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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмних засобів для відстеження і редагування характеристик NPC на прикладі гри Skyrim»

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

_____ Михайло ІВАНОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Михайло ІВАНОВ

Керівник: Оксана ЗОЛОТУХІНА
к.т.н., доцент

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

Іванову Михайлу Євгеновичу

1. Тема кваліфікаційної роботи: «Розробка програмних засобів для відстеження і редагування характеристик NPC на прикладі гри Skyrim»
керівник кваліфікаційної роботи к.т.н., доц. Оксана ЗОЛОТУХІНА,
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: науково-технічна література, документація інструментів модифікації гри Skyrim Special Edition (Creation Kit, xEdit), документація бібліотек WPF, MVVM, YAML.NET, документація компонентів Syncfusion для WPF.

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

1. Огляд та аналіз структури ігрових даних Skyrim SE.
2. Аналіз існуючих засобів для обробки ігрових даних (Creation Kit, xEdit) та постановка задачі автоматизації аналізу.
3. Проектування програмного забезпечення для аналізу Actor Values персонажів гри Skyrim SE з використанням патерну MVVM.
4. Програмна реалізація WPF-застосунку з використанням бібліотек WPF, YAML.NET та компонентів Syncfusion.
5. Тестування застосунку та аналіз його функціональності на прикладах реальних ігрових даних.
6. Висновки щодо ефективності інструмента та можливості його подальшого вдосконалення.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Архітектура програмного забезпечення
6. Система побудови вузлів
7. Ієрархія вузлів
8. Діаграма послідовності взаємодії користувача із застосунком
9. Екранні форми
10. 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 структури ігрових даних Skyrim SE	05.03-13.03.2025	
3	Визначення вимог до програмного забезпечення на основі огляду існючих інструментів	14.03-20.03.2025	
4	Проєктування застосунку для перегляду і аналізу Actor Values	21.03-01.04.2025	
5	Програмна реалізація застосунку	02.04-19.04.2025	
6	Тестування застосунку	20.04-28.04.2025	
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 _____ Михайло ІВАНОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 53 стор., 2 табл., 24 рис., 22 джерела.

Мета роботи – спрощення процесів відстеження і редагування характеристик NPC у відеогрі Skyrim.

Об'єкт дослідження – процес відстеження і редагування характеристик NPC у рольових іграх.

Предмет дослідження – засоби та технології для обробки ігрових даних у рольових іграх, зокрема з використанням мови програмування C#, платформи WPF та бібліотеки Mutagen.Bethesda.

Короткий зміст роботи: у роботі проаналізовано особливості структури ігрових даних відеогри The Elder Scrolls V: Skyrim Special Edition, зокрема характеристик NPC (Actor Values), а також наявні інструменти для їх перегляду та редагування, такі як Creation Kit та xEdit. Обґрунтовано доцільність створення окремого програмного засобу для спрощення аналізу ігрових сутностей.

Розроблено алгоритм взаємодії користувача із застосунком, спроєктовано архітектуру на основі патерну MVVM, реалізовано WPF-застосунок із використанням бібліотек YAML.NET, Syncfusion та Mutagen.Bethesda.

Програма дозволяє здійснювати завантаження YAML-даних, пошук NPC, перегляд та редагування характеристик, зокрема значень здоров'я, магії, витривалості, перків і здібностей. Проведено функціональне тестування програми на прикладах реальних ігрових даних.

Сферою використання застосунку є допомога розробникам модифікацій (модів) у прискоренні аналізу ігрових даних Skyrim SE та їх попередній підготовці для подальшої обробки в інших інструментах.

КЛЮЧОВІ СЛОВА: SKYRIM SE, NPC, ACTOR VALUES, WPF, MVVM, MUTAGEN.BETHESDA

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NPC (Non-Player Character) – персонаж у грі, яким не керує гравець.

Actor Value, AV – внутрішній параметр, що описує певну характеристику ігрового персонажа: здоров'я, магію, витривалість, швидкість руху, опір магії тощо.

Template NPC – шаблон NPC, з якого наслідуються властивості іншими NPC. Часто використовується для скорочення дублювання даних.

Race – раса персонажа (людина, орк, альф, аргоніанець тощо), що визначає частину його базових характеристик.

Spell – закляття або магічна здібність, що може бути прив'язана до NPC.

Perk – пасивна або активна здатність, яка модифікує ігрову механіку для NPC (наприклад, посилення атаки, знижене споживання магії тощо).

ВСТУП

У сучасному цифровому середовищі відеоігри стали не лише джерелом розваг, але й полем для розвитку складних програмних систем, інтерактивних середовищ та аматорських і професійних модифікацій. Зокрема, рольові ігри з відкритим світом, як The Elder Scrolls V: Skyrim Special Edition, забезпечують гравцям високий рівень занурення завдяки великій кількості неігрових персонажів (NPC) з унікальними параметрами поведінки.

Однією з найбільш технічно складних задач для модмейкерів є редагування характеристик NPC — так званих Actor Values, які відповідають за здоров'я, магію, навички, швидкість руху та інші ігрові параметри. У процесі розробки модифікацій необхідно не лише змінювати ці значення, а й розуміти, звідки вони надходять: від самої раси, закріплених заклять, перків, скриптів або шаблону персонажа. Усе це формує складну ієрархію, аналіз якої у традиційних інструментах, таких як Creation Kit або xEdit, потребує багато часу і глибокого технічного розуміння. Розробнику доводиться переходити від одного запису до іншого, вручну виявляючи, який саме елемент гри впливає на ту чи іншу характеристику NPC.

З огляду на це виникла потреба у спеціалізованому програмному інструменті, який би спрощував аналіз ієрархії впливу на Actor Values NPC, надаючи користувачу зведену інформацію про джерела характеристик без необхідності самостійного пошуку в глибинах ігрових файлів.

Мета роботи — спрощення процесів відстеження і редагування характеристик NPC у відеогрі Skyrim.

Об'єкт дослідження — процес відстеження і редагування характеристик NPC у рольових іграх.

Предмет дослідження — засоби та технології для обробки ігрових даних, зокрема з використанням мови програмування C#, платформи WPF та бібліотеки Mutagen.Bethesda

Для досягнення мети поставлено задачі:

1. Проаналізувати технічну специфікацію гри Skyrim SE, зокрема структуру зберігання ігрових даних;
2. Дослідити можливості наявних інструментів редагування, таких як SSEEdit та Creation Kit;
3. Визначити функціональні та нефункціональні вимоги до програмного засобу для аналізу Actor Values NPC;
4. Побудувати UML-моделі, що відображають архітектуру та логіку роботи застосунку;
5. Реалізувати програму з підтримкою пошуку, фільтрації та відображення характеристик NPC з урахуванням джерел їх формування;
6. Провести тестування розробленого програмного забезпечення.

Практична новизна розробленого інструменту полягає в автоматизованому аналізі структури Actor Values з урахуванням усіх пов’язаних ігрових сутностей (Race, Spell, Perk тощо), що дає змогу швидко виявляти джерела тих чи інших параметрів без ручного перегляду пов’язаних записів. Завдяки цьому, модмейкери отримують інструмент для точнішого і швидшого редагування NPC, що особливо корисно при створенні комплексних ігрових змін.

1 АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ РЕДАГУВАННЯ ДАНИХ В SKYRIM SPECIAL EDITION

1.1 Огляд платформи Skyrim SE та її модифікаційної архітектури

The Elder Scrolls V: Skyrim Special Edition (Skyrim SE) — це покращена версія культової RPG з графічними та технічними оновленнями, яка зберегла базову ігрову механіку і структуру даних оригіналу. Гра базується на рушії Creation Engine, який підтримує гнучку систему модифікацій (модів), що дозволяє користувачам змінювати контент, поведінку NPC, локації, квести та багато іншого.

Ключовою особливістю є складна система характеристик персонажів (Actor Values), яка визначає їх поведінку, реакції, здібності та ефективність у грі. Ці характеристики формуються на основі численних джерел — спорядження, заклинань, активних ефектів, перків тощо.

Ігрові дані зберігаються у файлах з розширеннями .esm, .esp, .esl, які мають складну структуру ієрархічних записів. Для роботи з ними використовуються спеціалізовані бібліотеки, наприклад, Mutagen.Bethesda, що надають програмний доступ до модифікаційних даних.

1.2 Огляд існуючих інструментів редагування ігрових даних

Для ефективного створення, редагування та аналізу ігрового контенту, зокрема в межах розробки модифікацій для The Elder Scrolls V: Skyrim, на сьогодні існує низка спеціалізованих інструментів. Кожен із них має власне призначення, набір функціональних можливостей, рівень зручності користування та технічні обмеження. У цьому підрозділі буде розглянуто три ключові інструменти, що найбільш часто використовуються як у професійному, так і в аматорському

середовищі моддінгу: Creation Kit, SSEEdit (xEdit) та zEdit. Знання їх особливостей дозволяє не лише орієнтуватися у сучасному стані справ у сфері редагування контенту Skyrim, але й виявити певні технічні ніші, які залишаються незаповненими — що, своєю чергою, створює підґрунтя для обґрунтування розробки нових інструментів.

1.2.1 Creation Kit

Creation Kit є офіційним інструментом розробки модифікацій від компанії Bethesda Game Studios. Його статус як головного і рекомендованого редактора обумовлений повною інтеграцією з внутрішньою архітектурою рушія Creation Engine, на якому побудовано Skyrim. Іншими словами, цей інструмент виступає "еталонним" редактором, що гарантує максимально коректне збереження і трактування даних усіма версіями гри.

Серед найважливіших переваг Creation Kit варто відзначити:

− Повну підтримку всіх ігрових об’єктів. Користувач отримує доступ до редагування будь-якого аспекту гри: від NPC та рас до скриптів, квестів, діалогових дерев, інтерфейсних налаштувань, зон навігації та триггерів. Це створює враження універсального середовища розробки.

− Інструменти візуального редагування. Creation Kit забезпечує рідкісну для подібних інструментів можливість прямого редагування тривимірного світу гри. Розміщення моделей, будівництво інтер’єрів, налаштування світла — усе це реалізовано у вигляді інтерактивного 3D-редактора, що значно пришвидшує розробку контенту.

− Інтеграцію з мовою Papyrus. Завдяки повній підтримці скриптової мови Papyrus, розробник може не лише додавати нові об’єкти, а й повністю контролювати їх логіку поведінки у грі, обробку подій, перемикання станів тощо.

− Наявність офіційної документації та спільноти. Як продукт від Bethesda, Creation Kit супроводжується великим обсягом документації, прикладів, wiki-статей і форумів, що полегшує ознайомлення з його можливостями.

Попри ці переваги, Creation Kit має ряд вагомих недоліків, які часто згадуються у відгуках розробників:

- Застарілий інтерфейс. Графічне середовище редактора візуально майже не змінилося з часів Fallout 3 і вже давно не відповідає сучасним стандартам UX-дизайну. Багато функцій заховані за комбінаціями клавіш або у вкладках, які інтуїтивно складно знайти.
- Високе навантаження на систему. При роботі з великими модами або плагінами можливі значні затримки інтерфейсу, зависання, а іноді навіть аварійне завершення роботи редактора.
- Відсутність сучасних інструментів порівняння та злиття змін. Creation Kit не має повноцінної підтримки конфліктного аналізу, що унеможливлює гнучке редагування кількох модів одночасно або модифікацію вже змінених об’єктів.
- Висока ймовірність нестабільної роботи. Відомо про численні випадки випадкових вильотів, особливо при спробах редагувати складні квести або навігаційні шляхи NPC.
- Обмежені можливості автоматизації. Створення шаблонів або масове редагування об’єктів ускладнене через відсутність інструментів для зручної роботи з великим обсягом даних.

Таким чином, незважаючи на статус офіційного рішення, Creation Kit слугує радше комплексним, але негнучким інструментом, якому часто бракує швидкості, стабільності й сучасного підходу до автоматизації.

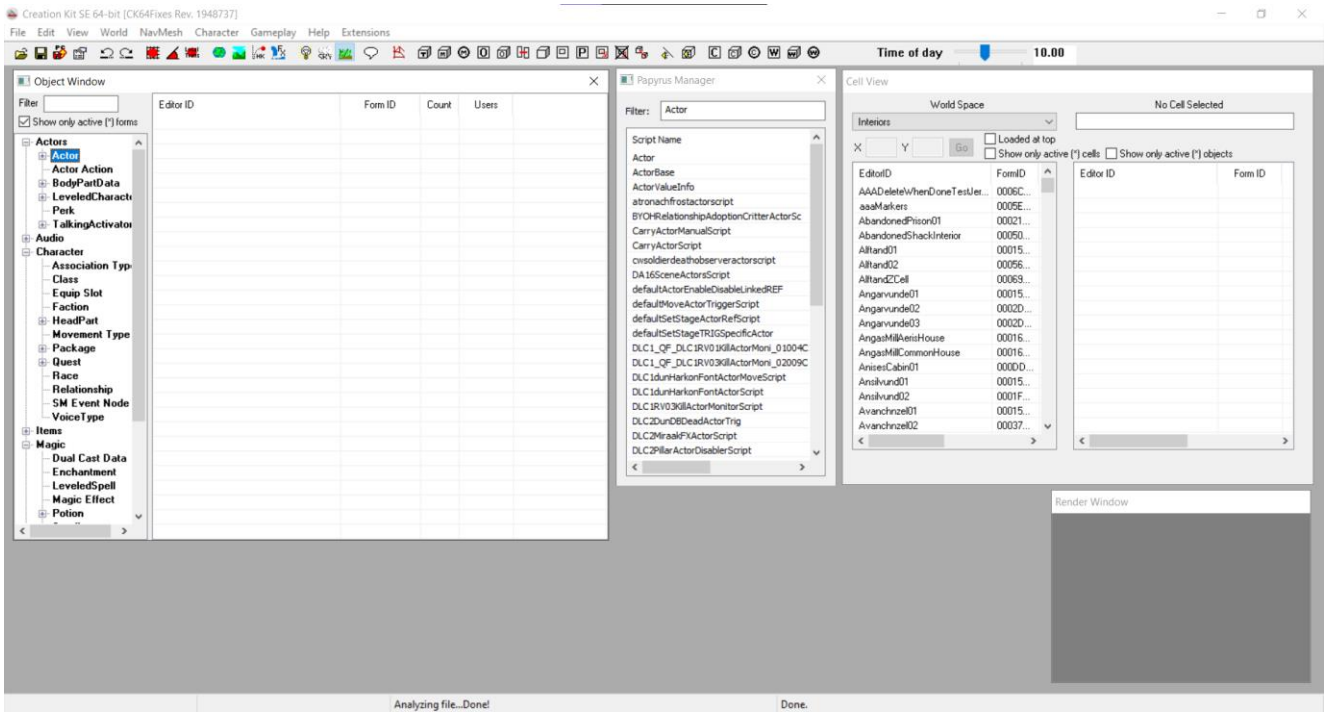


Рис. 1.1 Приклад інтерфейсу програми Creation Kit

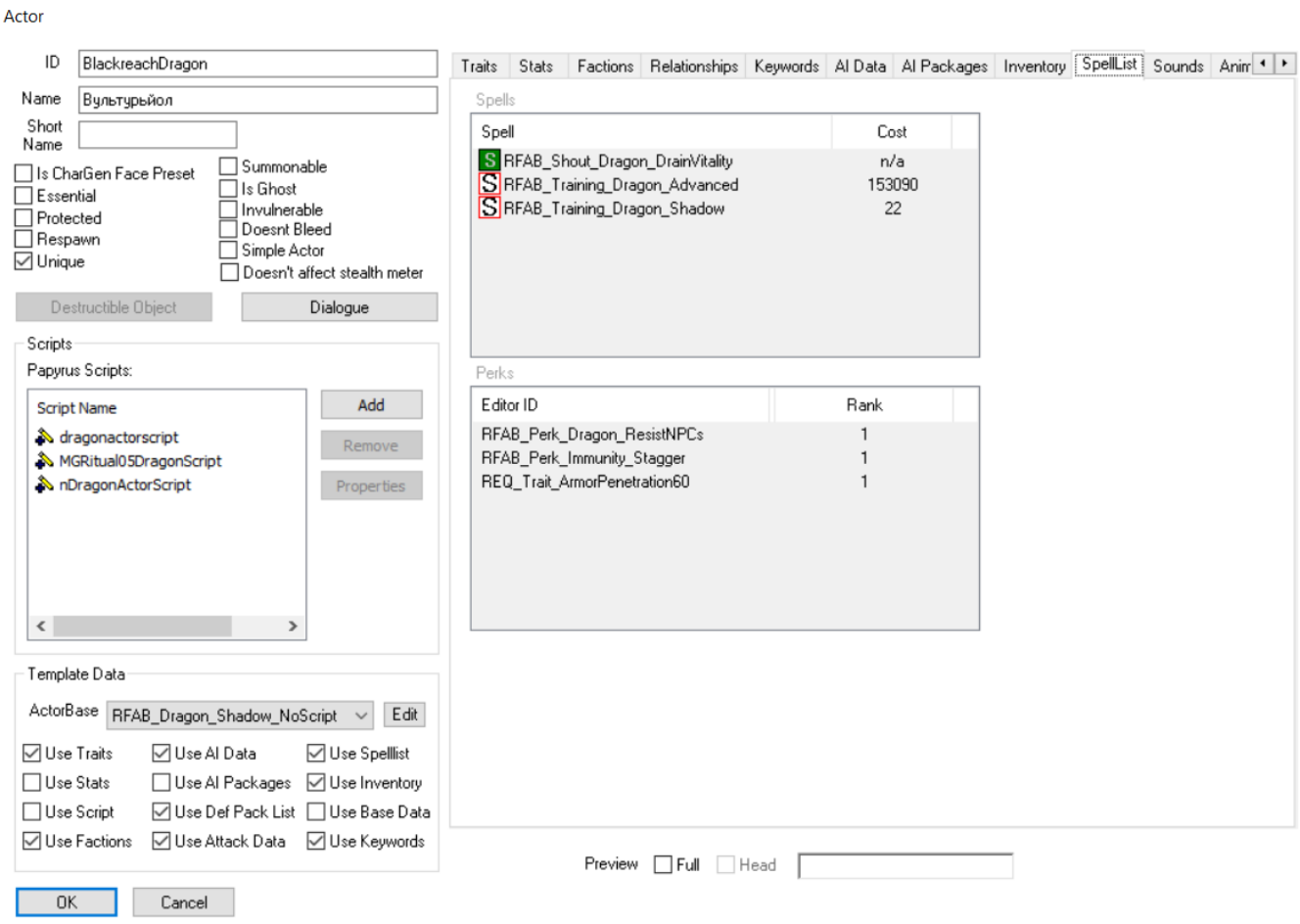


Рис. 1.2 Приклад запису NPC в Creation Kit

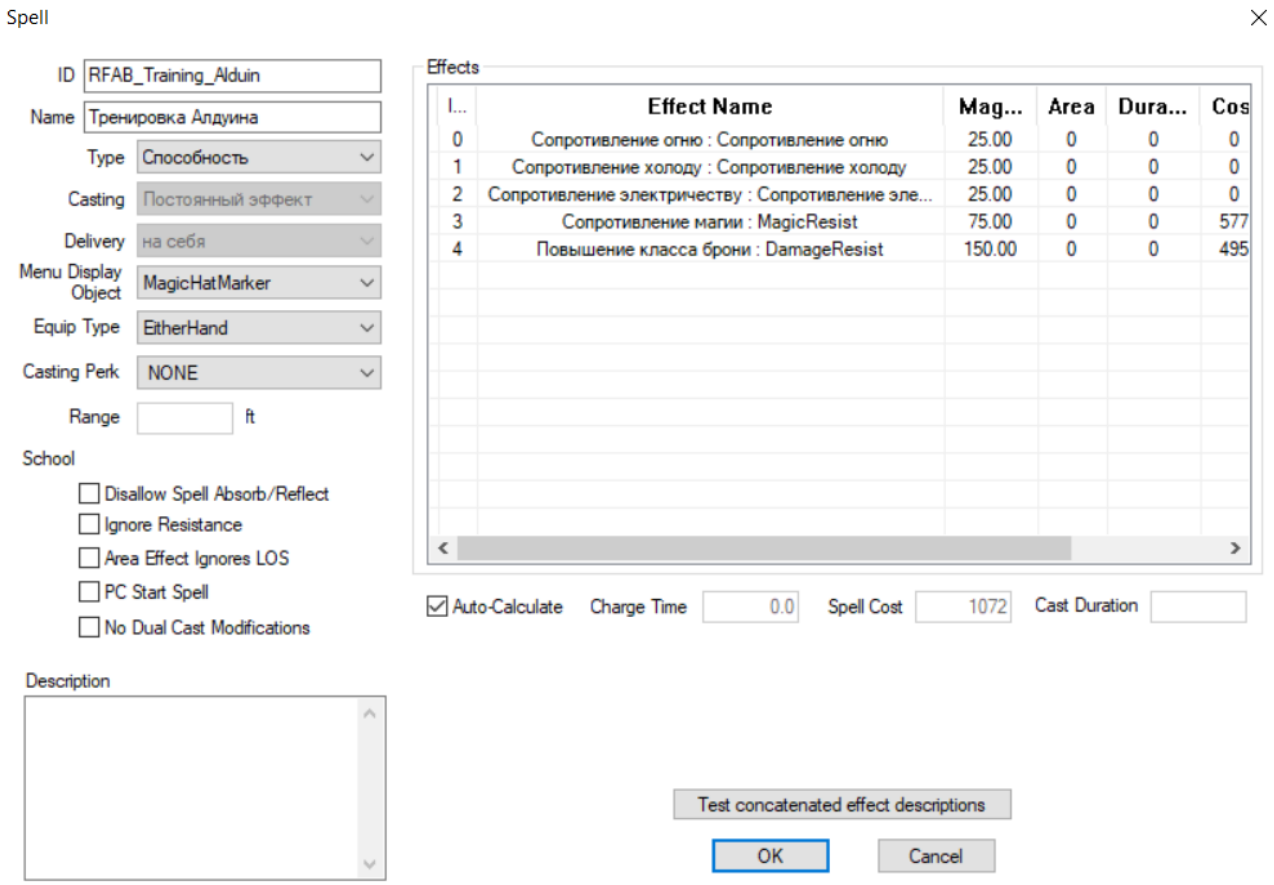


Рис. 1.3 Приклад запису Spell в Creation Kit

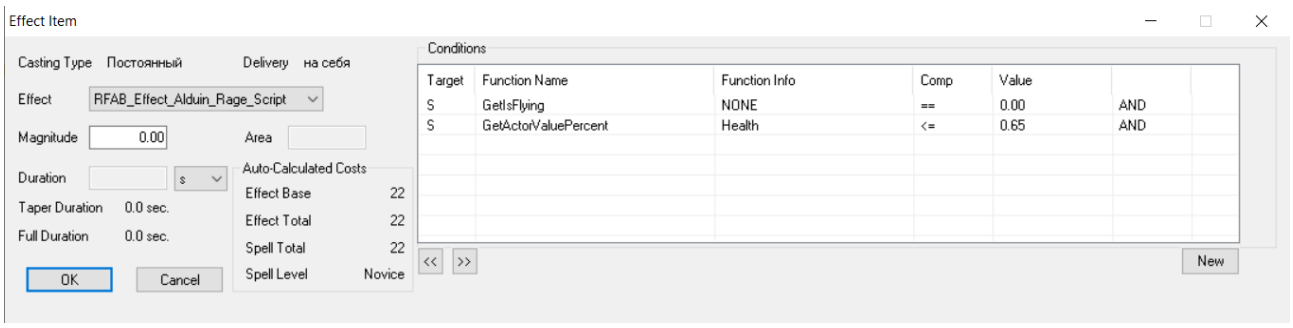


Рис. 1.4 Приклад умов Magic Effect в Creation Kit

1.2.2 SSEEdit

SSEEdit, також відомий під загальною назвою xEdit, є потужним незалежним редактором, створеним спільнотою ентузіастів. Він побудований за принципом максимальної близькості до внутрішньої структури ігрових файлів формату .esp, .esm, .esl, що робить його незамінним при аналізі та діагностиці модифікацій.

Основні переваги SSEEdit полягають у наступному:

− Низькорівневий доступ до даних. Програма дозволяє відкрити практично будь-який запис у грі, порівняти його значення між різними модами, а також переглянути "спадковість" змін між файлами.

− Швидкість роботи. SSEEdit запускається дуже швидко, споживає небагато пам’яті, працює стабільно навіть на застарілих комп’ютерах — чим вигідно відрізняється від Creation Kit.

− Система скриптів. Підтримка Pascal Script відкриває можливість створення власних інструментів для аналізу, автогенерації, злиття тощо.

− Потужний конфліктний аналіз. SSEEdit автоматично визначає та виділяє конфліктуючі записи, пропонуючи користувачу способи вирішення.

− Функції очищення (cleaning). Утиліта дозволяє видалити "брудні правки" з офіційних плагінів, що критично важливо для стабільності великих модифікацій.

Проте, як і будь-який утилітарний інструмент, SSEEdit має свої обмеження:

− Відсутність 3D-візуалізації. Редактор не підтримує перегляд об’єктів у просторі, через що його функціональність обмежена переважно табличним представленням.

− Складність для новачків. Інтерфейс вимагає базового розуміння структури плагінів та форм ID, а документація не завжди є достатньо зрозумілою для початківців.

– Мова скриптів — Pascal. Незважаючи на свою ефективність, Pascal є малопоширеним серед сучасних розробників, що обмежує розширення інструменту.

Таким чином, SSEEdit — це "скальпель" у світі моддінгу, здатний вирішити найтонші технічні проблеми, але непридатний для візуальної творчості чи швидкого ознайомлення новачків із контентом гри.

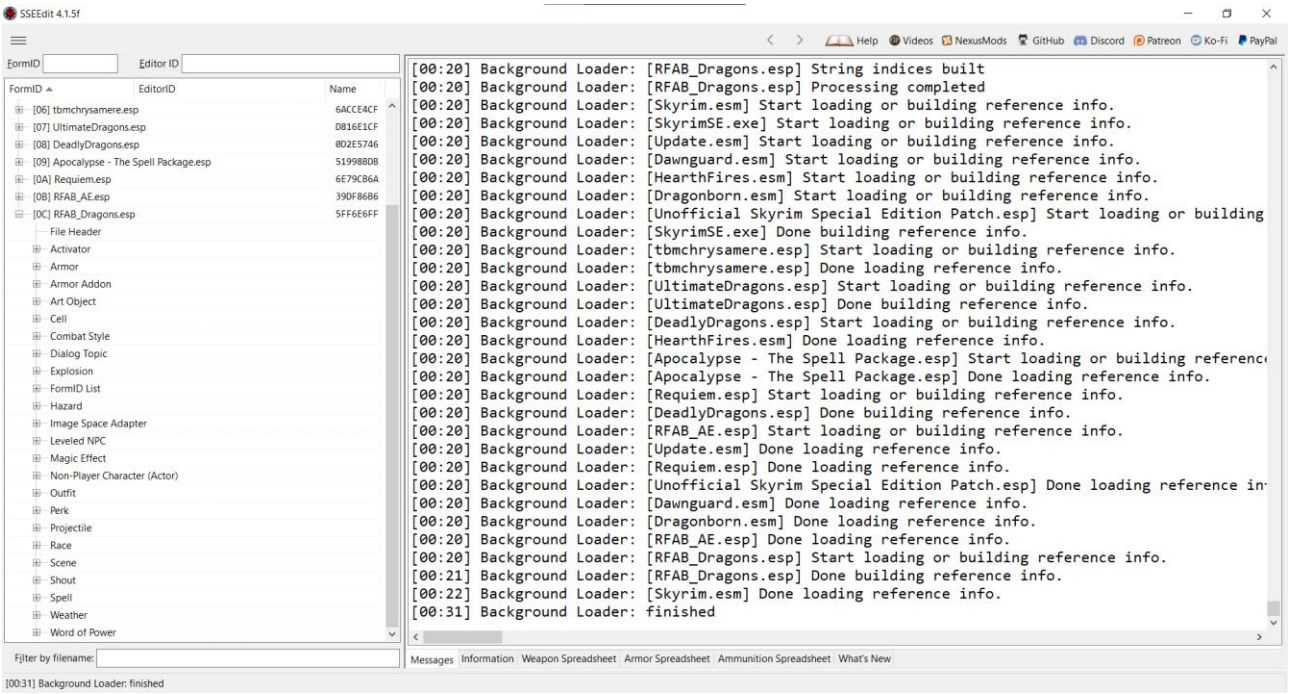


Рис. 1.5 Приклад інтерфейсу програми в SSEEdit

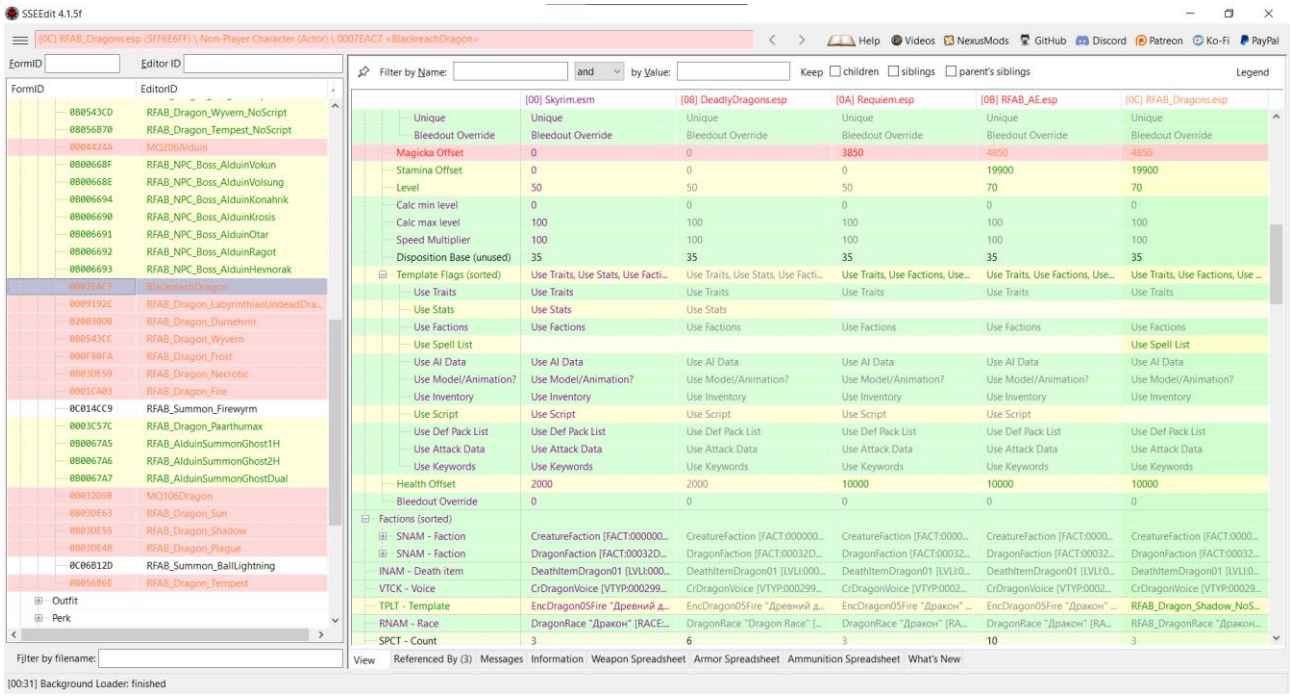


Рис. 1.6 Приклад запису NPC в SSEEdit

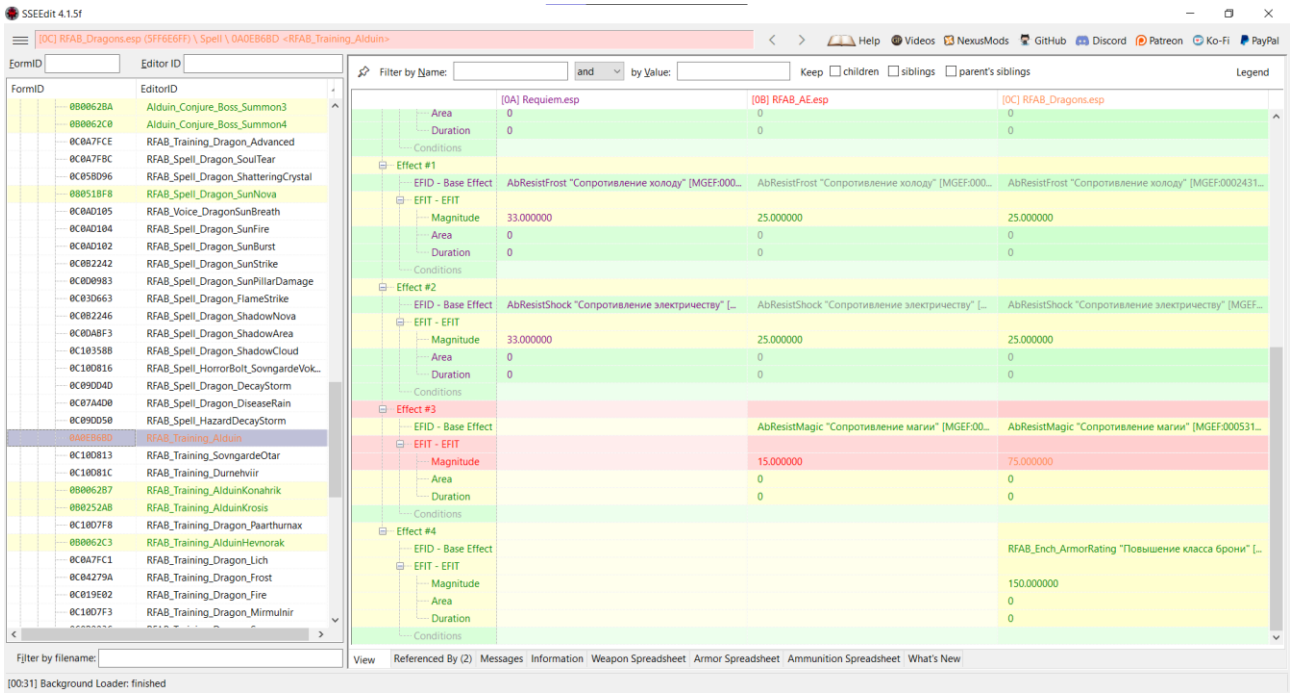


Рис. 1.7 Приклад запису Spell в SSEEdit

	[0C] RFAB_Dragons.esp
Archtype	Absorb
Actor Value	Stamina
Projectile	DLC01SoulRendProjectile [PROJ:02007CBB]
Explosion	NULL - Null Reference [00000000]
Casting Type	Fire and Forget
Delivery	Aimed
Second Actor Value	None
Casting Art	NULL - Null Reference [00000000]
Hit Effect Art	NULL - Null Reference [00000000]
Impact Data	VOCShoutPush01ImpactSet [IPDS:00015C7B]
Skill Usage Multiplier	1.000000
Dual Casting	
Art	NULL - Null Reference [00000000]
Scale	1.000000
Enchant Art	NULL - Null Reference [00000000]
Hit Visuals	NULL - Null Reference [00000000]
Enchant Visuals	NULL - Null Reference [00000000]
Equip Ability	NULL - Null Reference [00000000]
Image Space Modifier	DLC1SoulRendCastImod [IMAD:0201A3F8]
Perk to Apply	NULL - Null Reference [00000000]
Casting Sound Level	Very Loud
Script Effect AI	
Score	0.000000
Delay Time	0.000000
Counter Effects	
SNDD - Sounds	
#0	[Release] VOCShoutFXDrainVitalityA [SNDR:0200D94C]
DNAM - Magic Item Description	Высасывает <mag> ед. запаса сил в секунду в течение <dur> с.
Conditions	
Condition #0	Subject.HasMagicEffect(DLC1DragonDrainVitalityStaminaEffect [MGEF:0200844A]) = 0 AND
Condition #1	Target.GetDead = 0

Рис. 1.8 Приклад запису Magic Effect в SSEEdit

1.2.3 zEdit

zEdit є спробою переосмислення інтерфейсів для роботи з ігровими плагінами. Його основна особливість полягає у використанні вебтехнологій — зокрема, платформи Electron, що поєднує в собі можливості Node.js і браузерного рушія Chromium. Завдяки цьому, zEdit є, по суті, вебдодатком, що має доступ до тих самих даних, що й SSEEdit, але реалізує сучасні підходи до взаємодії з користувачем.

Переваги zEdit включають:

- Модернізований інтерфейс. Підтримка вкладок, тем, CSS-кастомізації, drag-and-drop, збереження макетів робочого простору тощо — усе це забезпечує гнучку та комфортну роботу.
- Модульність. zEdit побудований як платформа для плагінів: розширення на кшталт zMerge або zSmash дозволяють автоматизувати об’єднання модів або видалення дублікатів.
- JavaScript-скрипти. На відміну від SSEEdit, скрипти пишуться на популярній мові JavaScript, що значно розширює аудиторію потенційних розробників.
- Висока інтерактивність. Удосконалена система пошуку, кешування даних, віртуалізації таблиць забезпечує швидку взаємодію навіть з великим обсягом інформації.

Попри перспективність, zEdit має і серйозні вади:

- Велике споживання ресурсів. Electron-додатки за визначенням є "важкими" — zEdit споживає в рази більше пам’яті, ніж SSEEdit.
- Низька стабільність. Програма залишається експериментальною: є баги, відсутність деяких важливих функцій, можливі зависання при роботі з великими файлами.
- Низька частота оновлень. Розробка практично припинилася, а спільнота майже не підтримує нові модулі.

Отже, zEdit — це спроба "осучаснити" редагування модів, яка, попри гарну ідею, наразі страждає від нестачі стабільності та підтримки.

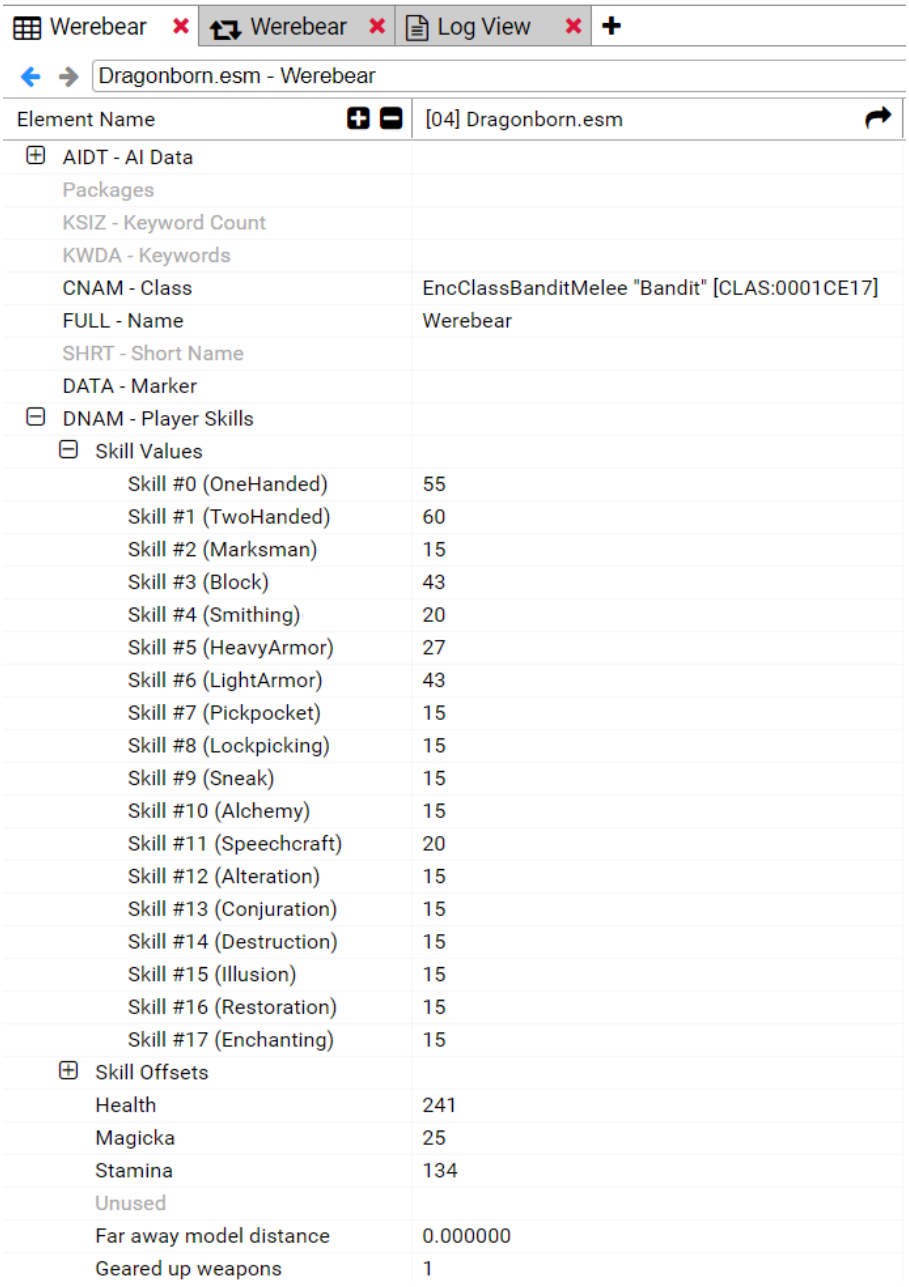


Рис. 1.9 Приклад запису NPC в zEdit

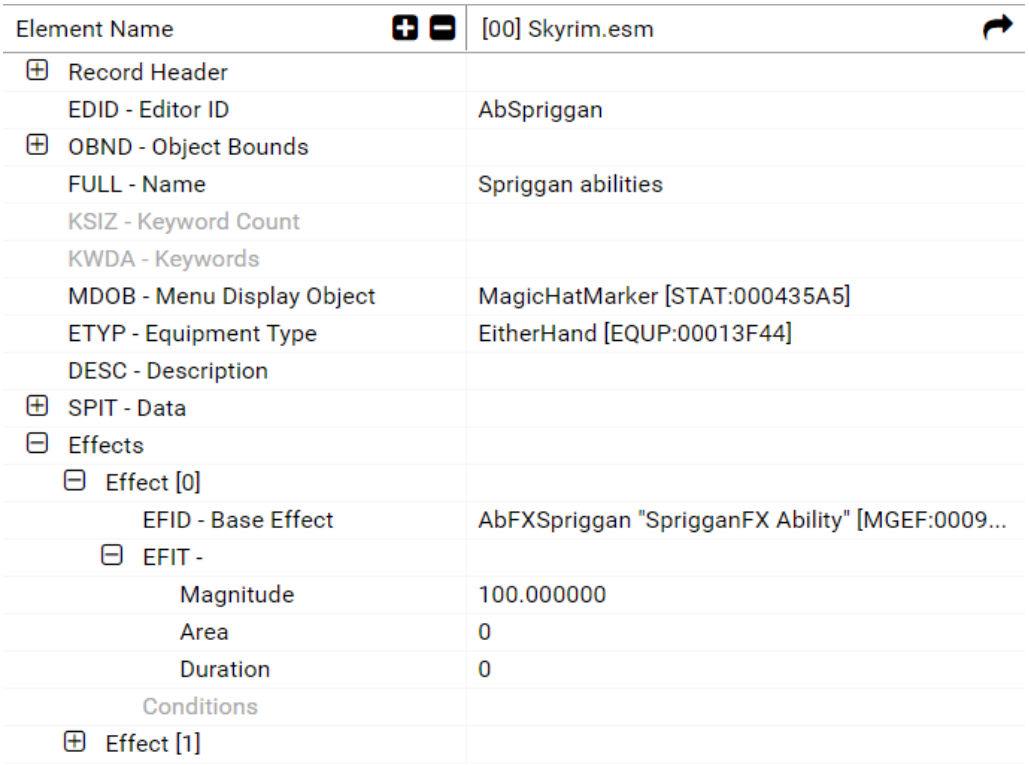


Рис. 1.10 Приклад запису Spell в zEdit

1.3 Визначення вимог до нового інструменту

Після аналізу можливостей існуючих інструментів редагування та аналізу модифікацій Skyrim SE, можна зробити висновок, що, незважаючи на потужність і гнучкість таких рішень, як Creation Kit, SSEEdit та zEdit, жодне з них не надає достатньо зручного й наочного способу перегляду, аналізу та порівняння характеристик персонажів (Actor Values). Більшість завдань, пов’язаних із виявленням джерел змін певного параметру NPC, вимагають глибоких знань структури даних та багатоетапного ручного аналізу. Це створює бар’єр для мододелів, особливо у випадках, коли необхідно швидко і точно зрозуміти, чому NPC має певне значення здоров’я, магії або інших характеристик.

1.3.1 Проблеми, не вирішені існуючими інструментами

Існуючі інструменти переважно орієнтовані на редагування даних, а не на їх аналіз. Вони не надають швидкого способу перегляду змін, внесених різними джерелами (наприклад, заклинаннями, расами, перками).

- У Creation Kit відсутні засоби порівняння, фільтрації або побудови дерева залежностей між ігровими об’єктами, які впливають на AV.
- SSEEdit дозволяє побачити джерела змін, однак вимагає ручного перегляду десятків записів у складному ієрархічному форматі, що ускладнює роботу навіть досвідченим користувачам.
- Немає візуалізації або інтуїтивно зрозумілих схем, які б пояснювали "чому це значення таке".
- Відсутні прості засоби масового перегляду та сортування персонажів за характеристиками, що обмежує можливості для тестування збалансованості модів або виявлення аномалій.

1.3.2 Цільова аудиторія та сценарії використання

Метою розробки нового інструменту є забезпечення зручного середовища для перегляду та аналізу Actor Values в NPC гри Skyrim SE. Очікується, що цим інструментом користуватимуться:

- Автори модифікацій, яким потрібно швидко оцінити ефекти раси, спорядження, магії та перків на характеристики персонажа.
- Розробники збірок модів, які прагнуть уникнути конфліктів між модами та забезпечити стабільність значень AV.
- Тестувальники, які перевіряють внутрішню логіку та баланс змінених NPC.
- Дослідники ігрової механіки, які потребують системного підходу до аналізу ефектів без втручання у самі файли гри.

Інструмент не призначений для повноцінного редагування плагінів або модифікацій, за винятком обмеженого редагування характеристик NPC (Health, Magicka, Stamina, Skill Values, Skill Offsets) — лише у випадках, коли персонаж не наслідує інші шаблони.

1.3.3 Основні функціональні вимоги до нового інструменту

На основі проаналізованих потреб можна сформулювати такі ключові вимоги до функціональності:

- Модульна архітектура інтерфейсу, де кожен тип даних (NPC, Race, Spell, Perk) представлений у вигляді окремих секцій з можливістю швидкої навігації.
- Механізм пошуку та фільтрації, що дозволяє миттєво знаходити потрібних персонажів за ім’ям або ID.
- Відображення джерел формування Actor Values — можливість бачити, які саме об'єкти гри (Race, Perk, Spell тощо) вплинули на кожне значення AV, з деталізацією за типом впливу.
- Зведена таблиця характеристик для NPC з можливістю порівняння.
- Збереження локальної копії даних у зручному форматі для подальшого аналізу або документації (наприклад, для експорту в текстові звіти чи таблиці).
- Підтримка великої кількості об'єктів без зниження продуктивності.
- Стабільність та незалежність від внутрішніх редакторів Bethesda — інструмент має працювати автономно на основі розпакованих даних.

1.3.4 Технічні припущення та обмеження

Інструмент розрахований переважно на читання даних та їх візуалізацію — зміни до .esm/.esp/.esl-файлів не зберігаються, за винятком окремих випадків.

У межах можливостей бібліотеки Mutagen.Bethesda програма отримує доступ до ігрових записів у вигляді об’єктної моделі, що дозволяє здійснювати детальний аналіз їхнього складу.

Передбачається, що користувач має доступ до попередньо розпакованих або змонтованих ігрових файлів у підтримуваному форматі.

Розрахунок значень Actor Values та їхнє відображення здійснюються динамічно при відкритті кожного NPC, з урахуванням усіх джерел модифікацій — рас, перків, ефектів тощо.

Основна увага зосереджена на простоті використання, зрозумілості представлення даних та швидкому доступі до ключових характеристик персонажа.

При цьому реалізовано можливість обмеженого редагування окремих параметрів NPC (Health, Magicka, Stamina, Skill Values, Skill Offsets), якщо об’єкт не наслідує інші шаблони або сутності. В усіх інших випадках редагування не допускається.

Таким чином, розробка інструменту є спробою вирішити практичну задачу — надання користувачу зручного засобу для аналізу характеристик NPC без потреби занурення в складну внутрішню структуру Skyrim вручну.

1.4 Зведений аналіз аналогів

У процесі аналізу існуючих рішень для роботи з ігровими даними Skyrim SE було розглянуто три основні інструменти: Creation Kit, SSEEdit та zEdit. Кожен з них має власну спеціалізацію і набір можливостей, однак жоден не фокусується на зручному аналізі Actor Values NPC у вигляді, що дозволяє швидко оцінити джерела формування характеристик та причини їхніх значень.

Нижче наведено короткий опис особливостей кожного з інструментів:

- Creation Kit — офіційний редактор, орієнтований на створення та редагування ігрових об'єктів з інтеграцією 3D-контенту. Не підтримує комплексного перегляду впливів на AV або їх порівняння.
- SSEEdit — потужний редактор плагінів з можливістю перегляду конфліктів та змін між модами. Має доступ до всіх записів, однак не надає зручного способу інтерпретувати результативні значення AV.

− zEdit — модульна платформа для розширеного редагування та скриптової обробки плагінів, однак підтримка актуальних версій обмежена, а фокус зосереджений більше на групових змінах, ніж на аналізі окремих NPC.

Результати аналізу програмних засобів подано в таблиці 1.1.

Таблиця 1.1

Результат аналізу існуючих програм

	Creation Kit	SSEEdit	zEdit	SAVE
Тип програми	Офіційний редактор контенту	Редактор та аналізатор плагінів	Модульна платформа для роботи з плагінами	Спеціалізований інструмент для перегляду Actor Values
Простота використання	Складний, особливо для новачків	Потребує розуміння структури ESP/ESM	Складна навігація	Простий інтерфейс
Швидкість і стабільність	Може бути повільним і аварійним	Загалом стабільний, швидкий	Можливі збої під час роботи	Швидкість і стабільність
Гнучкість і можливості	Максимальна гнучкість: підтримка скриптів, квестів тощо	Глибокий доступ до даних	Розширюєтьс я через плагіни, модулі	Гнучкість і можливості
Візуалізація характеристик NPC	Немає зручного перегляду	Відсутня візуалізація actor values	Обмежена, без спеціалізован ого перегляду	Візуалізація характеристик NPC
Редагування даних	Повне редагування плагінів	Пряме редагування ESP/ESM	Повноцінне редагування через модулі	Редагування даних

1.5 Структура даних у Skyrim SE

Skyrim Special Edition (SE) — це складна рольова гра з відкритим світом, яка використовує власний движок Bethesda Game Engine. Вся ігрова інформація, що описує світ, персонажів, предмети, магічні ефекти, а також логіку ігрових процесів, зберігається у вигляді плагінів і ресурсних файлів із розширеннями .esm, .esp та .esl.

1.5.1 Основні формати файлів

.esm (Elder Scrolls Master) — це основні, «базові» файли гри, які містять офіційний контент Bethesda: світ, локації, NPC, квести, текстури та інші ресурси. Вони є «мастер-файлами», від яких залежать всі додаткові модифікації. Вони завантажуються першими під час запуску гри.

.esp (Elder Scrolls Plugin) — плагіни користувача або мододелів, які змінюють, доповнюють чи переозначають дані з .esm або інших .esp файлів. Вони дозволяють додавати нові локації, змінювати характеристики персонажів, додавати нові предмети, квести тощо.

.esl (Elder Scrolls Light Plugin) — це легковагові плагіни, які мають обмеження на розмір та кількість записів, призначені для невеликих модифікацій. Підходять для збільшення кількості плагінів без перевищення обмеження гри.

1.5.2 Записи та структура даних

Ігрові дані організовані у вигляді записів (Records). Кожен запис описує певний об’єкт або сутність — NPC, предмет, заклинання, перк, локацію тощо. Записи мають унікальні ідентифікатори (FormID) і містять поля (Fields) з параметрами, які задають властивості об’єкта.

1.5.3 NPC і Actor Values

NPC (Non-Player Character) — це ігрові персонажі, яких контролює штучний інтелект або гравець. Записи NPC містять такі важливі параметри:

- Основні характеристики: ім’я, расу, клас, рівень.
- Візуальні атрибути: зовнішність, одяг, анімації.
- Поведінкові характеристики: патрулі, діалоги, реакції.
- Actor Values — набір параметрів, які впливають на ігрові механіки персонажа. Наприклад, здоров’я (Health), магічна енергія (Magicka), витривалість (Stamina).

Actor Values формуються з урахуванням кількох джерел:

- Базові значення, встановлені безпосередньо в записі NPC.
- Раса персонажа — наприклад, різні раси мають свої бонуси та штрафи.
- Ефекти перків, які додають або змінюють параметри.
- Вплив заклинань, зачарувань, спорядження.
- Тимчасові або постійні ефекти, які модифікують AV під час гри.

Через таку складну систему значення AV для одного й того ж NPC можуть значно варіюватися залежно від комбінації модів, обладнання та ігрової ситуації.

1.5.4 Особливості роботи з файлами

Для аналізу та редагування NPC і їх характеристик важливо мати можливість розібрати всю цю ієрархію, зрозуміти, звідки приходять зміни, і в якому порядку вони накладаються. Проте самі файли .esm і .esp мають складну двійкову структуру, що ускладнює безпосередню роботу з ними.

Для роботи з цими файлами використовують спеціалізовані бібліотеки та інструменти, які перетворюють двійкові дані у зручні об’єктні моделі, що дозволяють:

- Отримувати повний список NPC.
- Визначати базові та модифіковані значення AV.
- Аналізувати залежності між різними записами.
- Виявляти джерела змін параметрів.

Це є ключовим для розробки аналітичних інструментів, які допомагають мододелам і розробникам краще розуміти вплив модифікацій на ігровий баланс.

1.6 Опис бібліотеки Mutagen.Bethesda для роботи з ігровими файлами

Однією з ключових складових реалізації інструменту аналізу характеристик персонажів у Skyrim SE є робота з ігровими файлами у форматах .esm, .esp, .esl. Ці файли є основними контейнерами, в яких зберігається вся інформація про вміст гри: об'єкти, персонажі, скрипти, діалоги, квести, раси, заклинання та інші ігрові сутності. Оскільки формат файлів Bethesda є бінарним, для прямого доступу до даних потрібна спеціалізована бібліотека.

Для вирішення цього завдання у проєкті використано бібліотеку Mutagen.Bethesda — сучасний інструмент з відкритим кодом, написаний на мові програмування C# і призначений для парсингу та маніпуляції вмістом файлів плагінів Bethesda Game Studios. Mutagen підтримує такі ігри, як Skyrim SE, Fallout 4, Oblivion та інші, забезпечуючи єдиний уніфікований інтерфейс для взаємодії з їхніми ресурсами.

1.6.1 Загальна характеристика бібліотеки

Mutagen — це бібліотека, що надає об'єктну модель для читання, створення та редагування записів, які зберігаються у файлах плагінів. Вона побудована з орієнтацією на:

- Безпечну десеріалізацію даних із врахуванням специфіки кожної гри;
- Типізовану структуру — кожен запис має відповідний клас (наприклад, Npc, Race, Spell, Perk, ActorValueInfo тощо);
- Мінімальну залежність від специфічних API гри — Mutagen працює з файлами напряму, без запуску ігрового рушія;
- Можливість розширення — бібліотека активно розвивається спільнотою мододелів.

Mutagen.Bethesda не вимагає встановленої гри або Creation Kit. Все, що потрібно — доступ до розпакованих ресурсів (Skyrim.esm, Update.esm, модифікацій .esp, .esl тощо), які можна обробити програмно.

У контексті проєкту особливо важливими є такі можливості Mutagen:

- Читання та фільтрація записів NPC (Npc) — можна отримати доступ до базових параметрів: Health, Magicka, Stamina, навички, перки, заклинання тощо.
- Побудова зв’язків між сутностями — наприклад, NPC → Race → Perk → Entry Point → Ability → Effect → Actor Value.
- Підтримка ієрархічної моделі наслідування — можна враховувати записи, які мають батьківські сутності (наприклад, TemplatedFrom).
- Можливість обмеженого редагування записів — додавання або зміна параметрів для незв'язаних NPC.

Бібліотека дозволяє працювати не лише з основною грою, а й з додатковими плагінами, враховуючи порядок завантаження (Load Order) — важливий аспект для аналізу реальних значень AV.

1.6.2 Архітектура роботи з файлами

Типовий процес роботи з Mutagen передбачає:

– Завантаження модулів (файлів плагінів);

– Створення об'єкта Mod, який містить колекцію записів різних типів;

– Отримання потрібного запису (наприклад, INpcGetter) через LINQ або прямий доступ;

– Аналіз властивостей та пов'язаних сутностей, використовуючи поля типу FormLink<T>, IReadOnlyList<...> тощо;

– Збереження зміненого моду — якщо необхідно експортувати результати аналізу або редагування.

```
var mod = SkyrimMod.CreateFromBinary("MyMod.esp", SkyrimRelease.SkyrimSE);

foreach (var npc in mod.Npcs)
{
    Console.WriteLine($"NPC: {npc.EditorID}, Health: {npc.Configuration?.HealthOffset}");
}
```

Рис. 1.11 Приклад простого читання NPC

Переваги Mutagen

– Висока швидкість обробки великих обсягів даних;

– Безпечне й типізоване API для C#;

– Активна підтримка спільноти мододелів;

– Легко інтегрується у WPF-додатки.

Обмеження Mutagen

– Не є інструментом для візуального редагування — призначений лише для програмного доступу до даних.

– Відсутня повноцінна офіційна документація — для роботи з Mutagen необхідно аналізувати вихідний код або приклади з відкритих репозиторіїв.

1.7 Технології розробки інтерфейсу: WPF, MVVM, Syncfusion

1.7.1 Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF) - це потужна платформа для створення настільних застосунків із графічним інтерфейсом користувача під операційні системи Windows. Вона входить до складу .NET Framework і .NET Core (пізніше .NET 5+), забезпечуючи сучасні засоби побудови UI з глибоким розділенням логіки й подання.

Однією з головних переваг WPF є використання декларативної мови XAML (Extensible Application Markup Language) для опису елементів інтерфейсу. Це дозволяє дизайнерам і розробникам працювати паралельно: розмітка й стиль оформлення створюються у XAML, а бізнес-логіка - у мовах .NET (наприклад, C#).

Серед ключових переваг WPF можна виділити:

- Потужна система стилів і шаблонів: кожен елемент управління може мати свій стиль і шаблон, що дозволяє повністю змінювати вигляд без зміни логіки.
- Підтримка прив'язки даних (Data Binding): автоматичне оновлення інтерфейсу при зміні властивостей у моделі.
- Векторна графіка: рендеринг на базі DirectX забезпечує масштабованість і плавність інтерфейсу.
- Можливість створення анімацій, трансформацій, візуальних ефектів.
- Легке розділення інтерфейсу на модулі завдяки UserControl і механізмам ресурсів.

У даному проєкті WPF використовується як основна платформа для побудови віконної частини застосунку. Головне вікно містить вкладки з різними типами ігрових об'єктів (NPC, Race, Spell тощо), а також візуалізацію дерева характеристик на основі структури Actor Values. Кожен тип даних виводиться у власному UserControl, який автоматично підключається до відповідної ViewModel завдяки шаблонам DataTemplate.

1.7.2 MVVM (Model-View-ViewModel)

MVVM (Model-View-ViewModel) - це архітектурний шаблон, спеціально адаптований під особливості WPF і XAML. Його головною метою є чітке розділення відповідальності між різними частинами застосунку:

- Model - рівень даних, який представляє об'єкти з ігрових файлів (наприклад, Npc, Spell, Perk).
- View - графічний інтерфейс, створений на XAML, що відповідає за взаємодію з користувачем.
- ViewModel - проміжний шар, який надає View усі необхідні властивості та команди, керуючи прив'язкою та логікою.

У проєкті реалізовано базову ієрархію ViewModel-класів, зокрема:

- BaseViewModel - абстрактна базова ViewModel, що реалізує інтерфейс INotifyPropertyChanged і містить загальні утиліти;
- MainWindowViewModel - головна ViewModel програми, відповідає за навігацію між вкладками, запуск завантаження даних і вибір NPC;
- NpcCollectionView - модель для роботи з колекцією NPC, дозволяє фільтрувати та вибирати персонажів для аналізу;
- ActorValueTreeView - модель представлення дерева Actor Values, будується з урахуванням впливу усіх доступних джерел: раси, заклять, перків, умінь.

Завдяки MVVM структура програми є гнучкою, зручною для розширення та модульною. Зокрема, можна легко додати новий тип ігрових об'єктів або нову вкладку, створивши відповідну ViewModel і View, без потреби змінювати загальну логіку MainWindow.

1.7.3 Syncfusion

Syncfusion - це популярна бібліотека сторонніх UI-компонентів, яка значно розширює можливості стандартних елементів WPF. Вона містить велику кількість високопродуктивних, гнучко налаштованих контролів, що оптимізовані для корпоративних застосунків і складної візуалізації даних.

У межах цього проєкту використано компонент TreeGrid, який дозволяє будувати багаторівневе ієрархічне дерево з підтримкою розгортання, сортування, відображення властивостей та інтерактивної роботи з вузлами. TreeGrid є ідеальним вибором для візуалізації впливу різних елементів на Actor Values персонажа, оскільки кожне джерело (раса, перк, спел) може мати підлеглі впливи, які відображаються у вкладеній структурі.

Інтеграція Syncfusion дозволила значно покращити користувацький досвід, зробити інтерфейс більш інформативним, зрозумілим і швидким у роботі. Також компоненти бібліотеки мають підтримку тем, високої продуктивності, адаптивної верстки та сучасного зовнішнього вигляду.

2 ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

2.1 Постановка задачі та вимоги до функціоналу

Головною метою розробки програмного застосунку Skyrim Actor Value Editor є створення інструменту, який дозволить моддерам і розробникам модифікацій для гри Skyrim Special Edition ефективно аналізувати, переглядати та частково редагувати характеристики персонажів (NPC). Skyrim як гра має надзвичайно складну систему акторських значень (Actor Values), які безпосередньо впливають на ігрову поведінку, бої, здібності, ефекти та інші аспекти. Ці значення формуються як сукупність базових параметрів, спадкувань від шаблонів, а також змін від перків, заклять, броні, ефектів і скриптів.

Задача застосунку полягає у тому, щоб у зручній формі надати доступ до всіх цих значень, ураховуючи їх ієрархію та умови застосування, з можливістю виконувати:

- швидкий пошук і фільтрацію NPC за різними ознаками,
- огляд повного дерева характеристик для обраного NPC,
- часткове редагування цих характеристик,
- збереження внесених змін у новому моді, який не пошкоджує вихідні дані.

Унікальна складність системи Skyrim полягає в тому, що характеристики NPC — це не просто набір статичних значень. Вони формуються динамічно через безліч джерел і мають багато умовних залежностей. Традиційні інструменти для редагування модів часто мають або надто складний інтерфейс, або не підтримують зручного агрегування цих даних. Тому розробка спеціалізованого редактора з фокусом на Actor Values є актуальним завданням.

Основні вимоги до функціоналу:

− Імпорт і обробка ігрових даних із модів Skyrim Special Edition у форматах ESP/ESM.

− Автоматичне завантаження основних модів та підтримка великої кількості NPC без втрати продуктивності.

− Відображення значень Actor Values у вигляді ієрархічного дерева з можливістю згортання та розгортання гілок.

− Вказання джерел значень характеристик (раса, перки, заклинання, екіпіровка, базові параметри).

− Пошук NPC за іменем, EditorID або FormKey, із підтримкою фільтрації та сортування.

− Можливість редагування окремих характеристик із подальшим збереженням змін у вигляді окремого ESP-файлу без модифікації оригінальних даних.

2.2 Архітектура програми: загальний огляд та модульність

Архітектура програмного застосунку є фундаментальною складовою, що визначає загальну структуру системи, взаємодію її компонентів та організацію функціональних модулів. Правильне проектування архітектури забезпечує гнучкість, простоту підтримки та можливість подальшого розвитку програми.

У розробці застосунку Skyrim Actor Value Editor було прийнято архітектурний підхід, який дозволяє ефективно розподілити відповідальність між різними компонентами, водночас підтримуючи зручність розробки і масштабованість.

Для побудови інтерфейсу користувача застосовано патерн Model-View-ViewModel (MVVM). Цей підхід є стандартним для застосунків на основі WPF, оскільки забезпечує чисте розділення логіки від презентації і спрощує тестування.

У патерні MVVM виділяють три основні частини:

Model — представляє бізнес-логіку і структуру даних. В моделі зберігаються об’єкти, які містять основні дані NPC, їх характеристики і пов’язані сутності.

View — це інтерфейс користувача, який відповідає за відображення інформації. View реалізований у вигляді XAML-файлів з описом візуальних компонентів.

ViewModel — проміжний рівень, що містить логіку взаємодії між View та Model. ViewModel управляє даними, забезпечує їх відфільтровування, сортування, а також реакцію на дії користувача (наприклад, команди збереження або копіювання).

2.3 Основні модулі програми

Ядро програми (Core) — відповідає за початкову ініціалізацію, налаштування бібліотек та управління життєвим циклом застосунку. Включає в себе реєстрацію ліцензійних ключів, налаштування консолі для відлагодження і запуск основних сервісів.

Моделі даних (Models) — містять класи, що описують NPC, їх характеристики (Actor Values), а також складні структури даних, наприклад, ієрархічне дерево значень. Ці класи відповідають за збереження і обробку інформації, не включаючи логику відображення.

Моделі представлення (ViewModels) Містять логіку, що забезпечує взаємодію між моделями та інтерфейсом. Тут реалізовано фільтри, сортування, пошук по NPC, а також команди для роботи з даними (збереження, копіювання тощо). ViewModel-и повідомляють інтерфейс про зміни даних для автоматичного оновлення.

Інтерфейс користувача (Views) — складається з XAML-файлів, що описують зовнішній вигляд застосунку. Views відображають дані, передані з ViewModel, і не містять власної логіки обробки.

Побудовники дерев (NodeBuilders) — відповідають за формування ієрархічної структури характеристик NPC. Кожен побудовник відповідає за окремий тип даних (перки, броня, закляття, расові особливості) та додає відповідні вузли в дерево.

Сервіси (Services) — забезпечують роботу з зовнішніми ресурсами, такими як ігрові файли, кешування NPC, управління конфігураціями та збереження змін. Відокремлені від UI і надають інтерфейси для інших компонентів.

Утиліти та розширення (Utils, Extensions) — допоміжні класи для форматування, конвертації текстів, роботи з кодуваннями та інші утиліти, що підтримують основну логіку програми.

Взаємодія компонентів

При запуску ядро застосунку ініціалізує сервіси, зокрема завантажує модифікації Skyrim через GameContext.

NpcService завантажує і кешує NPC з модів, надаючи доступ до їх характеристик.

Моделі NPC та їх Actor Values формують основу даних для ViewModel, де реалізуються фільтрація, сортування і пошук.

За вибором користувачем NPC, NodeBuilders формують дерево властивостей, яке відображається у View.

Зміни, внесені користувачем, проходять через ViewModel і зберігаються через сервіси у вигляді окремого модифікованого ESP-файлу.

Переваги обраної архітектури

Чітке розмежування логіки, даних та інтерфейсу, що підвищує зручність підтримки та розвитку.

Гнучкість у додаванні нових функціональних блоків без значних змін у коді.

Масштабованість — ефективна робота з великою кількістю NPC і складними даними.

Реактивність — зміни в даних автоматично оновлюються у користувацькому інтерфейсі.

Можливість тестування логіки окремо від UI, що підвищує якість і надійність коду.

2.4 Реалізація основних компонентів

Реалізація основних компонентів програми Skyrim Actor Value Editor базується на чіткій організації коду за принципами модульності та розділення обов’язків, що дозволяє забезпечити масштабованість, підтримку та зрозумілість коду. У цьому розділі детально розглядаються ключові компоненти, які складають ядро функціональності програми.

2.4.1 Ядро програми

Ядро застосунку відповідає за ініціалізацію, налаштування та управління життєвим циклом програми. Зокрема, у цьому модулі реалізовано:

- Реєстрацію ліцензії Syncfusion, що необхідна для використання компонентів інтерфейсу.
- Ініціалізацію консолі відлагодження, яка може бути увімкнена або вимкнена через конфігураційні налаштування.
- Старт основних сервісів, таких як GameContext, що відповідає за роботу з ігровими файлами.

2.4.2 Моделі даних

Моделі даних формують основу інформації про NPC та їх характеристики. Вони забезпечують доступ до даних з ігрових файлів і включають:

- NpcModel — клас-обгортка над базовим інтерфейсом INpcGetter з Mutagen.Bethesda. Цей клас додатково реалізує функціональність обробки шаблонів (Templates) і форматування даних для відображення у інтерфейсі.
- ActorValueNode та похідні — класи, що реалізують ієрархічну структуру значень акторів. Вони представляють як базові характеристики (Health, Magicka), так і джерела цих значень (Perks, Armor, Spells).

2.4.3 ViewModels

ViewModels служать мостом між даними і користувацьким інтерфейсом. Головний клас MainWindowViewModel містить:

- Колекцію NPC, яка підтримує фільтрацію та сортування.
- Дерево значень акторів, яке динамічно оновлюється при виборі NPC.
- Команди, що відповідають за взаємодію користувача з програмою: збереження змін, копіювання значень, оновлення фільтрів.

Завдяки реалізації інтерфейсу INotifyPropertyChanged, ViewModels автоматично повідомляють View про зміни в даних, що забезпечує реактивність інтерфейсу.

На рис. 2.1 показано діаграму основних класів програмного забезпечення, яка реалізована з використанням архіеткрури ViewModels.

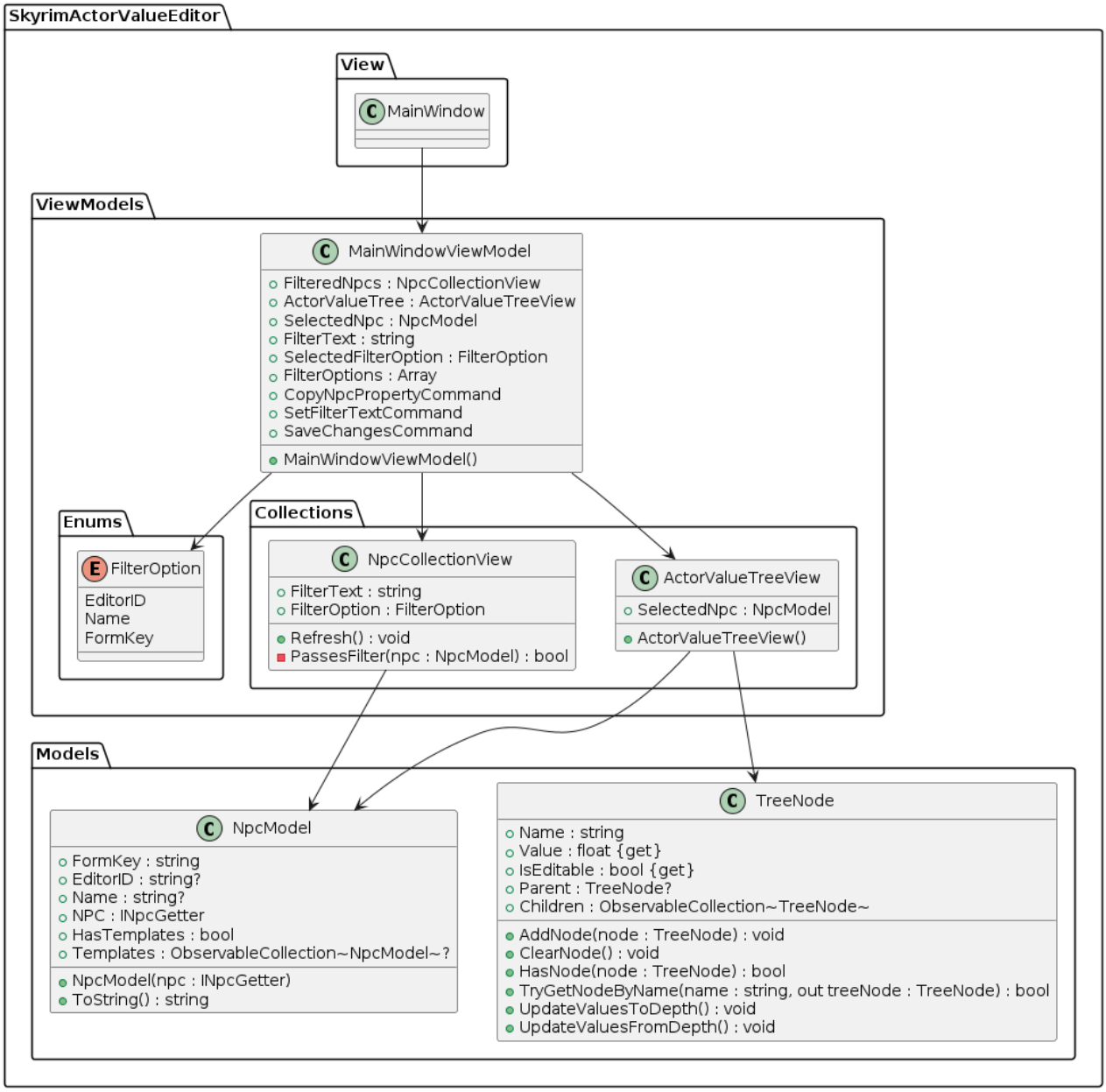


Рис. 2.1 Діаграма класів

2.4.4 NodeBuilders

NodeBuilders — це система побудови ієрархічних дерев значень акторів. Кожен побудовник відповідає за конкретний тип даних:

- ActorBaseNodeBuilder формує базові характеристики NPC.
- PerkNodeBuilder створює вузли з інформацією про перки.
- SpellNodeBuilder додає інформацію про закляття.
- Інші побудовники обробляють броню, расові характеристики, умови, магічні ефекти та скрипти.

На рис. 2.2 наведена діаграма класів, що ілюструє систему побудови вузлів, а на рис. 2.3 — діаграма класів з ієрархією типів вузлів.

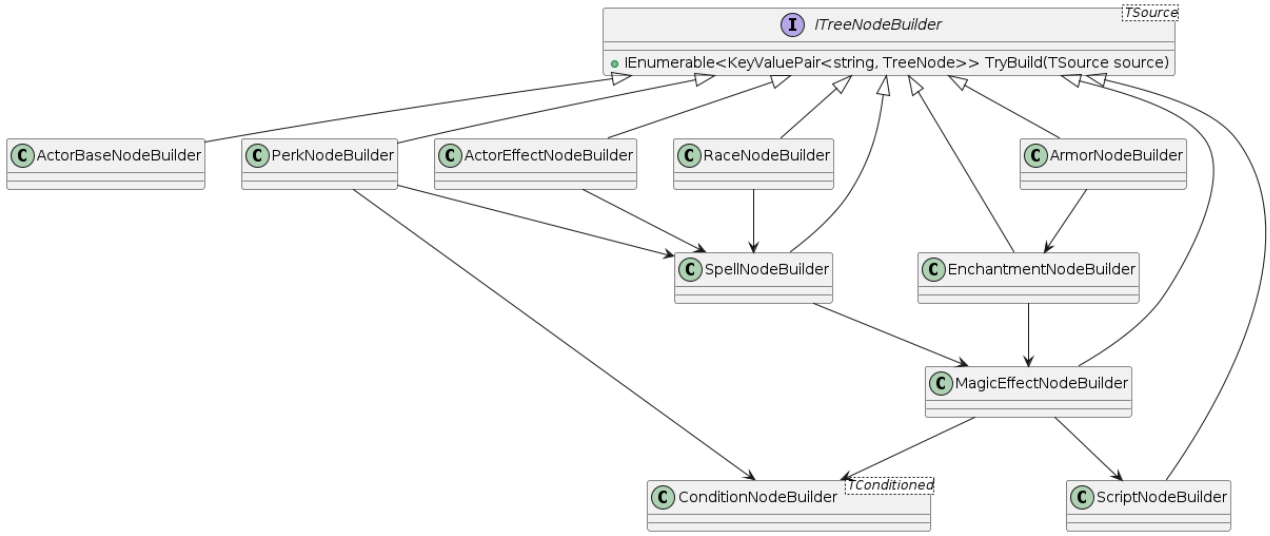


Рис. 2.2 Система побудови вузлів

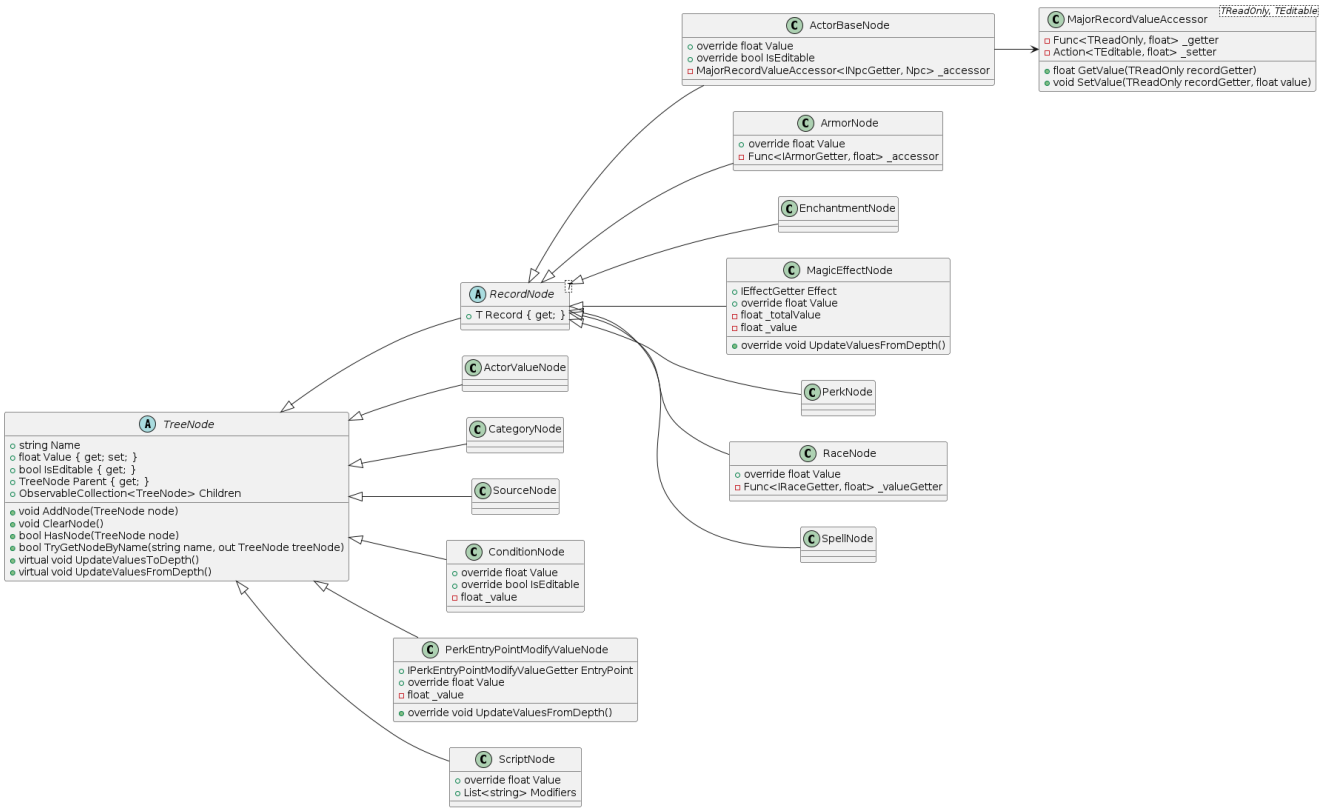


Рис. 2.3 Ієрархія вузлів

Ця архітектура дозволяє розширювати програму додаванням нових типів побудовників без суттєвих змін в основній логіці.

2.4.5 Сервіси

Сервіси реалізують функціонал роботи із зовнішніми даними та конфігураціями:

– GameContext — основний сервіс для завантаження модифікацій, доступу до записів та збереження змін.

– NpcService — відповідає за завантаження і кешування NPC, а також обробку спадковості властивостей.

– ConfigService — керує налаштуваннями програми, зокрема параметрами відлагодження.

– ScriptService — завантажує інформацію про скрипти з YAML-файлів.

Приклад одного із сервісів зображено на рис. 2.4.

```
Ссылок: 0
static GameContext()
{
    if (!GameLocations.TryGetDataFolder(GameRelease.SkyrimSE, out var pathSkyrimData))
        throw new Exception("Skyrim SE is not installed.");

    _skyrimDataPath = pathSkyrimData;
    _outputMod = GetOrCreateMod(_modName);
    _outputModPath = System.IO.Path.Combine(_skyrimDataPath, _modName);

    _environment = GameEnvironment.Typical.Builder<ISkyrimMod, ISkyrimModGetter>(GameRelease.SkyrimSE)
        .TransformLoadOrderListings(mods => mods.Where(mod => mod.Enabled))
        .WithOutputMod(_outputMod)
        .Build();

    _linkCache = _environment.LinkCache;
}

Ссылок: 19
public static bool TryResolve<TExpected>(IFormLinkGetter<TExpected> link, [MaybeNullWhen(false)] out TExpected record)
    where TExpected : class, ISkyrimMajorRecordGetter
{
    return _linkCache.TryResolve(link, out record);
}

Ссылок: 1
public static TRecord GetOriginalOrOverride<TRecord>(TRecord originalRecord)
    where TRecord : class, ISkyrimMajorRecordGetter
{
    return _outputMod.EnumerateMajorRecords<TRecord>()
            .FirstOrDefault(r => r.FormKey == originalRecord.FormKey)
        ?? originalRecord;
}

Ссылок: 1
public static TEditable GetAsOverride<TReadOnly, TEditable>(TReadOnly record)
    where TReadOnly : class, ISkyrimMajorRecordGetter
    where TEditable : class, ISkyrimMajorRecord, TReadOnly
{
    return _outputMod.GetTopLevelGroup<TEditable>()
        .GetOrAddAsOverride(record);
}
```

Рис. 2.4 Сервіс для завантаження ігрових даних

Використання сервісів дозволяє ізолювати логіку роботи з файлами і конфігураціями від інших частин застосунку.

2.5 Особливості реалізації пошуку, сортування і фільтрації NPC

Пошук, сортування та фільтрація NPC є одними з ключових функцій інтерфейсу, що дозволяють користувачу ефективно знаходити необхідних персонажів та аналізувати їх характеристики. Ці функції реалізовані на рівні ViewModel, використовуючи можливості WPF та C#.

2.5.1 Пошук NPC

Пошук реалізований через механізм фільтрації колекції NPC, представленої у вигляді ObservableCollection. Для ефективного пошуку використовуються наступні параметри:

- FormKey — унікальний ключ запису в ігрових файлах.
- EditorID — унікальний ідентифікатор NPC.
- Ім’я персонажа — пошук за текстовим іменем.

Користувач вводить текст у пошуковий рядок, після чого відбувається автоматична фільтрація списку NPC, що відображається у інтерфейсі. Див. рис. 2.5

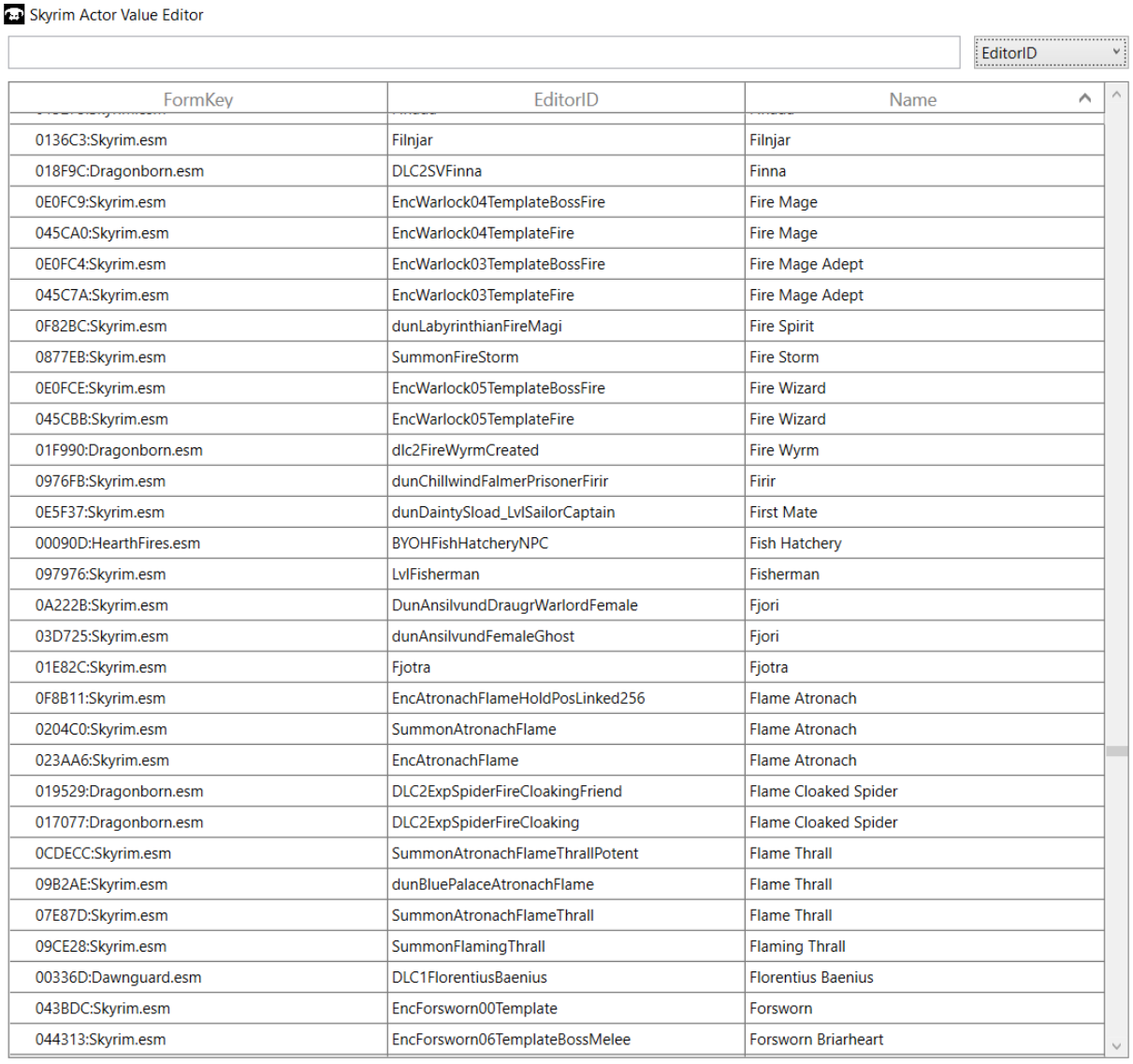
Рис. 2.5 Поле пошуку та фільтрації

2.5.2 Сортування NPC

Сортування списку NPC реалізоване через стандартні механізми WPF — CollectionView, які підтримують сортування за різними властивостями. Застосунок дозволяє сортувати NPC за:

- Ім’ям
- EditorID
- FormKey

Сортування можна змінювати динамічно, що підвищує зручність навігації по великому списку персонажів. Приклад списку відсортованих NPC наведено на рис. 2.6.



FormKey	EditorID	Name
0136C3:Skyrim.esm	Filnjar	Filnjar
018F9C:Dragonborn.esm	DLC2SVFinna	Finna
0E0FC9:Skyrim.esm	EncWarlock04TemplateBossFire	Fire Mage
045CA0:Skyrim.esm	EncWarlock04TemplateFire	Fire Mage
0E0FC4:Skyrim.esm	EncWarlock03TemplateBossFire	Fire Mage Adept
045C7A:Skyrim.esm	EncWarlock03TemplateFire	Fire Mage Adept
0F82BC:Skyrim.esm	dunLabyrinthianFireMagi	Fire Spirit
0877EB:Skyrim.esm	SummonFireStorm	Fire Storm
0E0FCE:Skyrim.esm	EncWarlock05TemplateBossFire	Fire Wizard
045CBB:Skyrim.esm	EncWarlock05TemplateFire	Fire Wizard
01F990:Dragonborn.esm	dlc2FireWyrmCreated	Fire Wyrm
0976FB:Skyrim.esm	dunChillwindFalmerPrisonerFirir	Firir
0E5F37:Skyrim.esm	dunDaintySload_LvlSailorCaptain	First Mate
00090D:HearthFires.esm	BYOHFishHatcheryNPC	Fish Hatchery
097976:Skyrim.esm	LvlFisherman	Fisherman
0A222B:Skyrim.esm	DunAnsilvundDraugrWarlordFemale	Fjori
03D725:Skyrim.esm	dunAnsilvundFemaleGhost	Fjori
01E82C:Skyrim.esm	Fjotra	Fjotra
0F8B11:Skyrim.esm	EncAtronachFlameHoldPosLinked256	Flame Atronach
0204C0:Skyrim.esm	SummonAtronachFlame	Flame Atronach
023AA6:Skyrim.esm	EncAtronachFlame	Flame Atronach
019529:Dragonborn.esm	DLC2ExpSpiderFireCloakingFriend	Flame Cloaked Spider
017077:Dragonborn.esm	DLC2ExpSpiderFireCloaking	Flame Cloaked Spider
0CDECC:Skyrim.esm	SummonAtronachFlameThrallPotent	Flame Thrall
09B2AE:Skyrim.esm	dunBluePalaceAtronachFlame	Flame Thrall
07E87D:Skyrim.esm	SummonAtronachFlameThrall	Flame Thrall
09CE28:Skyrim.esm	SummonFlamingThrall	Flaming Thrall
00336D:Dawnguard.esm	DLC1FlorentiusBaenius	Florentius Baenius
043BDC:Skyrim.esm	EncForsworn00Template	Forsworn
044313:Skyrim.esm	EncForsworn06TemplateBossMelee	Forsworn Briarheart

Рис. 2.6 Список NPC

2.5.3 Технічні деталі реалізації

В основі пошуку і фільтрації лежить механізм CollectionView, що підтримує інтерфейс ICollectionView. Це дозволяє застосовувати фільтрувальні функції, які викликаються при кожній зміні вхідних даних або параметрів пошуку.

Для збереження продуктивності при великій кількості NPC, фільтрування відбувається за допомогою делегатів та виконується у фоновому потоці із застосуванням асинхронних методів. Це запобігає затримкам у роботі інтерфейсу.

2.5.4 Інтерфейс користувача для пошуку і фільтрації

У UI реалізовані:

- Текстове поле для введення пошукових запитів.
- Випадаючі меню та чекбокси для вибору параметрів фільтрації.
- Автоматичне оновлення списку NPC у відповідь на зміну параметрів.
- Цей підхід забезпечує користувачам інтуїтивно зрозумілий і зручний інструмент для швидкого знаходження потрібних даних.

2.6 Робота з редагуванням Actor Values та збереженням змін

Редагування характеристик акторів (Actor Values) у Skyrim Actor Value Editor реалізовано як допоміжний функціонал, який дозволяє користувачеві ознайомитися із можливими значеннями параметрів NPC та у певній мірі змінювати їх для попереднього аналізу. Програма не орієнтована на глибоке редагування або комплексне управління ігровими даними, а скоріше забезпечує зручний інтерфейс для візуалізації та базових корекцій.

2.6.1 Концептуальна роль редагування

Основна мета реалізації можливості редагування полягає у наданні користувачу простого інструменту для корекції окремих числових значень, що відображаються в дереві характеристик NPC. Це дозволяє:

- Швидко перевірити, як певні зміни вплинуть на загальну картину.
- Провести мінімальні коригування без необхідності працювати з складними модінговими програмами.
- Візуально оцінити структуру впливів на характеристики NPC, включно з перками, бронею, расовими та іншими ефектами.

При цьому програма не призначена для глибинного редагування ігрових файлів або управління великою кількістю даних.

2.6.2 Технічна реалізація редагування

Редагування базується на моделі MVVM, де кожен вузол дерева значень акторів представлений окремим ViewModel-об’єктом. Ці об’єкти мають властивості, які відображають конкретні характеристики, і можуть бути відредаговані користувачем за допомогою інтерфейсу.

Однак можливості редагування обмежені:

- Підтримується лише зміна числових значень, що пов’язані з базовими параметрами NPC (здоров’я, витривалість, сила тощо).
- Не підтримується редагування складних або умовних значень, що залежать від багатьох факторів (наприклад, складні перки або заклинання).

Валідація введених даних здійснюється мінімально — система не перевіряє коректність або логічність змін, що може призвести до потенційних некоректних значень.

2.6.3 Взаємодія користувача із даними

Користувач може обирати конкретного NPC у списку, переглядати його характеристики, джерела та редагувати доступні поля (див. рис. 2.7 – 2.10). При внесенні зміни відповідний ViewModel оновлює локальне представлення даних, що одразу відображається в UI.

Варто відзначити, що редагування є одноразовим і локальним — внесені зміни не застосовуються автоматично до всіх пов’язаних записів або до шаблонів NPC, з яких можуть наслідуватись властивості. Це зумовлює, що програма більше орієнтована на демонстрацію змін, а не на їх комплексне розповсюдження.

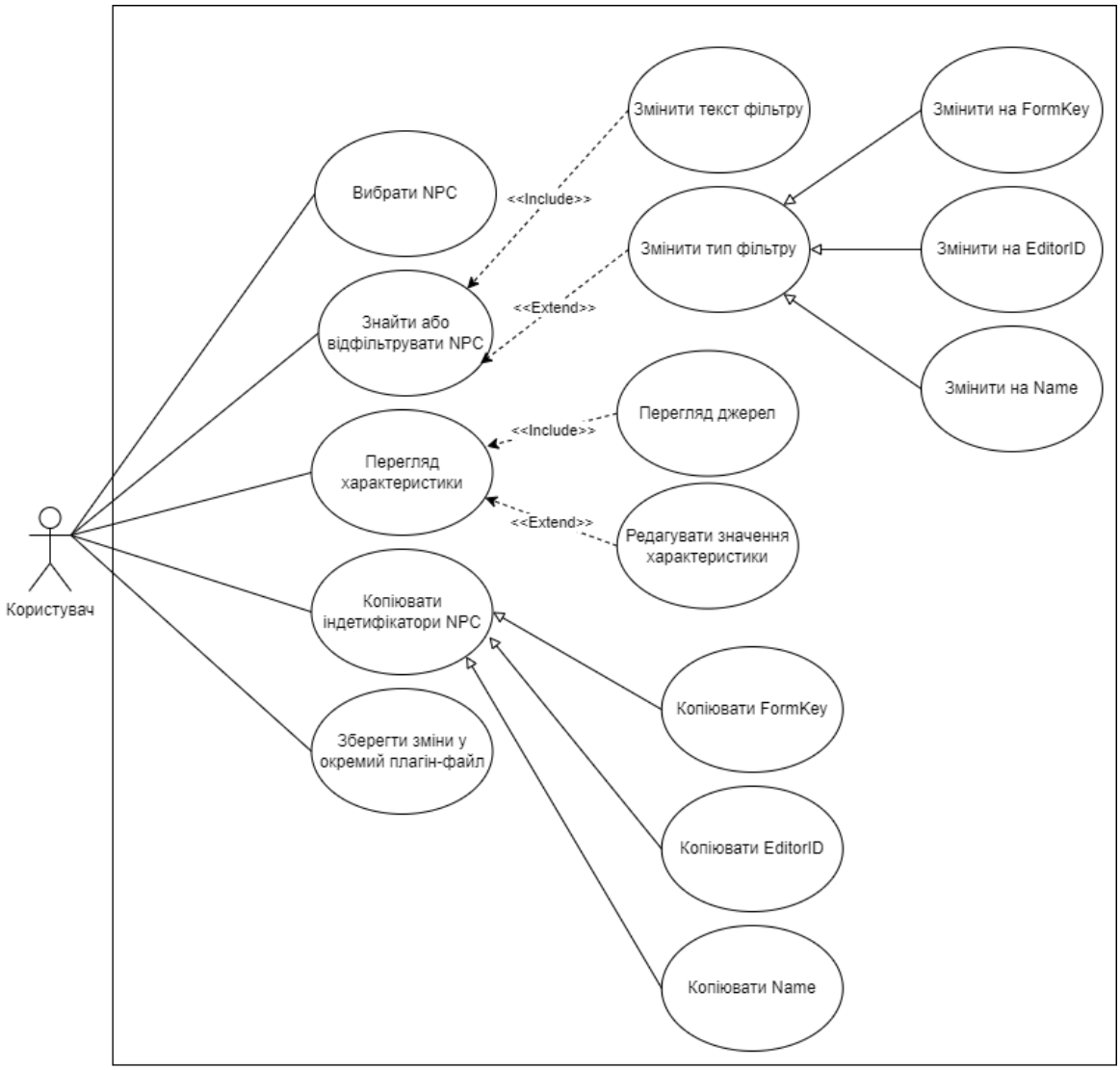
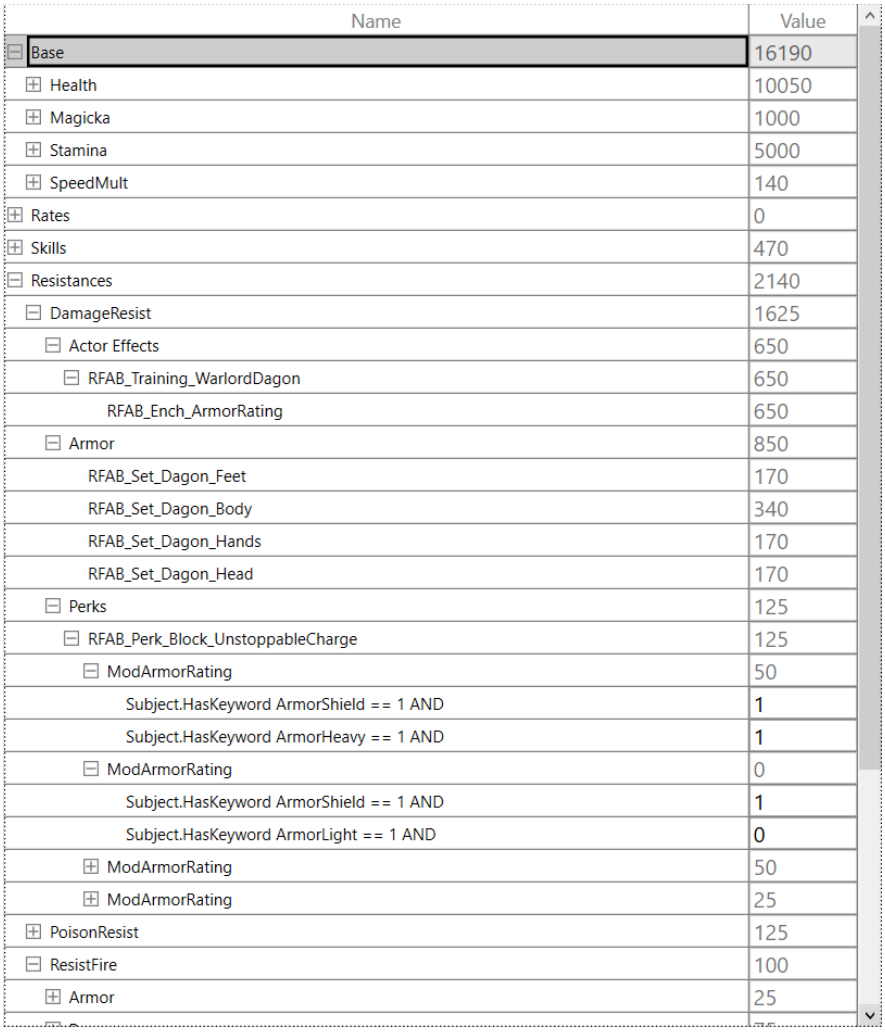
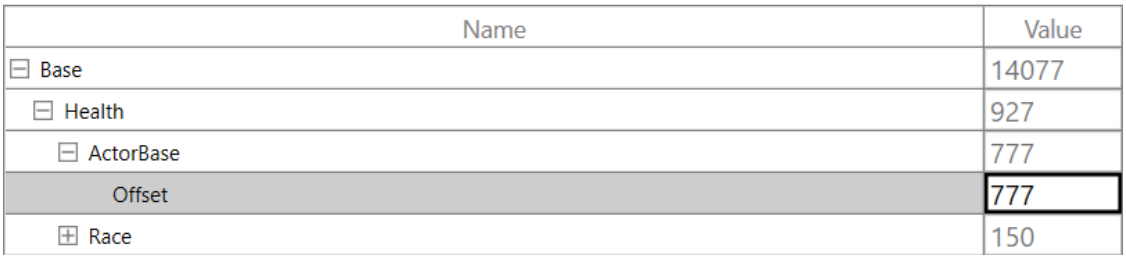


Рис. 2.7 Діаграма варіантів використання



Name	Value
Base	16190
Health	10050
Magicka	1000
Stamina	5000
SpeedMult	140
Rates	0
Skills	470
Resistances	2140
DamageResist	1625
Actor Effects	650
RFAB_Training_WarlordDagon	650
RFAB_Ench_ArmorRating	650
Armor	850
RFAB_Set_Dagon_Feet	170
RFAB_Set_Dagon_Body	340
RFAB_Set_Dagon_Hands	170
RFAB_Set_Dagon_Head	170
Perks	125
RFAB_Perk_Block_UnstoppableCharge	125
ModArmorRating	50
Subject.HasKeyword ArmorShield == 1 AND	1
Subject.HasKeyword ArmorHeavy == 1 AND	1
ModArmorRating	0
Subject.HasKeyword ArmorShield == 1 AND	1
Subject.HasKeyword ArmorLight == 1 AND	0
ModArmorRating	50
ModArmorRating	25
PoisonResist	125
ResistFire	100
Armor	25

Рис. 2.8 Таблиця характеристик, їх джерел та умов



Name	Value
Base	14077
Health	927
ActorBase	777
Offset	777
Race	150

Рис. 2.9 Редагування характеристик

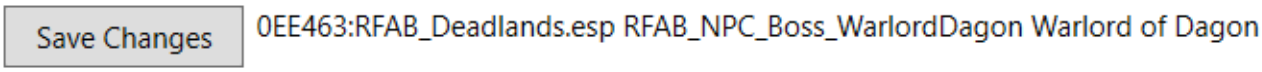


Рис. 2.10 Кнопка збереження змін та ідентифікатори NPC

2.6.4 Обмеження та особливості

Примітивність редагування пов’язана із декількома факторами:

- Відсутність складної логіки обробки залежностей і спадковості змін.
- Обмежена кількість типів підтримуваних для редагування значень.
- Відсутність підтримки масового редагування або батчових операцій.
- Мінімальна підтримка перевірок та повідомлень про помилки під час введення.

Це означає, що користувачеві слід бути обережним і усвідомлювати, що зміни є попередніми і можуть вимагати подальшої обробки у більш спеціалізованих інструментах.

2.6.5 Збереження змін

Для збереження внесених змін програма реалізує експорт у формат ESP — спеціальний формат модифікації для Skyrim Special Edition. Процес збереження включає:

- Формування нової або оновлення існуючої структури файлу.
- Запис змін лише тих значень, які були змінені у програмі.
- Збереження у окремий ESP-файл, щоб не впливати безпосередньо на оригінальні ігрові файли.

Незважаючи на це, збереження є базовим і не підтримує складні сценарії, такі як управління взаємозалежностями між записами або конфліктами між модами.

2.6.6 Роль редагування у загальній архітектурі

Редагування у Skyrim Actor Value Editor — це допоміжний інструмент для аналізу і швидких локальних корекцій, що забезпечує гнучкість у роботі з даними без складних операцій. Ця функція суттєво підвищує зручність користування програмою, не ускладнюючи при цьому її структуру та логіку.

2.7 Використання YAML для зберігання конфігурацій і налаштувань

2.7.1 Роль YAML у проекті

У Skyrim Actor Value Editor формат YAML використовується для зберігання конфігураційних параметрів, що визначають поведінку та налаштування програми. Відокремлення конфігурації від коду дозволяє гнучко змінювати налаштування без потреби змінювати та перекомпілювати сам застосунок. До таких налаштувань відносяться фільтри для списку NPC, параметри сортування, налаштування UI, а також параметри для налагодження і відлагоджувальної консолі.

2.7.2 Структура YAML-файлів

Конфігураційні файли YAML організовані у вигляді ієрархічних структур зі словників та списків, що відповідають класам налаштувань у програмі. Наприклад, фільтри NPC представлені у вигляді набору ключ-значення:

```
- Name: Skills
  Children:
    - Name: Warrior
      Children:
        - Name: OneHanded
        - Name: TwoHanded
        - Name: Archery
        - Name: Block
        - Name: Smithing
        - Name: HeavyArmor
    - Name: Mage
      Children:
        - Name: Alteration
        - Name: Conjuration
        - Name: Destruction
        - Name: Illusion
        - Name: Restoration
        - Name: Enchanting
    - Name: Thief
      Children:
        - Name: LightArmor
        - Name: Pickpocket
        - Name: Lockpicking
        - Name: Sneak
        - Name: Alchemy
        - Name: Speech
```

```
- Name: Base
  Children:
    - Name: Health
    - Name: Magicka
    - Name: Stamina
    - Name: SpeedMult

- Name: Rates
  Children:
    - Name: HealRate
    - Name: MagickaRate
    - Name: StaminaRate
    - Name: HealRateMult
    - Name: MagickaRateMult
    - Name: StaminaRateMult
```

Рис. 2.11 Приклади Yaml файлу

Ця структура дозволяє зберігати набір умов, що застосовуються для фільтрації NPC за різними параметрами.

2.7.3 Технічна реалізація роботи з YAML

Для обробки YAML у проекті застосовується бібліотека YamlDotNet, яка забезпечує механізми серіалізації та десеріалізації даних між YAML-файлами і C#-класами. При старті застосунку відповідні YAML-файли зчитуються і перетворюються у відповідні об’єкти, які далі використовуються для налаштування поведінки програми.

Серійна десеріалізація забезпечує можливість простого та гнучкого оновлення конфігурацій без необхідності зміни вихідного коду, що є важливим для підтримки і розвитку програми.

2.7.4 Взаємодія YAML з іншими компонентами програми

Дані, завантажені з YAML-файлів, безпосередньо впливають на поведінку інтерфейсу користувача та логіку програми. Наприклад, налаштування фільтрів передаються до ViewModel, яка формує відфільтрований список NPC для відображення у UI. Аналогічно, параметри сортування впливають на порядок відображення елементів.

Параметри конфігурації також регулюють роботу консолі відлагодження, що дозволяє вмикати або вимикати додатковий вивід для діагностики під час розробки.

2.7.5 Переваги та обмеження використання YAML

Переваги:

- Простий та зрозумілий для користувача формат, що дозволяє легко редагувати файли вручну.
- Гнучкість у збереженні складних ієрархічних структур.
- Відокремлення налаштувань від бізнес-логіки, що спрощує підтримку коду.
- Широка підтримка у .NET через стабільні бібліотеки, такі як YamlDotNet.

Обмеження:

− Формат чутливий до відступів, що вимагає акуратності при ручному редагуванні.

− Відсутність суворої типізації вимагає додаткової перевірки коректності даних після десеріалізації.

2.8 Взаємодія компонентів

Взаємодія компонентів у застосунку Skyrim Actor Value Editor побудована навколо принципів модульності, реактивного зв’язку між моделлю та представленням (MVVM), а також чітко визначеного потоку даних від джерел (ігрові модифікації) до користувацького інтерфейсу.

Робота програми починається з ініціалізації ядра застосунку — класу App.xaml.cs, який відповідає за реєстрацію ліцензії для UI-компонентів Syncfusion, конфігурацію внутрішніх сервісів та підключення до ігрових даних. Основна взаємодія між частинами програми далі організовується навколо класів GameContext, NpcService, MainWindowViewModel, а також модулів побудови дерева NodeBuilder.

2.8.1 Ініціалізація: запуск та завантаження даних

Після запуску застосунку виконується завантаження налаштувань (через ConfigService) та підготовка середовища (GameContext). GameContext надає доступ до колекції всіх записів, що містяться у завантажених модифікаціях Skyrim. Зокрема, здійснюється пошук об’єктів типу Npc_, які обробляються в NpcService для побудови колекції NPC.

При цьому NpcService також виконує аналіз спадковості (через шаблони NPC) та готує уніфіковану модель NpcModel для подальшої роботи у ViewModel.

2.8.2 Передача даних до ViewModel

Завантажені моделі NPC передаються у MainWindowViewModel, де ініціалізується колекція FilteredNpcs. Вона слугує джерелом даних для відображення в UI-компоненті Syncfusion TreeGrid.

Крім того, MainWindowViewModel зберігає поточний стан дерева значень — ActorValueTree, що будується динамічно при виборі NPC.

2.8.3 Побудова дерева значень акторів

При зміні вибраного NPC користувачем, MainWindowViewModel викликає відповідні NodeBuilder-и, кожен з яких відповідає за конкретну частину ієрархії значень акторів:

- ActorBaseNodeBuilder — основні значення (здоров’я, стаміна, мана тощо)
- RaceNodeBuilder — расові бонуси
- PerkNodeBuilder, SpellNodeBuilder, ArmorNodeBuilder — відповідно перки, заклинання, броня

Усі ці побудовники працюють із даними, які надає GameContext, та формують дерево TreeNode, яке потім підключається до ActorValueTreeView у головному вікні.

2.8.4 Робота з UI та реактивність

Завдяки використанню патерну MVVM, усі компоненти UI автоматично оновлюються у відповідь на зміну ViewModel. Компонент TreeGrid, який відображає список NPC, реагує на зміну колекції FilteredNpcs. Дерево характеристик оновлюється після кожної побудови нової моделі дерева акторів.

Фільтрація, вибірка, копіювання значень — все це реалізовано через команди у MainWindowViewModel, які напряму взаємодіють з відповідними сервісами або властивостями моделі.

2.8.5 Обробка змін та збереження

У випадку, якщо користувач вносить зміни до значення певного параметру, ці зміни відображаються безпосередньо в ієрархії TreeNode, після чого GameContext бере на себе відповідальність за оновлення внутрішньої моделі та підготовку ESP-файлу для збереження.

Цей процес є односпрямованим: від користувача до дерева, далі — до моделі, і в результаті — до вихідного ESP-файлу. Зворотного автоматичного оновлення з ESP не передбачено, тому збереження змін є завершальним кроком роботи з даними у рамках сесії.

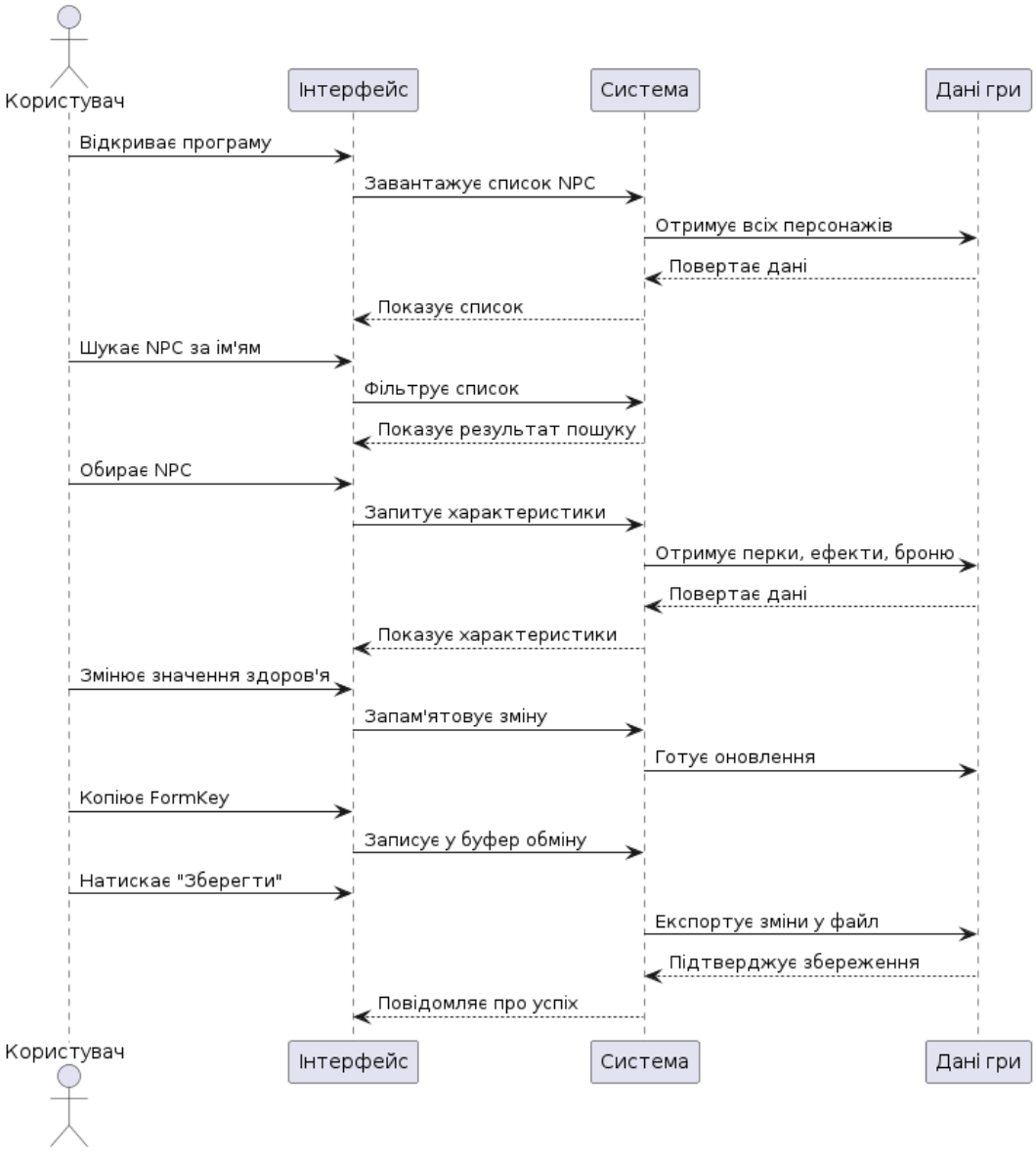


Рис. 2.12 Діаграма послідовності взаємодії користувача із застосунком

3 ТЕСТУВАННЯ ТА ОПТИМІЗАЦІЯ ЗАСТОСУНКУ

3.1 Підходи до тестування застосунку

Тестування програмного забезпечення Skyrim Actor Value Editor здійснювалося на декількох рівнях: ручне функціональне тестування, інтеграційне тестування ключових компонентів, а також неформальне профілювання продуктивності.

Основна увага приділялася перевірці:

- коректності завантаження модифікацій (ESP/ESM/ESL),
- правильності побудови дерева Actor Values,
- коректної обробки фільтрації та вибору NPC,
- збереження змін без порушення структури ігрових файлів.

Оскільки проєкт має відносно невелику складність та високий ступінь залежності від сторонніх бібліотек (Mutagen, Syncfusion), тестування зосереджено переважно на сценаріях взаємодії користувача з інтерфейсом.

3.1.1 Ручне тестування

Застосунок проходив серії ручних тестів На ігровому модульному наборі з понад 500 NPC (наприклад, з модами типу “Requiem”) з використанням ESP-файлів при різних налаштуваннях, зокрема вмиканні/вимиканні режиму налагодження (debug console). Результати тестування наведено у таблиці 3.1.

Таблиця 3.1

Тестування розробленного програмного забезпечення

Дія користувача	Очікуваний результат	Статус
Введення частини імені у фільтр	Після натискання Enter NPC фільтруються	Працює
Вибір NPC у списку	Будується дерево Actor Values з урахуванням шаблонів	Працює
Зміна значення у дереві	Значення змінюється і зберігається	Працює

3.1.2 Випробування на помилки

У процесі розробки Skyrim Actor Value Editor було враховано основні типи помилок, які можуть виникати під час роботи з ігровими файлами або при взаємодії з користувачем:

- Під час завантаження .esp/.esm/.esl-файлів можуть виникати винятки через пошкоджені або неігрові файли. У таких випадках бібліотека Mutagen.Bethesda генерує помилки, які перехоплюються додатком з виведенням відповідного повідомлення.
- При побудові дерева властивостей NPC можливі збої, якщо в даних відсутні Spell, Perk або Race. Для цього в NodeBuilder реалізовані перевірки на null і коректність структури записів.

3.2 Оптимізація продуктивності

3.2.1 Відкладене виконання фільтрації

Оскільки кількість NPC у великих модпаків може досягати кількох тисяч, було прийнято рішення не оновлювати результати пошуку при кожному введенні символу. Замість цього, фільтрація застосовується лише після натискання клавіші Enter. Це дозволяє уникнути надлишкових обчислень і зберігати стабільну швидкодію навіть при активному наборі тексту.

Результат: зменшено кількість оновлень колекції в десятки разів у порівнянні з "живим" пошуком.

3.2.2 Ліниве завантаження шаблонів NPC

Багато NPC у Skyrim використовують механізм наслідування властивостей через шаблони (Template). Завантаження цих шаблонів потребує рекурсивного аналізу та побудови ієрархії. Щоб не перевантажувати систему при початковому відображенні списку NPC, ці шаблони завантажуються лише під час першого вибору конкретного персонажа користувачем.

Результат: суттєво зменшено час початкового завантаження програми та зменшено об’єм тимчасових об'єктів у пам’яті.

3.2.3 Використання ефективних бібліотек

Програма заснована на бібліотеці Mutagen.Bethesda, яка реалізує роботу з ESP/ESM-файлами максимально ефективно завдяки прямому доступу до структур через слабооб'єктну модель. Також застосунок використовує потужну UI-бібліотеку Syncfusion, що забезпечує віртуалізацію відображення великих колекцій у TreeGrid.

Результат: навіть на комп’ютерах середнього рівня програма запускається менш ніж за 5 секунд та забезпечує плавну взаємодію користувача з деревом Actor Values.

3.3 Обмеження та потенційні вузькі місця

− Відсутність справжньої паралельної обробки при завантаженні — усе виконується синхронно (можливо, з блокуванням UI).

− Завантаження всіх NPC при старті, навіть якщо частина з них не буде використана.

− Немає агресивного кешування результатів побудови дерев — вони перегенеруються при повторному виборі.

3.4 Плани щодо покращення оптимізації

У наступних версіях програми можна передбачити:

− Асинхронне завантаження NPC та побудову дерева через Task/async-await.

− Додавання механізмів кешування побудованих дерев NPC.

− Можливість попереднього фільтрування модів за джерелом (наприклад, лише оригінальні або лише модифіковані).

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи було досягнуто основної мети — створено спеціалізоване програмне забезпечення Skyrim Actor Value Editor, призначене для аналізу та базового редагування характеристик (Actor Values) неігрових персонажів (NPC) у грі The Elder Scrolls V: Skyrim Special Edition.

У межах поставлених задач:

1. Проаналізовано структуру ігрових даних Skyrim SE, зокрема формат ESP/ESM/ESL-файлів і механізми спадкування властивостей NPC через шаблони, раси, закляття та перки.

2. Досліджено наявні інструменти модифікації гри (Creation Kit, SSEEdit) та встановлено їхні обмеження щодо зручності аналізу Actor Values. Це стало обґрунтуванням потреби у власному інструменті.

3. Визначено вимоги до застосунку, включаючи підтримку гнучкої побудови ієрархії, зручну навігацію, редагування та інтеграцію з Mutagen.Bethesda.

4. Розроблено архітектуру програми на основі патерну MVVM, що забезпечило відокремлення логіки, представлення та моделей. Застосунок реалізовано з використанням технологій .NET та WPF.

5. Імплементовано механізм побудови дерева Actor Values для NPC, з урахуванням спадкування та типізації вузлів (Spell, Perk, Race тощо). Для інтерактивної візуалізації використано компонент Syncfusion TreeGrid.

6. Реалізовано базову функціональність редагування Actor Values без порушення структур ESP-файлів, що дозволяє користувачу змінювати характеристики NPC без ризику втрати даних.

7. Проведено тестування програми на великій вибірці NPC (500+ записів), включаючи складні ієрархічні конфігурації. Застосунок продемонстрував стабільну роботу, високу швидкодію і відсутність критичних помилок.

Таким чином, усі поставлені задачі виконано повністю. Розроблений інструмент забезпечує новий рівень зручності для розробників модів, дозволяючи швидко аналізувати структуру Actor Values та виявляти джерела тих чи інших характеристик персонажів. Це значно скорочує час, необхідний для внесення змін у поведінку NPC, та усуває потребу в ручному аналізі численних ігрових записів.

Робота пройшла апробацію на наукових конференціях, за результатами апробації опубліковано тези доповідей:

1. Іванов М.Є., Золотухіна О.А. Визначення вимог до програмного засобу для відстеження і редагування характеристик NPC у відеогрі TES V: Skyrim // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2025, ДУІКТ, Київ, Україна, с. 226-227.
2. Іванов М.Є. Захист інформації у застосунку Skyrim Actor Values Editor // XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології - 2025». Збірник тез. 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 282 - 283.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microsoft .NET Documentation. URL: https://learn.microsoft.com/en-us/dotnet/
2. C# Programming Guide Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/csharp/
3. Nexus mods Skyrim Special Edition URL: https://www.nexusmods.com/games/skyrimspecialedition
4. Microsoft C# Language Reference URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/
5. WPF GitHub Repository. URL: https://github.com/dotnet/wpf
6. Mutagen Modding Framework Documentation. URL: https://mutagen-modding.github.io/Mutagen/#goals
7. Mutagen GitHub Repository. URL: https://github.com/Mutagen-Modding/Mutagen
8. zEdit Documentation: xEdit Comparison. URL: https://z-edit.github.io/#/docs?t=Overview
9. TES5Edit Official Documentation. URL: https://tes5edit.github.io/docs/
10. Syncfusion TreeGrid Component. URL: https://www.syncfusion.com/aspnet-core-ui-controls
11. YamlDotNet GitHub Repository. URL: https://github.com/aaubry/YamlDotNet
12. Unofficial Elder Scrolls Pages. Creation Kit. URL: https://ck.uesp.net/wiki/Creation_Kit
13. Unofficial Elder Scrolls Pages. Spell Mechanics. URL: https://ck.uesp.net/wiki/Spell
14. Unofficial Elder Scrolls Pages. Perk System. URL: https://ck.uesp.net/wiki/Perk
15. Unofficial Elder Scrolls Pages. Race Attributes. URL: https://ck.uesp.net/wiki/Race
16. Unofficial Elder Scrolls Pages. Actor Traits. URL: https://ck.uesp.net/wiki/Category:Actor
17. Unofficial Elder Scrolls Pages. Enchantments. URL: https://ck.uesp.net/wiki/Enchantment

18. Unofficial Elder Scrolls Pages. Magic Effects. URL: https://ck.uesp.net/wiki/Magic_Effect
19. Unofficial Elder Scrolls Pages. Actor Values. URL: https://ck.uesp.net/wiki/Actor_Value
20. Nexus Mods Skyrim Special Edition. URL: https://www.nexusmods.com/skyrimspecialedition/
21. Іванов М.Є., Золотухіна О.А. Визначення вимог до програмного засобу для відстеження і редагування характеристик NPC у відеогрі TES V: Skyrim // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2025, ДУІКТ, Київ, Україна, с. 226-227.
22. Іванов М.Є. Захист інформації у застосунку Skyrim Actor Values Editor // XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології - 2025». Збірник тез. 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 282 - 283.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмних засобів для відстеження і редагування характеристик NPC на прикладі гри Skyrim

Виконав студент 4 курсу
групи ПД-43
Іванов Михайло Євгенович
Керівник роботи
доцент кафедри ІПЗ, кандидат технічних наук, доцент Золотухіна Оксана Анатоліївна

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесів відстеження і редагування характеристик NPC у відеогрі Skyrim.

Об'єкт дослідження: процес відстеження і редагування характеристик NPC у рольових іграх.

Предмет дослідження: засоби та технології для обробки ігрових даних, зокрема з використанням мови програмування C#, платформи WPF та бібліотеки Mutagen.Bethesda.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести дослідження технічної специфікації гри The Elder Scrolls V: Skyrim, зокрема, дослідити структуру зберігання та обробки ігрових даних.
2. Провести аналіз існуючих інструментів для модифікації ігрових даних, таких як SSEEdit та Creation Kit.
3. Визначити перелік функціональних та нефункціональних вимог до програмних засобів для відстеження і редагування характеристик NPC, провести моделювання вимог засобами UML.
4. Виконати моделювання програмних засобів для відстеження і редагування характеристик NPC з використанням моделей UML.
5. Розробити програмні засоби для відстеження і редагування характеристик NPC з урахуванням джерел їх формування.
6. Провести тестування програми.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Creation kit	SSEEdit	Skyrim Actor Value Editor
Аналіз характеристик NPC	Відкриття 5+ вкладок, ручний пошук серед непотрібних даних	Ручне порівняння записів у різних списках	Автоматичне відображення всіх характеристик у єдиному вікні
Інтерфейс	80% непотрібних інструментів (рендер, квести)	50% технічних списків (FormID, конфлікти)	Лише необхідні дані для роботи з AV
Час аналізу 1 NPC	10-15 хвилин	5-10 хвилин	1-2 хвилини

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Завантаження даних NPC з ігрових файлів Skyrim;
2. Автоматичне обчислення характеристик з урахуванням усіх джерел впливу;
3. Відображення ієрархії впливів для кожної характеристики;
4. Пошук і фільтрація NPC за критеріями;
5. Редагування окремих значень характеристик (основні атрибути та навички);
6. Збереження змін в окремому плагін-файлі.

Нефункціональні вимоги:

1. Програма має запускатися менш ніж за 10 секунд;
2. Простий інтерфейс із контекстним відображенням елементів;
3. Застосунок не повинен використовувати більше 500 МБ оперативної пам'яті;
4. Програма повинна працювати на Windows 10 і вище.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

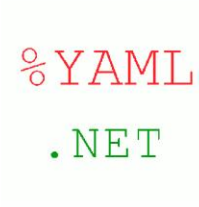
Visual Studio

WPF .NET

Mutagen.Bethesda.Skyrim

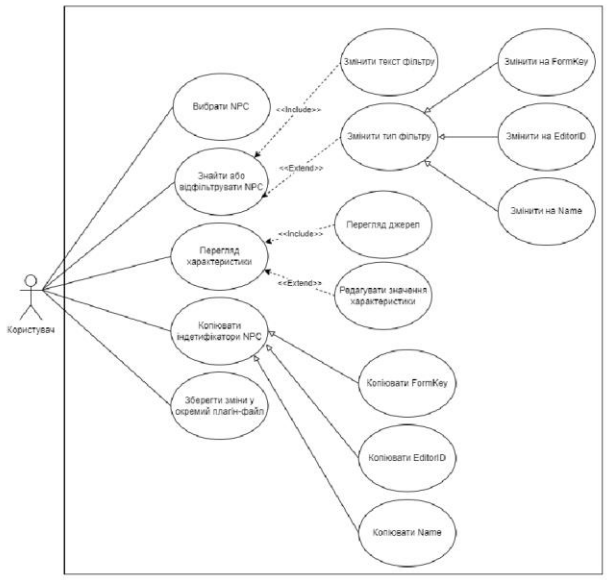
GitHub

Syncfusion



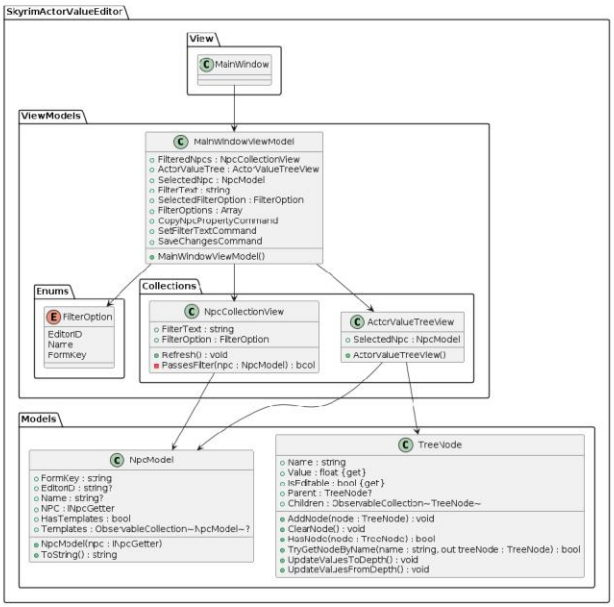
6

Діаграма варіантів використання



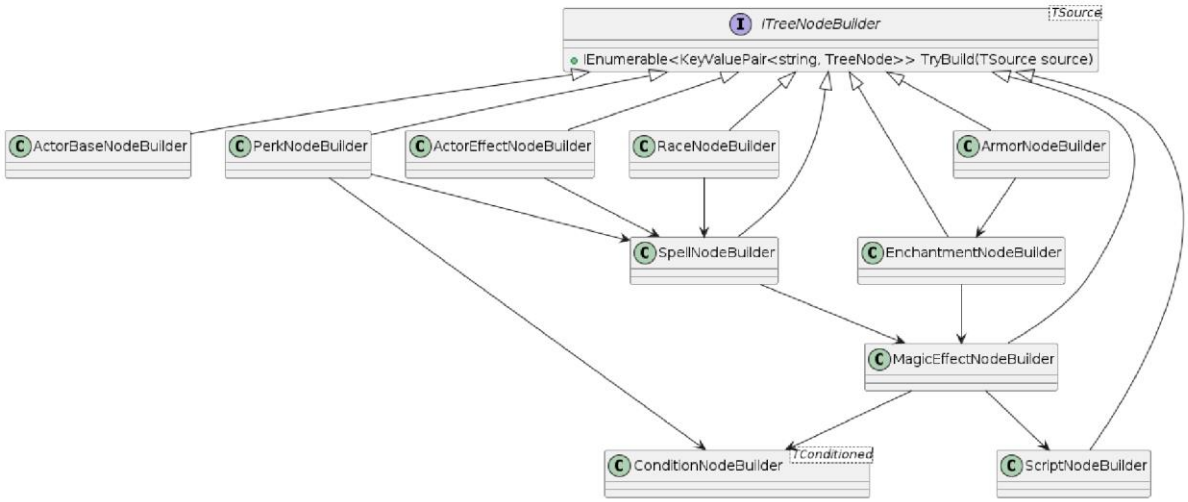
7

Архітектура програмного забезпечення



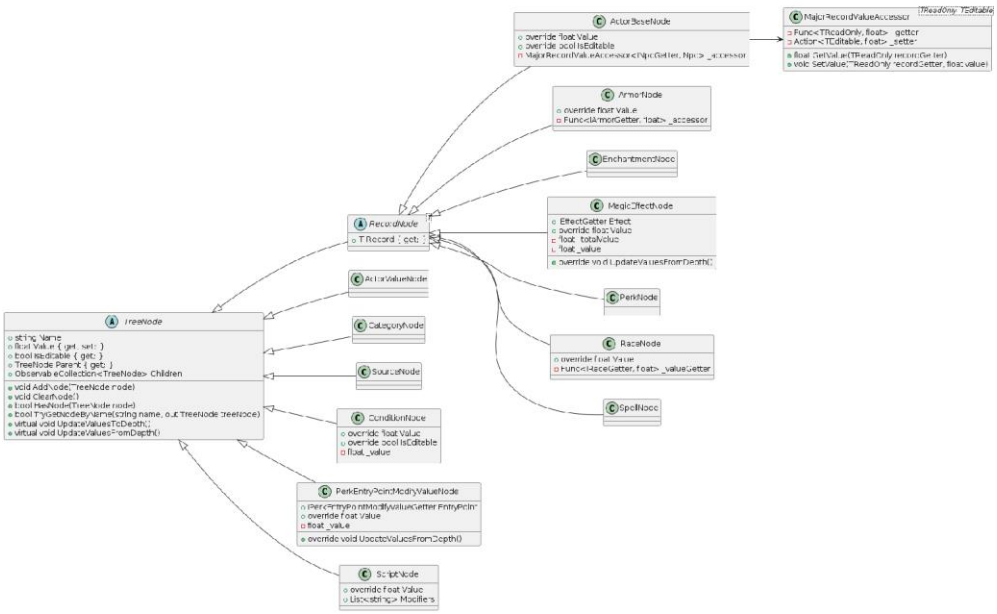
8

Система побудови вузлів



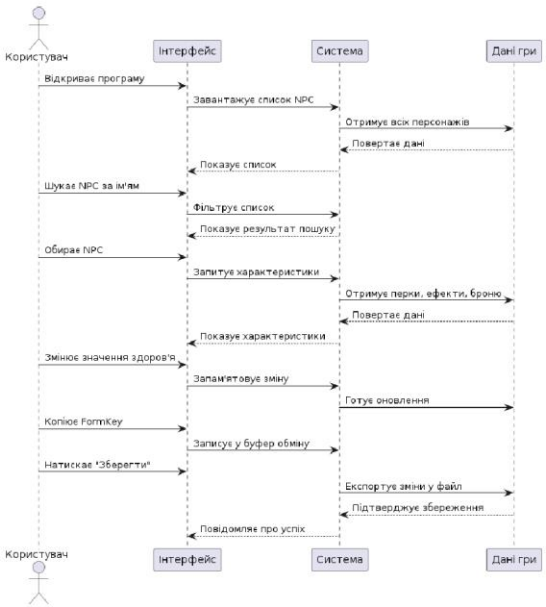
9

Ієрархія вузлів



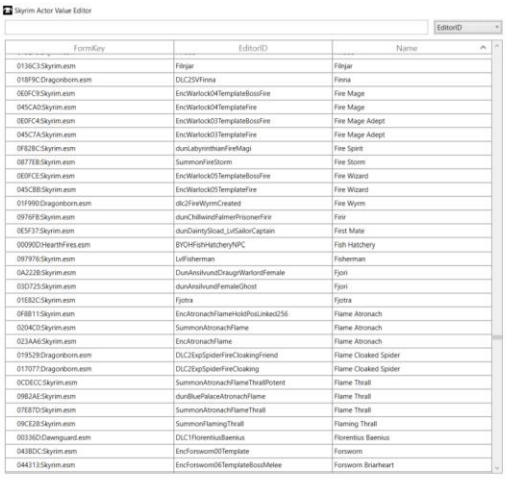
10

Діаграма послідовності взаємодії користувача із застосунком

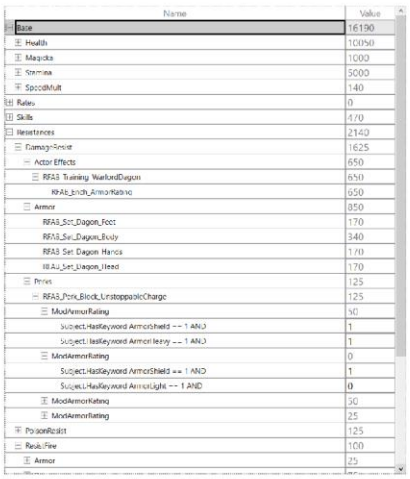


11

Екранні форми



Список NPC



Таблиця характеристик, їх джерел та умов

12

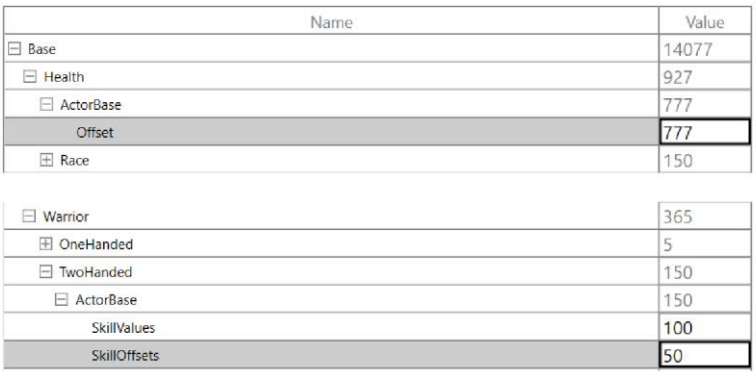
Екранні форми

Поле пошуку та фільтрації

Кнопка збереження змін та ідентифікатори NPC

13

Екранні форми



Змінення характеристик

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Іванов М.Є., Золотухіна О.А. Визначення вимог до програмного засобу для відстеження і редагування характеристик NPC у відеогрі TES V: Skyrim // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез. 24.04.2025, ДУІКТ, Київ, Україна, с. 226-227.

2. Іванов М.Є. Захист інформації у застосунку Skyrim Actor Values Editor // XII Всеукраїнська науково-практична конференція молодих учених «Інформаційні технології - 2025». Збірник тез. 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 282 - 283.

15

ВИСНОВКИ

1. Проаналізовано технічну специфікацію Skyrim та структуру зберігання ігрових даних для забезпечення коректної обробки інформації про NPC.
2. Оцінено можливості існуючих інструментів модифікації (SSEEdit, Creation Kit) з метою визначення їх обмежень у контексті аналізу Actor Values.
3. Сформульовано вимоги до застосунку для аналізу характеристик NPC та змодельовано його логіку за допомогою UML для побудови чіткої архітектури.
4. Реалізовано програмний засіб для перегляду та аналізу характеристик NPC із частковим редагуванням, орієнтований на потреби розробників модифікацій.
5. Проведено тестування функціональності програми на реальних ігрових даних, що підтвердило коректність обробки та відображення інформації без помилок і збоїв у роботі.

16

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
using SkyrimActorValueEditor.Core.Services;
using SkyrimActorValueEditor.Models.Npcs;
using SkyrimActorValueEditor.ViewModels.Base;
using
SkyrimActorValueEditor.ViewModels.Collections;
using
SkyrimActorValueEditor.ViewModels.Commands;
using SkyrimActorValueEditor.ViewModels.Enums;
using System.Windows;
using System.Windows.Input;

namespace SkyrimActorValueEditor.ViewModels
{
   public class MainWindowViewModel :
BaseViewModel
   {
      public NpcCollectionView FilteredNpcs { get; } =
new();
      public ActorValueTreeView ActorValueTree { get;
} = new();

      public NpcModel? SelectedNpc
      {
         get => ActorValueTree.SelectedNpc;
         set
         {
            ActorValueTree.SelectedNpc = value;
            OnPropertyChanged();
         }
      }

      public string FilterText
      {
         get => FilteredNpcs.FilterText;
         set => FilteredNpcs.FilterText = value;
      }
      public FilterOption SelectedFilterOption
      {
         get => FilteredNpcs.FilterOption;
         set => FilteredNpcs.FilterOption = value;
      }
      public Array FilterOptions { get; } =
Enum.GetValues(typeof(FilterOption));

      public ICommand CopyNpcPropertyCommand {
get; }
      public ICommand SetFilterTextCommand { get; }
      public ICommand SaveChangesCommand { get; }

      //public ICommand LoadTemplatesCommand { get;
}

      public MainWindowViewModel()
      {
         CopyNpcPropertyCommand = new
RelayCommand(arg =>
         {
            if (arg is string text &&
!string.IsNullOrEmpty(text))
               Clipboard.SetText(text);
         });

         SetFilterTextCommand = new
RelayCommand(arg => FilterText = arg as string ??
string.Empty);

         SaveChangesCommand = new RelayCommand(_
=> GameContext.SaveChanges());

         //LoadTemplatesCommand = new
LoadTemplatesCommand();
      }
   }
} using SkyrimActorValueEditor.ViewModels.Base;
using System.Collections.ObjectModel;
using System.Diagnostics.CodeAnalysis;
using System.Reactive.Linq;

namespace
SkyrimActorValueEditor.Models.ActorValues.Nodes.Ba
se
{
   public abstract class TreeNode : BaseViewModel
   {
      public string Name { get; }

      public virtual float Value
      {
         get => _value;
         set { }
      }

      public virtual bool IsEditable => false;

      public TreeNode? Parent { get; private set; }
      public ObservableCollection<TreeNode> Children
{ get; } = new();

      private float _value;

      protected TreeNode(string name)
      {
         Name = name;
      }

      public void AddNode(TreeNode node)
      {
         node.Parent = this;
```

```
            Children.Add(node);
        }

        public void ClearNode()
        {
            foreach (var node in Children)
            {
                node.ClearNode();
                node.Parent = null;
            }

            Children.Clear();
        }

        public bool HasNode(TreeNode node) =>
Children.Contains(node);

        public bool TryGetNodeByName(string name,
[MaybeNullWhen(false)] out TreeNode treeNode)
        {
            treeNode = Children.FirstOrDefault(x =>
x.Name.Equals(name));
            return treeNode != null;
        }

        public virtual void UpdateValuesToDepth()
        {
            foreach (var treeNode in Children)
                treeNode.UpdateValuesToDepth();

            _value = Children.Sum(x => x.Value);
            OnPropertyChanged(nameof(Value));
        }

        public virtual void UpdateValuesFromDepth()
        {
            _value = Children.Sum(x => x.Value);
            OnPropertyChanged(nameof(Value));

            Parent?.UpdateValuesFromDepth();
        }
    }
} using Mutagen.Bethesda.Skyrim;
using
SkyrimActorValueEditor.Models.ActorValues.Nodes.Ba
se;

namespace
SkyrimActorValueEditor.Models.ActorValues.Nodes.Re
cords.Base
{
    public abstract class RecordNode<T> : TreeNode
        where T : ISkyrimMajorRecordGetter
    {
        public T Record { get; }

        protected RecordNode(string name, T record) :
base(name)
        {
            Record = record;
        }

        protected RecordNode(T record) :
base(GetRecordNodeName(record))
        {
            Record = record;
        }

        private static string GetRecordNodeName(T record)
        {
            return record.EditorID ?? "UNKNOWN";
        }
    }
} using Mutagen.Bethesda.Skyrim;
using SkyrimActorValueEditor.Core.Services;
using SkyrimActorValueEditor.Core.Services.Yaml;
using
SkyrimActorValueEditor.Models.ActorValues.NodeBuil
ders.Builders;
using
SkyrimActorValueEditor.Models.ActorValues.NodeBuil
ders.Interfaces;
using
SkyrimActorValueEditor.Models.ActorValues.Nodes.Ba
se;
using
SkyrimActorValueEditor.Models.ActorValues.Nodes.Se
parate;
using SkyrimActorValueEditor.Models.Npcs;

namespace
SkyrimActorValueEditor.Models.ActorValues
{
    public static class ActorValueNodeBuilder
    {
        private static readonly Dictionary<string,
ITreeNodeBuilder<INpcGetter>> _nodeBuilders = new()
        {
            { "Actor Effects", new ActorEffectNodeBuilder()
},
            { "Armor", new ArmorNodeBuilder() },
            { "ActorBase", new ActorBaseNodeBuilder() },
            { "Race", new RaceNodeBuilder() },
            { "Perks", new PerkNodeBuilder() }
        };

        private static readonly List<TreeNode>
_categoryNodes;
        private static readonly Dictionary<string,
ActorValueNode> _actorValuesNodesDictionary;

        static ActorValueNodeBuilder()
        {
            var yamlBuilder = new YamlNodeBuilder();
            var (categoryNodes,
actorValuesNodesDictionary) =
yamlBuilder.LoadActorValues();

            _categoryNodes = categoryNodes;
            _actorValuesNodesDictionary =
actorValuesNodesDictionary;
        }
```

```
        public static IEnumerable<TreeNode>
LoadActorValues() => _categoryNodes;

        public static void RebuildActorValues(NpcModel
npcModel)
        {
            foreach (var treeNode in
_actorValuesNodesDictionary.Values)
                treeNode.ClearNode();

            foreach (var (key, treeNode) in
GetActorValues(NpcService.GetNpcWithTemplates(npc
Model)))

_actorValuesNodesDictionary[key].AddNode(treeNode);

            foreach (var treeNode in _categoryNodes)
                treeNode.UpdateValuesToDepth();
        }

        private static IEnumerable<KeyValuePair<string,
TreeNode>> GetActorValues(INpcGetter npc)
        {
            foreach (var (sourceName, nodeBuilder) in
_nodeBuilders)
            {
                foreach (var (actorValue, buildedNode) in
nodeBuilder.TryBuild(npc))
                {
                    if
(!_actorValuesNodesDictionary.TryGetValue(actorValu
e, out var actorValueNode))
                        continue;

                    if
(!actorValueNode.TryGetNodeByName(sourceName,
out var sourceNode))
                        sourceNode = new
SourceNode(sourceName);


                    sourceNode.AddNode(buildedNode);



                    if (!actorValueNode.HasNode(sourceNode))

                        yield return new(actorValue,
sourceNode);

                }

            }

        }

    }

}
```