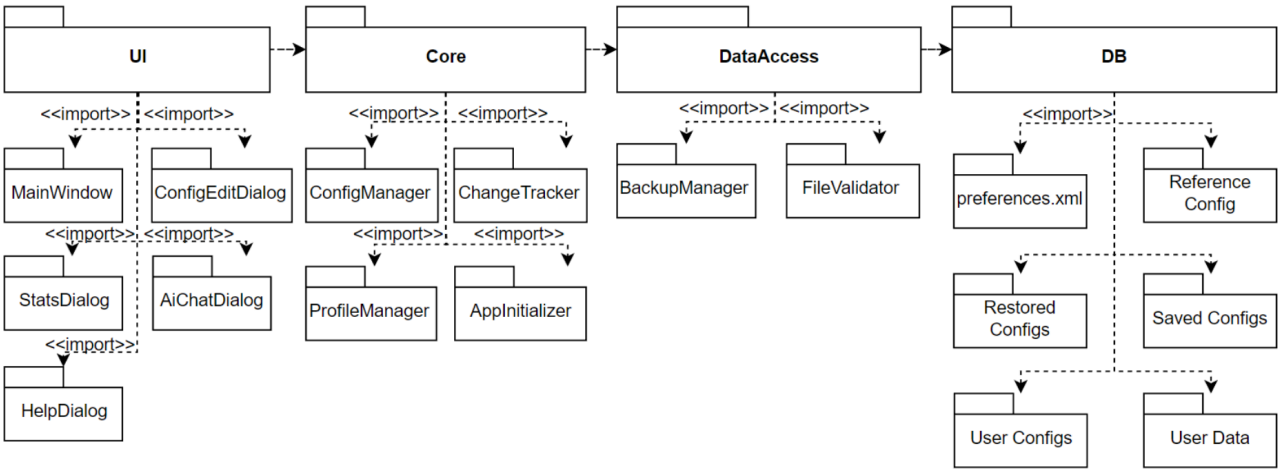
15

ДІАГРАМА РОЗГОРТАННЯ



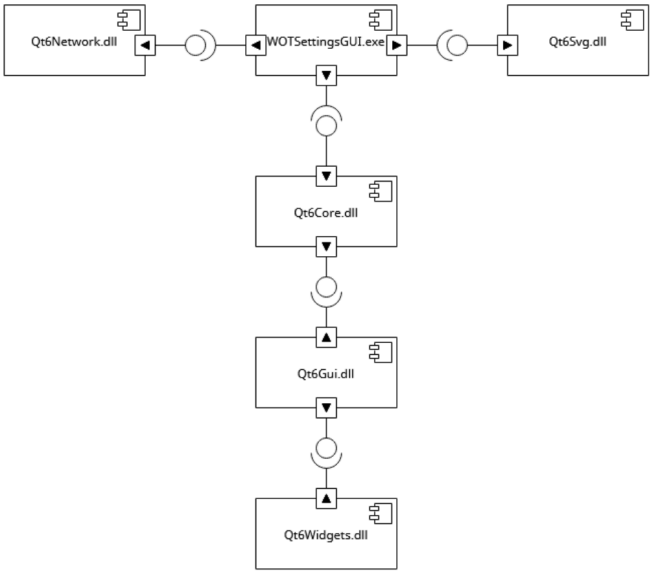
16

ЛОГІЧНА АРХІТЕКТУРА



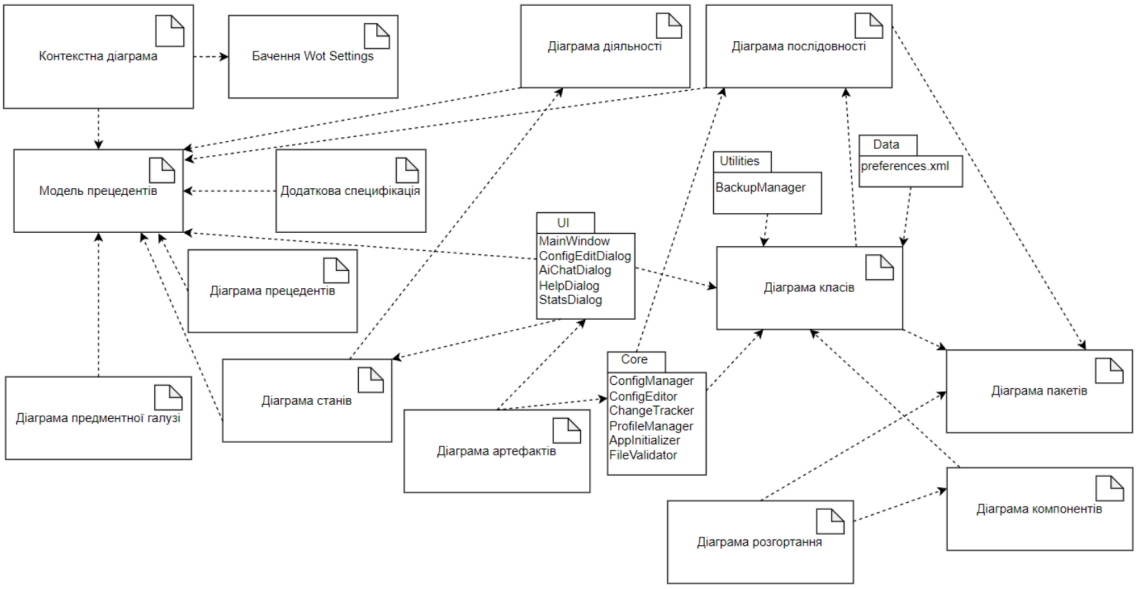
17

ДІАГРАМА КОМПОНЕНТІВ



18

ДІАГРАМА АРТЕФАКТІВ



МОДУЛЬНЕ ТЕСТУВАННЯ

```
21:05:39: Starting C:\Users\Kroll\Desktop\Диплом\TESTS\WOTConfiguratorGUITests\build\Desktop_Qt_6_8_2_MinGW_64_bit-Debug\all_tests.exe...
********* Start testing of TestAppInitializer *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestAppInitializer::initTestCase()
PASS   : TestAppInitializer::testCheckFolders()
PASS   : TestAppInitializer::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped, 0 blacklisted, 19ms
********* Finished testing of TestAppInitializer *********
********* Start testing of TestFileValidator *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestFileValidator::initTestCase()
PASS   : TestFileValidator::testValidateFile()
PASS   : TestFileValidator::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped, 0 blacklisted, 3ms
********* Finished testing of TestFileValidator *********
********* Start testing of TestBackupManager *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestBackupManager::initTestCase()
PASS   : TestBackupManager::cleanupTestCase()
Totals: 2 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms
********* Finished testing of TestBackupManager *********
********* Start testing of TestConfigManager *********
Config: Using QtTest library 6.8.2, Qt 6.8.2 (x86_64-little_endian-llp64 shared (dynamic) release build; by GCC 13.1.0), windows 10
PASS   : TestConfigManager::initTestCase()
PASS   : TestConfigManager::testGetCurrentGameConfigPath()
PASS   : TestConfigManager::testViewCurrentGameConfigContent()
PASS   : TestConfigManager::testUploadConfig()
PASS   : TestConfigManager::cleanupTestCase()
Totals: 5 passed, 0 failed, 0 skipped, 0 blacklisted, 1ms
********* Finished testing of TestConfigManager *********
21:05:39: C:\Users\Kroll\Desktop\Диплом\TESTS\WOTConfiguratorGUITests\build\Desktop_Qt_6_8_2_MinGW_64_bit-Debug\all_tests.exe exited with code 0
```

РЕЗУЛЬТАТИ КОРИСТУВАЦЬКОГО ТЕСТУВАННЯ

Критерій оцінювання	Середній бал (з 10)
Автоматичне виявлення шляху до гри World of Tanks	10.0
Зручність створення резервних копій конфігурацій	9.83
Зручність відновлення конфігурацій з резервних копій	9.83
Зручність перегляду налаштувань через графічний інтерфейс	9.0
Зручність редагування налаштувань через графічний інтерфейс (зрозумілість опцій, легкість зміни значень)	9.17
Ефективність валідації XML-файлу (запобігання помилкам, коректність перевірки)	10.0
Функціонал керування нік-неймами (додавання, редагування, видалення, відображення)	10.0
Зручність пошуку статистики гравців	9.92
Зручність та користь ШІ-помічника	9.92
Загальна стабільність роботи застосунку (відсутність збоїв, зависань)	9.67
Швидкодія застосунку (швидкість завантаження, збереження, виконання операцій)	10.0
Інтуїтивність та зрозумілість користувацького інтерфейсу	9.75
Привабливість візуального дизайну застосунку	9.0
Наскільки застосунок "WOT Configurator" спростив Вам управління налаштуваннями гри World of Tanks?	10.0
Ймовірність рекомендації "WOT Configurator" іншим гравцям World of Tanks	10.0

Які функції Ви хотіли б бачити у майбутніх версіях застосунку?

Готові конфігурації обладнання відомих/найкращих гравців або напопулярніші конфігурації відповідно до класу техніки

Повну Статистику(WN8, EFF, WTR і тд.). Можливо керування своим Аккаунтом та навіть кланом

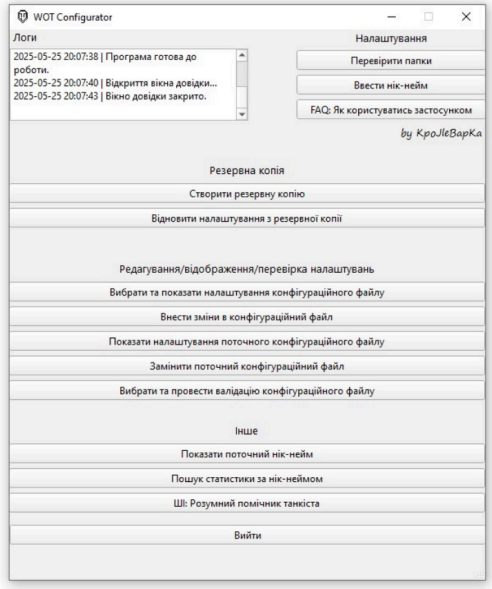
Інший дизайн більш зручний та комфортний. Можливість прямого скачування з офіційного сайту WOT. Додання модів, це було б супер чудовим модом, не дивлячись на те що цей застосунок призначений для налаштувань гри, я би хотів порекомендувати додати моди, по-перше це було б якоюсь інноваційною продукцією.

З якими проблемами або труднощами Ви зіткнулися під час використання "WOT Configurator"?

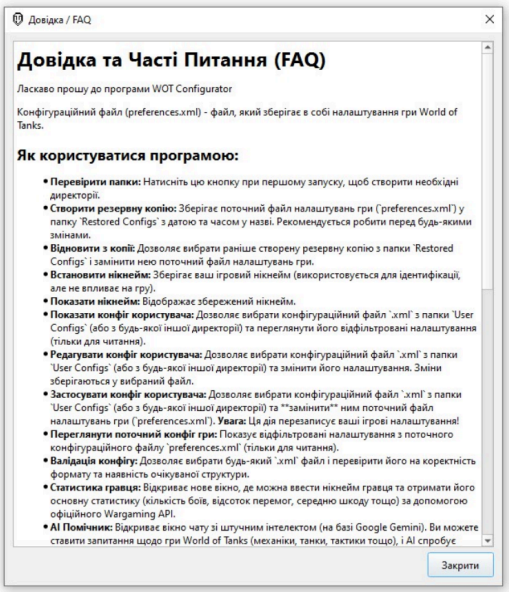
Загалом труднощів не було. Єдине на що натрапив при ранніх тестах - це неможливість використання клавіші "Enter" для вводу запиту до ШІ помічника.

Особливо важно було щось знайти при першому запуску, треба змінити інтерфейс та додати кольори, як до налаштувань так до самих іконок

ЕКРАННІ ФОРМИ

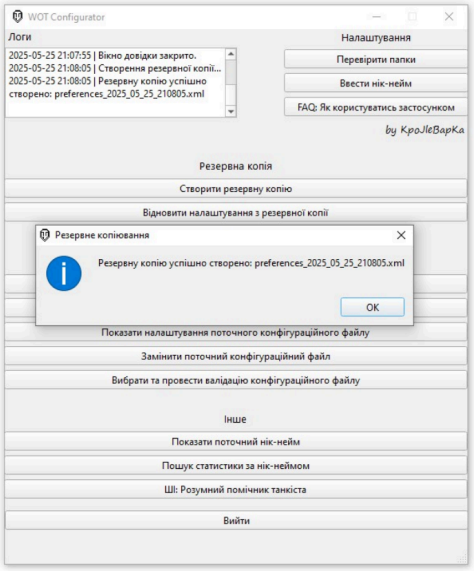


Головне меню

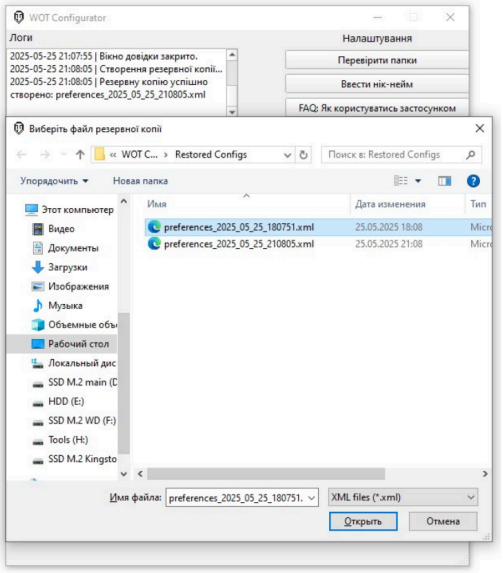


Вікно довідки / FAQ

ЕКРАННІ ФОРМИ



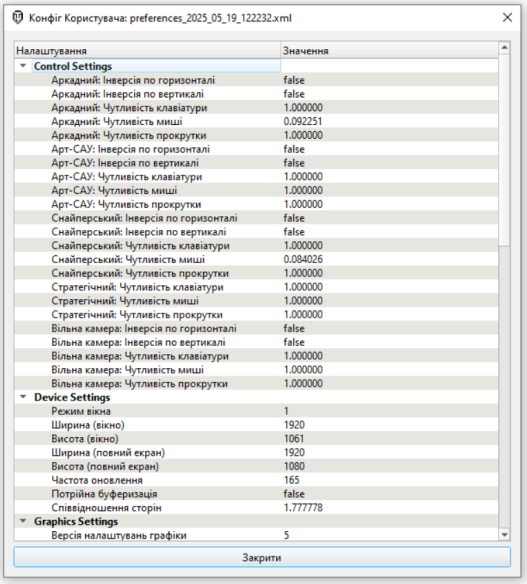
Головне меню, створення резервної копії



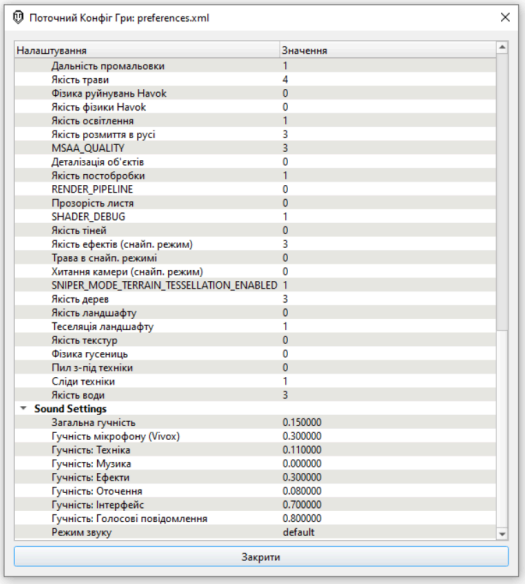
Головне меню, відновлення резервної копії

23

ЕКРАННІ ФОРМИ



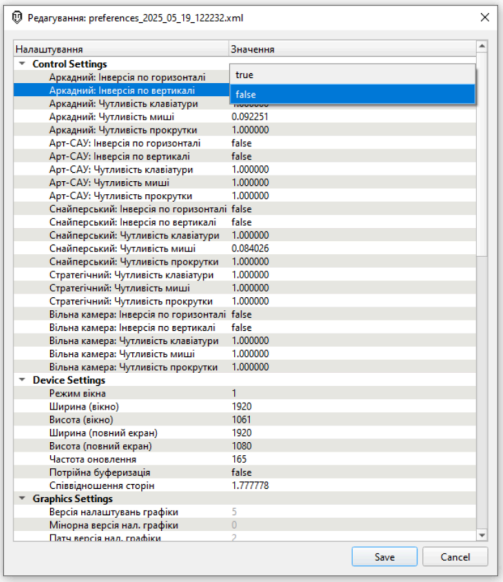
Вікно перегляду вибраної користувачем конфігурації



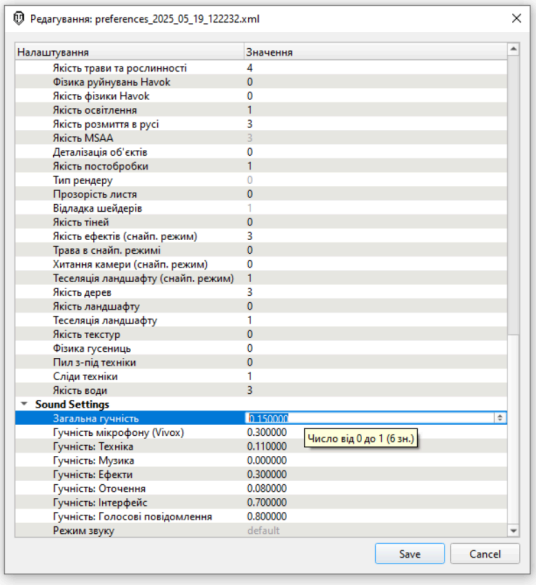
Вікно перегляду поточної (активної) конфігурації

24

ЕКРАННІ ФОРМИ



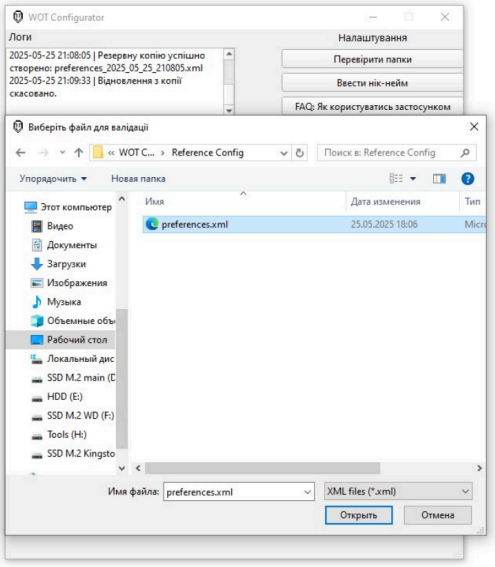
Вікно редагування вибраної користувачем конфігурації (булеві значення)



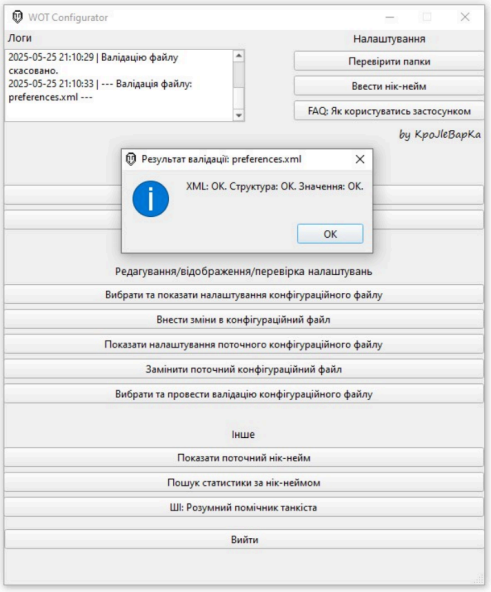
Вікно редагування вибраної користувачем конфігурації (числові значення)

25

ЕКРАННІ ФОРМИ



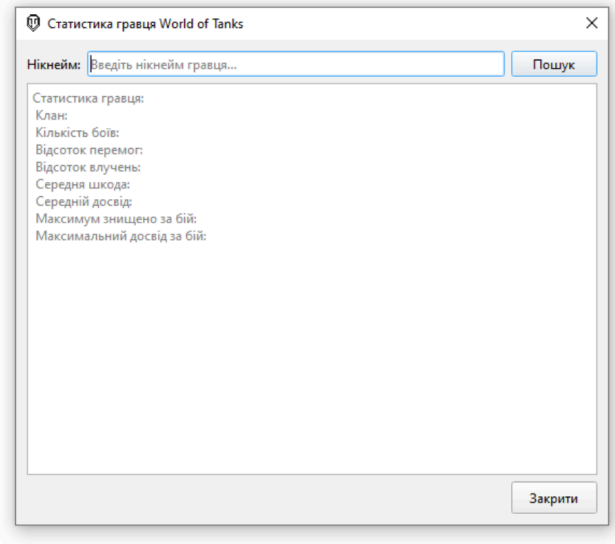
Вибір конфігураційного файлу для проведення валідації



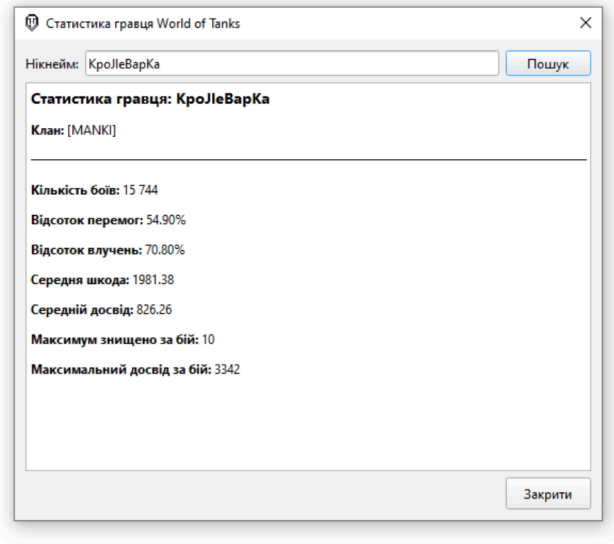
Вікно результату валідації

26

ЕКРАННІ ФОРМИ



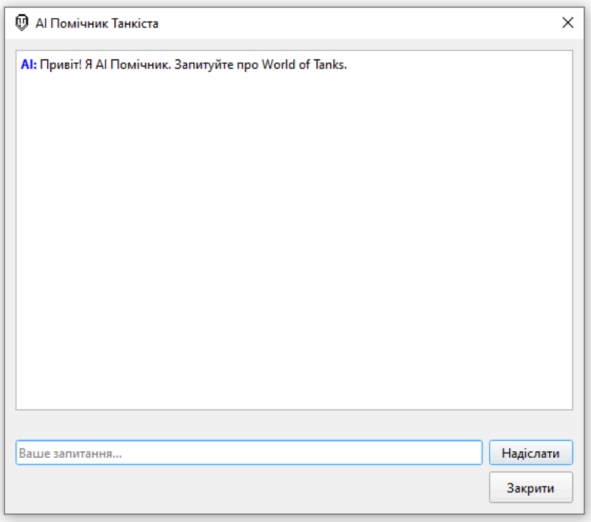
Вікно пошуку статистики гравців гри World of Tanks за нікнеймом



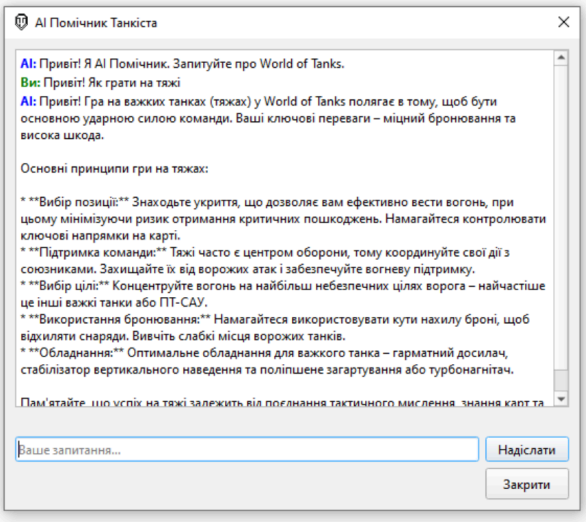
Вікно пошуку статистики гравців гри World of Tanks (результат пошуку)

27

ЕКРАННІ ФОРМИ



Вікно AI помічника



Вікно AI-помічника (результат запиту)

28

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Солов’ян А.О., Гаманюк І.М. ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ДАНИХ У ЗАСТОСУНКУ WOT CONFIGURATOR. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, С.327-328

2. Солов’ян А.О., Гаманюк І.М. ОБРОБКА КОНФІГУРАЦІЙНИХ ДАНИХ ДЛЯ ОПТИМІЗАЦІЇ НАЛАШТУВАНЬ ГРИ WORLD OF TANKS. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2025, ДУІКТ, м.Київ, С.563-565

3. Солов’ян А.О., Гаманюк І.М. РОЗРОБКА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ ДЛЯ ЕФЕКТИВНОГО УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ГРИ WORLD OF TANKS. Матеріали №21 студентського наукового журналу «UNIVERSUM», 20.06.2025, подано до друку

ВИСНОВКИ

1. Проаналізовано предметну область, досліджено існуючі рішення та проблеми керування конфігураціями гри World of Tanks.
2. На основі аналізу визначено функціональні та нефункціональні вимоги до десктопного застосунку.
3. Вибрано оптимальні технології та інструменти для реалізації поставлених вимог при створенні десктопного застосунку.
4. Реалізовано запланований функціонал та інтерфейс користувача для ефективного керування конфігураціями WoT.
5. Протестовано та підтверджено працездатність, стабільність та зручність використання розробленого застосунку.
6. В результаті створено програмний застосунок, який відповідає вимогам та вирішує завдання спрощення керування конфігураціями World of Tanks.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Код файлу BackupManager.cpp

```
fs::path BackupManager::getGameConfigPath() {
	const char* appDataPath_cstr =
getenv("APPDATA");
	if (!appDataPath_cstr) {
	throw std::runtime_error("Не вдалося отримати
системний шлях APPDATA.");
	}
	// Перевіряємо, чи шлях існує і чи це директорія
(хоча б базовий %APPDATA%)
	fs::path appDataDir(appDataPath_cstr);
	if (!fs::exists(appDataDir) ||
!fs::is_directory(appDataDir)) {
	throw std::runtime_error("Шлях APPDATA не існує
або не є директорією: " + std::string(appDataPath_cstr));
	}
	// Повертаємо повний шлях до файлу
	return appDataDir / "Wargaming.net" /
"WorldOfTanks" / "preferences.xml";
}

// Повертає шлях до створеного бекапу або кидає виняток
fs::path BackupManager::createBackup() {
	fs::path sourcePath = getGameConfigPath();

	if (!fs::exists(sourcePath)) {
	throw std::runtime_error("Файл конфігурації гри
для резервного копіювання не знайдено: " +
sourcePath.string());
	}

	if (!m_validator.validateBeforeAction(sourcePath,
"Створення резервної копії (перевірка джерела)", false)) {
	throw std::runtime_error("Перевірка
файлу-джерела перед резервним копіюванням не пройдена
або скасована.");
	}

	auto now = std::chrono::system_clock::now();
	auto in_time_t =
std::chrono::system_clock::to_time_t(now);
	std::tm local_tm;
#ifdef _WIN32
	localtime_s(&local_tm, &in_time_t);
#else
	localtime_r(&in_time_t, &local_tm); // POSIX
#endif
	std::stringstream ss;
	ss << "preferences_" << std::put_time(&local_tm,
"%Y_%m_%d_%H%M%S") << ".xml";
	fs::path backupDir = "Restored Configs";
	fs::path backupPath = backupDir / ss.str();

	try {
	if (!fs::exists(backupDir)) {
	fs::create_directories(backupDir);
	}
	fs::copy_file(sourcePath, backupPath,
fs::copy_options::overwrite_existing);
	return backupPath;
	} catch (const fs::filesystem_error& e) {
	throw std::runtime_error(std::string("Помилка
файлової системи при створенні резервної копії: ") +
e.what());
	} catch (...) {
	throw std::runtime_error("Невідома помилка під
час створення резервної копії.");
	}
}

// Приймає шлях до файлу бекапу, кидає виняток при
помилці
void BackupManager::restoreFromBackup(const fs::path&
backupPath) {
	if (!fs::exists(backupPath)) {
	throw std::runtime_error("Обраний файл резервної
копії не знайдено: " + backupPath.string());
	}

	fs::path targetPath = getGameConfigPath();
	fs::path targetDir = targetPath.parent_path();

	try {
	if (!fs::exists(targetDir)) {
	fs::create_directories(targetDir);
	}
	fs::copy_file(backupPath, targetPath,
fs::copy_options::overwrite_existing);
	} catch (const fs::filesystem_error& e) {
	throw std::runtime_error(std::string("Помилка
файлової системи при відновленні з копії: ") + e.what());
	} catch (...) {
	throw std::runtime_error("Невідома помилка під
час відновлення з копії.");
	}
}
```

Код файлу ConfigEditor.cpp

```
namespace {

std::string ltrim(std::string s) {
```

```
        s.erase(s.begin(), std::find_if_not(s.begin(), s.end(),
::isspace));
        return s;
}

std::string rtrim(std::string s) {
        s.erase(std::find_if_not(s.rbegin(), s.rend(),
::isspace).base(), s.end());
        return s;
}

std::string trim(std::string s) {
        return ltrim(rtrim(std::move(s)));
}

}

// Метод для читання всього вмісту файлу як рядка
std::string ConfigEditor::readConfigContent(const fs::path&
configPath) {
        if (!fs::exists(configPath)) {
        throw std::runtime_error("Файл конфігурації не
знайдено: " + configPath.string());
        }
        if (!fs::is_regular_file(configPath)){
        throw std::runtime_error("Шлях не є файлом: " +
configPath.string());
        }

        std::ifstream file(configPath);
        if (!file.is_open()) {
        throw std::runtime_error("Не вдалося відкрити
файл конфігурації: " + configPath.string());
        }

        std::stringstream buffer;
        try {
        buffer << file.rdbuf();
        } catch (const std::exception& e) {
        throw std::runtime_error(std::string("Помилка
читання файлу конфігурації: ") + e.what());
        }
        return buffer.str();
}

// Метод для отримання відфільтрованих налаштувань
FilteredSettingsMap ConfigEditor::getFilteredSettings(const
fs::path& configPath) {
        FilteredSettingsMap categorizedSettings;

        // Завантаження та перевірка XML
        pugi::xml_document doc;
        pugi::xml_parse_result result =
doc.load_file(configPath.wstring().c_str());
        if (!result) {
        throw std::runtime_error("Помилка завантаження
XML: " + std::string(result.description()) + " у файлі " +
configPath.string());
        }

        pugi::xml_node root = doc.child("root");
        if (!root) {
        throw std::runtime_error("Відсутній кореневий
елемент <root> у файлі: " + configPath.string());
        }

        // Визначення фільтрів
        const std::unordered_set<std::string>
soundSettingsFilter = {
        "masterVolume", "volume_micVivox",
"volume_vehicles", "volume_music",
        "volume_effects", "volume_ambient", "volume_gui",
"volume_voice", "soundMode",
        "bass_boost"
        };
        const std::unordered_set<std::string>
controlSettingsFilter = {
        "horzInvert", "vertInvert", "keySensitivity",
"sensitivity", "scrollSensitivity"
        };
        const std::unordered_set<std::string>
deviceSettingsFilter = {
        "windowMode", "windowedWidth",
"windowedHeight", "fullscreenWidth",
        "fullscreenHeight", "fullscreenRefresh",
"aspectRatio", "gamma", "tripleBuffering"
        };
        const std::vector<std::string> graphicDirectTagsFilter
= {
        "graphicsSettingsVersion",
"graphicsSettingsVersionMinor",
"graphicsSettingsVersionMaintainance",
        "graphicsSettingsStatus",
"ParticlSystemNoRenderGroup", "distributionLevel",
        "colorGradingStrength", "brightnessDeferred",
"contrastDeferred", "saturationDeferred"
        };

        pugi::xml_node scriptsPreferences =
root.child("scriptsPreferences");
        if (scriptsPreferences) {
        // Звук
        pugi::xml_node soundPrefs =
scriptsPreferences.child("soundPrefs");
        if (soundPrefs) {
        std::vector<std::pair<std::string, std::string>>
currentCategorySettings;
        for (const auto& setting : soundPrefs.children()) {
        std::string name = trim(setting.name());
        if (soundSettingsFilter.count(name)) {
                currentCategorySettings.push_back({name,
trim(setting.text().as_string())});
        }
        }
        if (!currentCategorySettings.empty())
categorizedSettings["Sound Settings"] =
currentCategorySettings;
        }
        // Керування
        pugi::xml_node controlMode =
scriptsPreferences.child("controlMode");
        if (controlMode) {
        std::vector<std::pair<std::string, std::string>>
currentCategorySettings;
        for (const auto& mode : controlMode.children()) {
        std::string modeName = trim(mode.name());
        pugi::xml_node camera = mode.child("camera");
        if (camera) {
                for (const auto& setting :
camera.children()) {
                std::string settingNamePart =
trim(setting.name());
                if
(controlSettingsFilter.count(settingNamePart)) {
                std::string fullSettingName = modeName +
"/" + settingNamePart; // Формуємо повний ключ

currentCategorySettings.push_back({fullSettingName,
trim(setting.text().as_string())});
```

```
                    }
                    }
            }
            }
            if (!currentCategorySettings.empty())
categorizedSettings["Control Settings"] =
currentCategorySettings;
            }
            }

            // Графіка
            pugi::xml_node graphicsPreferences =
root.child("graphicsPreferences");
            if (graphicsPreferences) {
            std::vector<std::pair<std::string, std::string>>
currentCategorySettings;
            // Обробка прямих тегів
            for(const std::string& tagName :
graphicDirectTagsFilter) {
            pugi::xml_node node =
graphicsPreferences.child(tagName.c_str());
            if(node) {
            currentCategorySettings.push_back({tagName,
trim(node.text().as_string())});
            }
            }
            // Обробка тегів <entry>
            for (const auto& entry :
graphicsPreferences.children("entry")) {
            std::string label = entry.child_value("label");
            std::string trimmed_label = trim(label);
            if (!trimmed_label.empty()) {
            std::string activeOption =
entry.child_value("activeOption");
            currentCategorySettings.push_back({trimmed_label,
trim(activeOption)});
            }
            }
            if (!currentCategorySettings.empty())
categorizedSettings["Graphics Settings"] =
currentCategorySettings;
            }

            // Пристрій
            pugi::xml_node devicePreferences =
root.child("devicePreferences");
            if (devicePreferences) {
            std::vector<std::pair<std::string, std::string>>
currentCategorySettings;
            for (const auto& setting :
devicePreferences.children()) {
            std::string name = trim(setting.name());
            if (deviceSettingsFilter.count(name)) {
            currentCategorySettings.push_back({name,
trim(setting.text().as_string())});
            }
            }
            if (!currentCategorySettings.empty())
categorizedSettings["Device Settings"] =
currentCategorySettings;
            }

            return categorizedSettings;
}

// Метод для збереження змін в XML
void ConfigEditor::saveFilteredSettings(const fs::path&
configPath, const FilteredSettingsMap& settings) {
            pugi::xml_document doc;
            pugi::xml_parse_result loadResult =
doc.load_file(configPath.wstring().c_str());

            if (!loadResult) {
            throw std::runtime_error("Не вдалося завантажити
файл для збереження: " + configPath.string() + " (" +
loadResult.description() + ")");
            }

            pugi::xml_node root = doc.child("root");
            if (!root) {
            throw std::runtime_error("Не знайдено <root>
елемент у файлі: " + configPath.string());
            }

            for (const auto& categoryPair : settings) {
            const std::string& categoryName = categoryPair.first;
            const auto& settingsInCategory =
categoryPair.second;

            pugi::xml_node categoryNode;
            pugi::xml_node scriptsPrefsNode;
            pugi::xml_node graphicsPrefsNode;

            if (categoryName == "Sound Settings") {
            scriptsPrefsNode = root.child("scriptsPreferences");
            if (scriptsPrefsNode) categoryNode =
scriptsPrefsNode.child("soundPrefs");
            } else if (categoryName == "Graphics Settings") {
            graphicsPrefsNode =
root.child("graphicsPreferences");
            categoryNode = graphicsPrefsNode;
            } else if (categoryName == "Control Settings") {
            scriptsPrefsNode = root.child("scriptsPreferences");
            if (scriptsPrefsNode) categoryNode =
scriptsPrefsNode.child("controlMode");
            } else if (categoryName == "Device Settings") {
            categoryNode = root.child("devicePreferences");
            }

            if (!categoryNode) {
            std::cerr << "Warning: Category node not found in
XML for category during save: " << categoryName <<
std::endl;
            continue;
            }

            for (const auto& settingPair : settingsInCategory) {
            const std::string& settingName = settingPair.first;
            const std::string& newValue = settingPair.second;

            pugi::xml_node settingNode;

            if (categoryName == "Control Settings" &&
settingName.find('/') != std::string::npos) {
            size_t slashPos = settingName.find('/');
            std::string modeName = settingName.substr(0,
slashPos);
            std::string actualSettingName =
settingName.substr(slashPos + 1);
            pugi::xml_node modeNode =
categoryNode.child(modeName.c_str());
            if(modeNode) {
                        pugi::xml_node cameraNode =
modeNode.child("camera");
                        if(cameraNode) {
                        settingNode =
cameraNode.child(actualSettingName.c_str());
                        } else {
```

```
                        std::cerr << "Warning: <camera> node not
found for mode '" << modeName << "' during save." <<
std::endl; continue;
                        }
                } else {
                        std::cerr << "Warning: Control mode node
not found: '" << modeName << "' during save." << std::endl;
continue;
                }
                }
                else if (categoryName == "Graphics Settings") {
                settingNode =
categoryNode.child(settingName.c_str());
                if (!settingNode) {
                        bool foundEntry = false;
                        for (pugi::xml_node entry :
categoryNode.children("entry")) {
                        if (trim(entry.child_value("label")) ==
settingName) {
                        settingNode = entry.child("activeOption");
                        foundEntry = true;
                        break;
                        }
                        }
                        if (!foundEntry) {
                        std::cerr << "Warning: Graphics node
(direct or entry/label) not found for key: " << settingName << "
during save." << std::endl;
                                continue;
                                }
                        }
                        }
                        // Обробка для інших категорій (Sound, Device)
                        else {
                        settingNode =
categoryNode.child(settingName.c_str());
                        }

                        if (settingNode) {
                        if (!settingNode.text().set(newValue.c_str())) {
                                std::cerr << "Warning: Failed to set text for
node: " << settingName << std::endl;
                        }
                        } else {
                        std::cerr << "Warning: Setting node could not be
found in XML for key: '" << settingName << "' in category '"
<< categoryName << "' during save." << std::endl;
                        }
                        }
                        }

                        if (!doc.save_file(configPath.wstring().c_str())) {
                        throw std::runtime_error("Не вдалося зберегти
зміни у файл: " + configPath.string());
                        }
}
```

Код файлу FileValidator.cpp

```
namespace {
bool tryParseFloat(const char* text, float& outValue) {
        if (!text) return false;
        std::string s(text);
        std::istringstream iss(s);
        iss >> outValue;
        char remaining;
        return !iss.fail() && !(iss >> remaining);
}
bool tryParseInt(const char* text, int& outValue) {
        if (!text) return false;
        std::string s(text);
        std::istringstream iss(s);
        iss >> outValue;
        char remaining;
        return !iss.fail() && !(iss >> remaining);
}
}


// Реалізація основного методу валідації
ValidationResult FileValidator::validateFile(const fs::path&
filePath) {
        ValidationResult result;
        pugi::xml_document doc;

        if (!fs::exists(filePath)) {
        result.wellFormedError = "Файл не знайдено: " +
filePath.string();
        return result; // isWellFormed = false
        }
        if (!fs::is_regular_file(filePath)) {
        result.wellFormedError = "Шлях не є файлом: " +
filePath.string();
        return result; // isWellFormed = false
        }

        // Перевірка Well-Formed
        std::string wfError;
        result.isWellFormed =
isXmlWellFormedInternal(filePath, doc, wfError);
        if (!result.isWellFormed) {
        result.wellFormedError = wfError;
        return result;
        }

        // Перевірка структури (тільки якщо XML
коректний)
        std::string structWarn;
        result.hasStructure =
hasExpectedStructureInternal(doc, structWarn);
        result.structureInfo = structWarn; // Записуємо
повідомлення (може бути "OK" або попередження)

        // Перевірка значень (тільки якщо XML
коректний)
        result.valueErrors =
findInvalidSimpleValuesInternal(doc);

        return result;
}

// Реалізація допоміжного методу для GUI
bool FileValidator::validateBeforeAction(const fs::path&
filePath, const std::string& actionNameStd, bool showSuccess)
{
        QString actionName =
QString::fromStdString(actionNameStd);
        ValidationResult result = validateFile(filePath);

        if (result.isValid()) {
        QString summary;
```

```
        bool hasWarnings = false;

        summary += "XML: OK.\n";
        summary += QString("Структура:
%1\n").arg(QString::fromStdString(result.structureInfo));
        if (!result.hasStructure) hasWarnings = true;

        if (result.valueErrors.empty()) {
        summary += "Значення: OK.";
        } else {
        summary += QString("Значення: Знайдено %1
потенційних проблем.").arg(result.valueErrors.size());
        hasWarnings = true;
        summary += "\nПриклади помилок значень:\n";
        int count = 0;
        for(const auto& err : result.valueErrors) {
        if (++count > 3) {
                summary += "- ...\n";
                break;
        }
        summary += QString("-
%1\n").arg(QString::fromStdString(err));
        }
        }

        if (hasWarnings) {
        QMessageBox::StandardButton reply;
        QString warningText = QString("Увага! Файл '%1'
містить попередження:\n\n%2\n\nПродовжити дію '%3'?")

.arg(QString::fromStdWString(filePath.filename().wstring()))
                        .arg(summary)
                        .arg(actionName);
        reply = QMessageBox::warning(nullptr, actionName
+ " - Попередження валідації",
                        warningText,
                        QMessageBox::Yes |
QMessageBox::No,
                        QMessageBox::No);
        return (reply == QMessageBox::Yes);
        } else {
        if (showSuccess) {
        QMessageBox::information(nullptr, actionName + "
- Валідація успішна",
                        QString("Файл '%1' успішно
пройшов валідацію.")

.arg(QString::fromStdWString(filePath.filename().wstring())));
        }
        return true;
        }
        } else { // Критична помилка XML
        QMessageBox::critical(nullptr, actionName + " -
Помилка валідації",
                QString("Не вдалося виконати дію
'%1'.\nФайл '%2' не пройшов валідацію:\n%3")
                .arg(actionName)

.arg(QString::fromStdWString(filePath.filename().wstring()))

.arg(QString::fromStdString(result.wellFormedError)));
        return false;
        }
}


bool FileValidator::isXmlWellFormedInternal(const fs::path&
filePath, pugi::xml_document& doc, std::string& errorMsg) {
        pugi::xml_parse_result result =
doc.load_file(filePath.c_str());
        if (result.status != pugi::status_ok) {
        // Формуємо повідомлення про помилку
        std::stringstream ss;
        ss << "Помилка XML: " << result.description() << "
(позиція " << result.offset << ")";
        errorMsg = ss.str();
        return false;
        }
        errorMsg = "OK";
        return true;
}

bool FileValidator::hasExpectedStructureInternal(const
pugi::xml_document& doc, std::string& warnings) {
        bool structureOk = true;
        std::stringstream warningStream;

        const pugi::xml_node root = doc.child("root");
        if (!root) {
        warnings = "Відсутній кореневий елемент
<root>.";
        return false; // Це критична помилка структури
        }

        if (!root.child("scriptsPreferences")) {
        warningStream << "Відсутня секція
<scriptsPreferences>. ";
        structureOk = false;
        }
        if (!root.child("graphicsPreferences")) {
        warningStream << "Відсутня секція
<graphicsPreferences>. ";
        structureOk = false;
        }
        if (!root.child("devicePreferences")) {
        warningStream << "Відсутня секція
<devicePreferences>. ";
        structureOk = false;
        }

        const pugi::xml_node scriptsPrefs =
root.child("scriptsPreferences");
        if (scriptsPrefs && !scriptsPrefs.child("soundPrefs"))
{
        warningStream << "Відсутня підсекція
<soundPrefs> всередині <scriptsPreferences>. ";
        }
        if (scriptsPrefs &&
!scriptsPrefs.child("controlMode")) {
        warningStream << "Відсутня підсекція
<controlMode> всередині <scriptsPreferences>. ";
        }


        warnings = warningStream.str();
        if (structureOk && warnings.empty()) {
        warnings = "OK";
        }
        return structureOk;
}

std::vector<std::string>
FileValidator::findInvalidSimpleValuesInternal(const
pugi::xml_document& doc) {
        std::vector<std::string> errors;
        const pugi::xml_node root = doc.child("root");
        if (!root) return {"Помилка: Відсутній <root>
елемент для перевірки значень."};

        // Перевірка <soundPrefs>
```

```
		const pugi::xml_node scriptsPrefs =
root.child("scriptsPreferences");
		if (scriptsPrefs) {
		const pugi::xml_node soundPrefs =
scriptsPrefs.child("soundPrefs");
		if (soundPrefs) {
		const std::vector<std::string> volumeTags = { //
список тегів гучності
		"masterVolume", "volume_micVivox",
"volume_vehicles", "volume_music",
		"volume_effects", "volume_ambient", "volume_gui",
"volume_voice",
		"volume_masterFadeVivox",
"volume_masterVivox", "volume_music_hangar",
		"volume_ev_ambient", "volume_ev_effects",
"volume_ev_gui",
		"volume_ev_music", "volume_ev_vehicles",
"volume_ev_voice"
		};
		for (const std::string& tagName : volumeTags) {
		pugi::xml_node node =
soundPrefs.child(tagName.c_str());
		if (node) {
			float value;
			const char* text = node.text().as_string();
			if (tryParseFloat(text, value)) {
			if (value < 0.0f || value > 1.0f) {
			errors.push_back("<" + tagName + ">: '" +
text + "'. Очікується [0.0, 1.0].");
			}
			} else {
			errors.push_back("<" + tagName + ">: '" +
text + "'. Очікується число.");
			}
		}
		}
		}

		// Перевірка <fov>
		const pugi::xml_node fovNode =
scriptsPrefs.child("fov");
		if(fovNode) {
		float value;
		const char* text = fovNode.text().as_string();
		if(tryParseFloat(text, value)) {
		if (value < 50.0f || value > 130.0f) { // Допустимий
діапазон FOV
			errors.push_back("<fov>: '" +
std::string(text) + "'. Нетипове значення (очікується
50-130).");
		}
		} else {
		errors.push_back("<fov>: '" + std::string(text) + "'.
Очікується число.");
		}
		}

		}

		// Перевірка <devicePreferences>
		const pugi::xml_node devicePrefs =
root.child("devicePreferences");
		if (devicePrefs) {
		// Window Mode
		pugi::xml_node wmNode =
devicePrefs.child("windowMode");
		if (wmNode) {
		int value;
		const char* text = wmNode.text().as_string();
		if(tryParseInt(text, value)) {
		if (value < 0 || value > 2) { // 0 - вікно, 1 -
повноекранне вікно, 2 - повний екран
			errors.push_back("<windowMode>: '" +
std::string(text) + "'. Очікується 0, 1, або 2.");
		}
		} else {
		errors.push_back("<windowMode>: '" +
std::string(text) + "'. Очікується ціле число.");
		}
		}
		// Resolutions
		const std::vector<std::pair<std::string, int>>
resolutionTags = {
		{"windowedWidth", 640}, {"windowedHeight",
480},
		{"fullscreenWidth", 800}, {"fullscreenHeight", 600}
		};
		for (const auto& pair : resolutionTags) {
		pugi::xml_node resNode =
devicePrefs.child(pair.first.c_str());
		if(resNode) {
		int value;
		const char* text = resNode.text().as_string();
		if(tryParseInt(text, value)) {
			if (value < pair.second) {
			errors.push_back("<" + pair.first + ">: '" +
text + "'. Очікується >= " + std::to_string(pair.second) + ".");
			}
		} else {
			errors.push_back("<" + pair.first + ">: '" +
text + "'. Очікується ціле число.");
		}
		}
		}
		// Refresh Rate
		pugi::xml_node rrNode =
devicePrefs.child("fullscreenRefresh");
		if(rrNode) {
		int value;
		const char* text = rrNode.text().as_string();
		if(tryParseInt(text, value)) {
		if (value < 10 || value > 400) {
			errors.push_back("<fullscreenRefresh>: '"
+ std::string(text) + "'. Нетипове значення (очікується
50-240).");
		}
		} else {
		errors.push_back("<fullscreenRefresh>: '" +
std::string(text) + "'. Очікується ціле число.");
		}
		}
		}

		// Перевірка графіки
		const pugi::xml_node graphPrefs =
root.child("graphicsPreferences");
		if (graphPrefs) {
		const std::vector<std::string> graphTags = {
		"colorGradingStrength", "brightnessDeferred",
"contrastDeferred", "saturationDeferred"
		};
		for (const std::string& tagName : graphTags) {
		pugi::xml_node node =
graphPrefs.child(tagName.c_str());
		if (node) {
		float value;
		const char* text = node.text().as_string();
		if (tryParseFloat(text, value)) {
			if (value < 0.0f || value > 1.5f) {
```

```
                errors.push_back("<" + tagName + ">: '" +
text + "'. Нетипове значення (очікується ~[0.0, 1.0]).");
                }
        } else {
                errors.push_back("<" + tagName + ">: '" +
text + "'. Очікується число.");
        }
        }
        }
        }

        return errors;
}
```

Код файлу ProfileManager.cpp

```
namespace fs = std::filesystem;

const fs::path userDataDir = "User Data";
const fs::path userdataFile = userDataDir / "userdata.txt";

// Приймає ім'я, зберігає у файл, кидає виняток при
помилці
void ProfileManager::setName(const std::string& name) {
        saveNameToFile(name);
}

void ProfileManager::saveNameToFile(const std::string&
name) {
        try {
        if (!fs::exists(userDataDir)) {
        fs::create_directories(userDataDir);
        }
        std::ofstream outFile(userdataFile);
        if (outFile.is_open()) {
        outFile << name; // Записуємо ім'я
        outFile.close();
        } else {
        throw std::runtime_error("Не вдалося відкрити
файл " + userdataFile.string() + " для запису.");
        }
        } catch (const fs::filesystem_error& e) {
        throw std::runtime_error(std::string("Помилка
файлової системи при збереженні імені: ") + e.what());
        } catch (const std::runtime_error& e) {
        throw;
        } catch (...) {
        throw std::runtime_error("Невідома помилка під
час збереження імені.");
        }
}

std::string ProfileManager::loadNameFromFile() {
        if (!fs::exists(userdataFile)) {
        return "";
        }
        std::ifstream inFile(userdataFile);
        std::string loadedName = "";
        if (inFile.is_open()) {
        std::getline(inFile, loadedName);
        inFile.close();
        } else {
        throw std::runtime_error("Не вдалося відкрити
файл " + userdataFile.string() + " для читання.");
        }
        return loadedName;
}
```

Код файлу ConfigManager.cpp

```
fs::path ConfigManager::getGameConfigPathInternal() {
        const char* appDataPath_cstr =
getenv("APPDATA");
        if (!appDataPath_cstr) {
        throw std::runtime_error("Не вдалося отримати
системний шлях APPDATA.");
        }
        fs::path appDataDir(appDataPath_cstr);
        if (!fs::exists(appDataDir) ||
!fs::is_directory(appDataDir)) {
        throw std::runtime_error("Шлях APPDATA не існує
або не є директорією: " + std::string(appDataPath_cstr));
        }
        return appDataDir / "Wargaming.net" /
"WorldOfTanks" / "preferences.xml";
}

// Повертає шлях до поточного конфігу гри
fs::path ConfigManager::getCurrentGameConfigPath() {
        return getGameConfigPathInternal();
}

// Приймає шлях до конфігу користувача, кидає виняток
при помилці
void ConfigManager::changeCurrentConfig(const fs::path&
sourceConfigPath) {
        if (!fs::exists(sourceConfigPath)) {
        throw std::runtime_error("Обраний файл
конфігурації користувача не знайдено: " +
sourceConfigPath.string());
        }
        if (!fs::is_regular_file(sourceConfigPath)) {
        throw std::runtime_error("Вибраний шлях не є
файлом: " + sourceConfigPath.string());
        }

        fs::path targetPath = getGameConfigPathInternal();
        fs::path targetDir = targetPath.parent_path();

        // Застосування (копіювання з заміною)
        try {
        if (!fs::exists(targetDir)) {
        fs::create_directories(targetDir);
        }
        fs::copy_file(sourceConfigPath, targetPath,
fs::copy_options::overwrite_existing);
        } catch (const fs::filesystem_error& e) {
        throw std::runtime_error(std::string("Помилка
файлової системи при застосуванні конфігу: ") + e.what());
```

```
        } catch (...) {
        throw std::runtime_error("Невідома помилка під
час застосування конфігу.");
        }
}

// Читає та повертає вміст поточного конфігу гри
std::string ConfigManager::viewCurrentGameConfigContent()
{
        fs::path gameConfigPath =
getGameConfigPathInternal();
        if (!fs::exists(gameConfigPath)) {
        throw std::runtime_error("Файл конфігурації гри
(preferences.xml) не знайдено.");
        }

        // Валідація перед читанням
        ValidationResult valResult =
m_validator.validateFile(gameConfigPath);
        if (!valResult.isValid()) {
        throw std::runtime_error("Поточний конфіг гри не є
коректним XML: " + valResult.wellFormedError);
        }

        std::ifstream file(gameConfigPath);
        if (!file.is_open()) {
        throw std::runtime_error("Не вдалося відкрити
файл конфігурації гри: " + gameConfigPath.string());
        }
        std::stringstream buffer;
        try {
        buffer << file.rdbuf();
        } catch (const std::exception& e) {
        throw std::runtime_error(std::string("Помилка
читання файлу конфігурації гри: ") + e.what());
        }
        return buffer.str();
}
```

Код файлу ChangeTracker.cpp

```
namespace fs = std::filesystem;

// Метод для запису дії в лог
void ChangeTracker::logAction(const std::string&
functionName, bool success, const std::string& details) {
        const fs::path logDir = "User Data";
        const fs::path logFilePath = logDir / "logs.txt";

        try {
        if (!fs::exists(logDir)) {
        try {
        fs::create_directory(logDir);
        } catch(const fs::filesystem_error& e) {
        std::cerr << "CRITICAL ERROR: Could not create
log directory '" << logDir.string() << "'. Error: " << e.what() <<
std::endl;
        return;
        }
        }

        std::ofstream logFile(logFilePath, std::ios::app);

        if (!logFile.is_open()) {
        std::cerr << "ERROR: Could not open log file for
writing: " << logFilePath.string() << std::endl;
        return;
        }

        auto now = std::chrono::system_clock::now();
        auto now_c =
std::chrono::system_clock::to_time_t(now);
        std::tm now_tm;
#ifdef _WIN32
        localtime_s(&now_tm, &now_c);
#else
        localtime_r(&now_c, &now_tm);
#endif

        logFile << std::put_time(&now_tm,
"%Y-%m-%d_%H:%M:%S") << " | "
        << functionName << " | Result: [" << (success ?
"OK" : "NOK") << "]";

        if (!details.empty()) {
        logFile << " | Details: " << details;
        }

        logFile << std::endl;

        } catch (const fs::filesystem_error& e) {
        std::cerr << "Filesystem error during logging: " <<
e.what() << std::endl;
        } catch (const std::exception& e) {
        std::cerr << "Standard exception during logging: " <<
e.what() << std::endl;
        } catch (...) {
        std::cerr << "Unknown error during logging." <<
std::endl;
        }
}
```

Код файлу AppInitializer.cpp

```
void AppInitializer::checkFolders() {
        // Список папок, які мають існувати
        const std::vector<fs::path> folders = {
        "User Data", "Saved Configs", "Restored Configs",
"User Configs", "Reference Config"
        };

        try {
        for (const auto& folder : folders) {
        if (!fs::exists(folder)) {
        fs::create_directory(folder);
        }
```

```
        }

        // Створення reference config, якщо його немає
        fs::path refFolder = "Reference Config";
        fs::path refFile = refFolder / "preferences.xml";
        if (!fs::exists(refFile)) {
        if (!fs::exists(refFolder)) { // Переконуємось, що
папка є
        fs::create_directories(refFolder);
        }
        std::ofstream outFile(refFile, std::ios::binary);
        if (outFile.is_open()) {
        outFile.write(reinterpret_cast<const
char*>(preferences_xml), preferences_xml_len);
        outFile.close();
        } else {
        throw std::runtime_error("Не вдалося створити
еталонний файл конфігурації: " + refFile.string());
        }
        }
        } catch (const fs::filesystem_error& e) {
        throw std::runtime_error(std::string("Помилка
файлової системи при перевірці/створенні папок: ") +
e.what());
        } catch (const std::runtime_error& e) {
        throw;
        }
        catch (...) {
        throw std::runtime_error("Невідома помилка під
час перевірки/створення папок.");
        }
}
```