Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження можливостей редагування байт-коду CIL з метою аналізу та модифікації алгоритмів скомпільованого програмного забезпечення

Виконав: студент 6 курсу, групи СПм-61
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

_____ Грибун І.Є.
(підпис) (прізвище та ініціали)

Керівник _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

Рецензент _____ Марценко С.В.
(підпис) (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2022 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Грибуну Ігору Євгеновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження можливостей редагування байт-коду CIL з метою аналізу та модифікації алгоритмів скомпільованого програмного забезпечення

Керівник роботи Петрик Михайло Романович, д.ф.-м.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи алгоритм модифікації байт-коду CIL, алгоритм пошуку фрагментів байт-коду, пояснювальна записка

4. Зміст роботи (перелік питань, які потрібно розробити)
аналіз предметної галузі модифікування байт-коду, обґрунтування вибору напрямку дослідження, технічні перешкоди предметної області, дослідження інструментального функціоналу, проектування архітектури ін'єктованого коду, застосування методології аналізу алгоритмів, реалізація програмної частини, тестування та оцінка якості, охорона праці та безпека в надзвичайних ситуаціях

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
спрощена UML діаграма класів проекту, UML діаграма послідовності взаємодії класів проекту, повна UML діаграма реалізованих класів проекту, UML діаграма варіантів використання кінцевої системи проекту, зображення внутрішньої структури змінюваного проекту, зображення процесу пошуку цільових фрагментів байт-коду, зображення методів реалізованого коду проекту, презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Осухівська Галина Михайлівна	30.11.22	
Безпека в надзвичайних ситуаціях	Клепчик Василь Михайлович	30.11.22	

7. Дата видачі завдання 30 листопада 2022р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	18.09.22	
2	Обґрунтування вибору напрямку дослідження	25.09.22	
3	Технічні перешкоди предметної галузі	02.10.22	
4	Основна частина	20.11.22	
5	Підготовка пояснювальної записки	23.11.22	
6	Підготовка презентації та доповіді	27.11.22	
7	Попередній захист	30.11.22	
8	Охорона праці		
9	Безпека в надзвичайних ситуаціях		
10	Нормоконтроль		
11	Рецензування		
12	Занесення диплома в електронний архів		
13	Допуск до захисту у зав. кафедри		

Студент _____ Грибун І.Є.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота магістра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення», 2022 рік. Робота містить: 77 с., 35 рис., 1 табл., презентація.

Метою даної роботи є дослідження існуючих засобів редагування скомпільованого коду, оцінка легкості та зручності їх використання та впровадження у нових проектах, а також техніко-економічне обґрунтування створення модифікацій до існуючих проектів.

Основними досліджуваними засобами є інструментарій «ILSpy» та бібліотека «Harmony», які використовуються для аналізу та модифікації скомпільованого коду програмної технології .NET Framework, яка впроваджує методи компіляції та виконання байт-коду Common Intermediate Language.

У даній роботі розглянуто методологію та практичне виконання аналізу скомпільованого коду, структуру CIL та його набір інструкцій, дослідження байт-коду існуючого проекту, ідентифікацію функціоналу у змінюваному проекті та пошук відповідного коду, написання нових алгоритмів за допомогою байт-коду без використання компілятора, та методи ін’єкції нових інструкцій у існуючий проект.

Ключові слова: аналіз коду, байт-код, бібліотека, декомпіляція, документація, інтеграція, інструкції, модифікація, операційний код, проектна документація, C Sharp, Harmony, ILSpy, .NET Framework

ABSTRACT

Master’s qualification work. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 "Software Engineering", year 2022. The work contains: 77 p., 35 fig., 1 table, presentation.

The purpose of this work is to research methods of editing compiled code, evaluation of their ease of use and integration into new projects, and economic justification of creation of modifications for existing projects.

The main subjects of this research are “ILSpy” toolkit and “Harmony” library, which are used for analysis and modification of compiled code in software made using .NET Framework that provides methods of compilation and execution of Common Intermediate Language code.

This work reviews methodology and practical execution of analysis of compiled code, the structure of CIL and its list of instructions, exploration of bytecode for an existing project, identification of functional of targeted project and search of its code, creation of new algorithms using bytecode without the aid of a compiler, and methods of instruction injection into an existing project.

Keywords: code analysis, bytecode, library, decompilation, documentation, integration, instructions, modification, operation code, project documentation, C Sharp, Harmony, ILSpy, .NET Framework

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ТЕРМІНІВ

Скорочення або термін	Значення
ACS	Amazing Cultivation Simulator – програмне забезпечення, обране для модифікації у ході цього дослідження
Bytecode, байт-код	Машино-незалежний код низького рівня, який виконується інтерпретатором
C#, C Sharp, Сі-шарп	Мова програмування. Підтримується платформою .NET Framework
CIL	Common Intermediate Language, байт-код платформи .NET Framework
CLI	Common Language Infrastructure, специфікація стандарту ISO/IEC 23271, яка описує код та середовище виконання мови програмування
.NET	Програмна технологія .NET Framework
ПЗ	Програмне забезпечення
ПТ	Програмна технологія

ВСТУП

Зазвичай, при розробці нового програмного продукту має бути прийняте рішення: або зробити продукт дуже налаштовуваним, щоб охопити якомога більше потенційних клієнтів, або розробити спеціалізований продукт з лише найбільш суттєвими функціями, які обираються під час початкового планування. Однак навіть після завершення розробки потреби клієнтів можуть змінюватись, і інколи початкові розробники не є доступними для оновлення чи модернізації їх продукту. У такій ситуації, найняття третьої сторони для оновлення існуючого продукту є доцільною можливістю.

Складність модифікації існуючого продукту здебільшого залежить від виду програми та використовуваної мови програмування. Веб-сторінки та програмне забезпечення створене за допомогою інтерпретовуваних мов програмування зазвичай не потребують додаткового інструментарію для внесення змін. З іншого боку, модифікація скомпільованого коду є складною проблемою з великою кількість змінних та вимагає детального аналізу задовго до внесення будь-яких алгоритмічних змін.

Також, варто звернути увагу на існування потенційного ринку у сфері створення самодостатніх модифікацій до існуючого програмного забезпечення, тобто таких, що не були створені з метою повної заміни початкового продукту, а існують як невеликі додатки, які можуть бути активовані чи видалені користувачами по бажанню.

Об’єктом дослідження даної роботи є можливість зміни коду вже існуючого продукту. Конкретним досліджуваним продуктом є бібліотека Harmony, яка надає можливість вносити зміни до скомпільованого коду, який був створений на основі програмної технології .NET Framework. Дана робота описує процес створення модифікації до існуючого продукту: аналітичне дослідження продукту, ідентифікацію функціоналу та відповідного коду, та реалізацію модифікації.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ МОДИФІКУВАННЯ БАЙТ-КОДУ

1.1 Обґрунтування вибору напрямку дослідження

На сьогодні, інформаційні технології використовуються в усіх сучасних сферах суспільства. Існує постійна потреба в розробці нових інформаційних систем та програмного забезпечення для вирішення існуючих проблем. Незважаючи на стрімкий темп розвитку та вдосконалення методів розробки програмного забезпечення, можна помітити появу та популяризацію низки продуктів, ціллю яких є уніфікація методів розробки шляхом впровадження та поширення загальних стандартів створення нового та інтеграції існуючого ПЗ. Прикладами найпоширеніших продуктів такого типу є мова програмування JavaScript [1], яка є найбільш використовуваною мовою при створенні веб-контенту, та багатоплатформна програмна технологія .NET Framework [2], однією з цілей якої є зменшення потреби у врахуванні безлічі можливих конфігурацій обладнання, яке використовується для виконання програмного продукту. Популярність таких універсальних технологій призвела до появи великої кількості програмного забезпечення на їх основі.

Програмна технологія .NET Framework була вперше опублікована у 2002 році, більш ніж 20 років тому. З того часу дана технологія отримала безліч оновлень, змін та доповнень. Також, за 20 років було створено неймовірну кількість програмного забезпечення на цій основі в усіх можливих галузях, які використовують інформаційні технології: офіси, підприємства, розважальний сектор, приватний сектор та багато інших.

Однак, з часом потреби користувачів такого програмного забезпечення можуть змінюватись. Це може відбутись як результат розширення їх діяльності, або оновлення програмного або апаратного забезпечення, або появи нових потреб, які не було взято до уваги під час початкової розробки. У великій кількості випадків така ситуація може бути вирішена шляхом контакту з компанією або індивідуальним розробником, який працював над проектом. Однак, існують

випадки, коли це не є можливим або є небажаним: компанія завершила свій бізнес, програміст змінив місце роботи, початковий код програми втрачено тощо.

Одним з подальших варіантів розвитку є повна заміна програмного забезпечення новим. Цей шлях є дуже трудомістким та фінансово витратним: найняття нових розробників або схвалення контрактів на нову розробку з іншими компаніями, зупинка виробництва, необхідність в оновленні апаратури та документації, інструктаж персоналу та інше. Іншим варіантом є модифікація програмного забезпечення спеціалізованими розробниками, які зможуть мінімізувати негативний вплив на роботу клієнта.

Даний напрямок розробки є досить перспективним. Хоча технологія .NET Framework не відповідальна за 100% програмного забезпечення у світі, фокусування дослідження на конкретному випадку допоможе узагальнити методологію створення модифікацій до програмного забезпечення.

1.2 Середовище та мова програмування

Оскільки створюваний програмний продукт націлений на модифікацію ПЗ на основі .NET Framework є доцільним використання тієї ж технології для даного проекту. Це гарантуватиме працездатність кінцевого продукту в будь-яких умовах, оскільки кінцевий користувач вже буде мати середовище необхідне для успішного виконання продукту.

Будь-яка мова програмування, що відповідає стандарту CLI, може бути використана для написання програмного забезпечення з використанням ПТ .NET. У зв’язку з цим можна обрати будь-яку з підтримуваних мов програмування: C#, F#, Swift, C++/CLI, Ada, Visual Basic тощо. Для даного проекту було обрано мову програмування C#.

Обране середовище програмування: Visual Studio 2022. Дане середовище підтримує використання ПТ .NET та мови програмування C#.

1.3 Технічні перешкоди предметної галузі та необхідний інструментарій

Програмна технологія .NET Framework використовує комбінацію компілювання та інтерпретування. Вхідний код, написаний однією з підтримуваних мов програмування, наприклад C#, компілюється у проміжний машино-незалежний байт-код. Задля успішного виконання даного коду використовуване апаратне забезпечення повинне мати встановлений .NET інтерпретатор. Даний інтерпретатор розповсюджується у вільному доступі компанією Microsoft і не потребує додаткових ліцензійних витрат на його використання.

Хоча початковий код можна редагувати з використанням будь-якого текстового редактора, як тільки код було скомпільовано – це перестає бути можливим. Вихідний байт-код не може бути зчитаний чи доцільно змінений без використання більш спеціалізованого інструментарію. Також, оскільки компілятор проводить низку оптимізаційних процесів – отриманий байт-код, зазвичай, втрачає пряму відповідність до початкового коду. Структури класів та процедур залишаються здебільшого незмінні, однак їх список операцій може бути переставленим, а деякі програмні структури заміняються на їх низькорівневі еквіваленти.

Оскільки розуміння використаного алгоритму є передумовою його редагування, для успішного виконання даного проекту необхідно мати доступ до опису даного алгоритму. Ідеальний сценарій у цій ситуації – повний доступ до вихідного коду модифіковуваного програмного забезпечення. Нажаль, у деяких випадках такий доступ не є можливим.

У випадку використання мов програмування, які компілюються напряму в машинний код – пряме зчитування машинних операцій описаних в виконавчому файлі програми є єдиним методом аналізу. Даний опис є дуже низькорівневим і вимагає спеціалізоване знання набору машинних інструкцій та вичерпний аналіз

машинного коду, а також залежить від бібліотек та апаратного забезпечення, які було використано під час компілювання.

Задля підвищення стабільності, програмні файли ПТ .NET Framework, які містять байт-код, містять набір резервної інформації, яка може бути використана спеціалізованим програмним забезпеченням для часткового відтворення початкового коду скомпільованого ПЗ. Прикладом такого інструменту є ILSpy – продукт, який дозволяє майже ідеально відтворити використовувані алгоритми. Однак, пряме використання вихідного коду отриманого цим шляхом може вважатись порушенням прав інтелектуальної власності в деяких законодавствах. У таких випадках, відтворення вихідного коду краще використовувати лише для аналізу алгоритмів (на відміну від редагування отриманого вихідного коду та його компіляції як нового продукту).

Для редагування байт-коду було обрано бібліотеку Harmony. Дана бібліотека дозволяє динамічно редагувати, заміняти, та доповнювати .NET методи без необхідності в прямому редагуванні скомпільованих виконавчих файлів.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Аналіз конкретних вимог модифікації

Оскільки модифікація байт-коду створюється на основі вже існуючого програмного забезпечення – дане дослідження важко виконати як розробку конкретного програмного продукту. Бізнес модель та вимоги щодо архітектури та методології розробки будуть змінюватись залежно від ПЗ до якого розробляється модифікація. Офісні продукти будуть вимагати інтенсивної роботи з засобами людино-машинного інтерфейсу, в той час як промислові виробництва будуть фокусуватись на програмно-апаратній взаємодії.

У зв’язку з цим було прийнято рішення обрати сторонній програмний продукт та звернути більшу увагу на загальну методологію створення модифікації до даного продукту. За потреби дана методологія може бути пристосована до інших продуктів.

Конкретним продуктом для модифікації було обрано програмне забезпечення «Amazing Cultivation Simulator». Даний продукт є розважальним та поширюється за допомогою сервісу електронної дистрибуції Steam. Продукт створено за допомогою програмної технології Unity, яка є сумісною з ПТ .NET Framework, та може бути використаний для опису методології модифікації байт-коду.

Програмний код даного продукту налічує більш ніж 30 просторів імен, які містять більш ніж 2500 класів. Великий проект такого розміру є досить важко синтаксично проаналізувати, навіть якщо велика кількість програмних класів є лише різними наслідувачами елементів інтерфейсу або поділяють однаковий функціонал. В той же час, методологія модифікації існуючого продукту вимагає деякий рівень обізнаності з загальноприйнятими парадигмами програмування, без яких проект такого рівня був би неможливим. У поєднанні з можливостями інструментарію ILSpy, було прийнято рішення не зменшувати обсяг обраного проекту.

У ході ознайомлення з ACS було обрано наступні конкретні вимоги щодо модифікації:

- Цільовий об’єкт зміни: дія Miracle:
    - Охопити 4 конкретні реалізації Miracle: Third Eye, Marrow Sanitization, Charismatic, та Enlightened Awakening;
    - Анулювати використання генератору випадкових чисел під час виконання коду, відповідального за кінцевий ефект даних реалізацій;
- Цільовий об’єкт зміни: Demigod Statue:
    - Замінити використання генератору випадкових чисел під час виконання коду, відповідального за ефект даного об’єкта;
- Модифікація повинна ідентифікувати фрагменти коду, відповідальні за реалізацію логіки цільових об’єктів, та змінювати їх відповідно до поставлених вимог;
- Модифікація повинна використати журнал повідомлень ACS для створення замітки про результат ін’єкції коду. Повідомлення повинне містити кількість змінених операцій при успішному виконанні, або текст повідомлення помилки при неуспішній модифікації;
- Модифікація повинна реалізовувати взаємодію з конфігураційним файлом, у якому зберігаються змінювані значення модифікації;
    - Користувач повинен мати змогу змінювати ці значення шляхом редагування файлу текстовим редактором;
    - Значення, які зчитуються з файлу повинні пройти перевірку на коректність.

Варто зауважити, що ці вимоги було обрано абсолютно довільно. Єдиною метою конкретизації цих вимог є створення конкретної цілі для модифікації. Ця ціль встановлюється клієнтом, і в багатьох випадках виконавець не завжди буде мати відповідний рівень обізнаності з продуктом. У такому випадку необхідно або провести додаткові консультації з клієнтом або відповідним технічним персоналом компанії, або самостійно виконати дослідження щодо наявного функціоналу

програми. Це дослідження може бути виконано як шляхом прямого користування програмою, або як поступовий аналіз коду проекту.

У зв’язку з великим розміром проекту, обидва підходи є досить часогроміздкими. Також припустимо, що нам було надано додаткову інформацію щодо даних вимог: обидва об’єкти модифікують властивості об’єктів інших класів програми, і ці властивості включають стрічки «Perception», «Constitution», «Charisma», «Intelligence», та «Luck»; різні Miracle модифікують лише по одній властивості, а Demigod Statue – усі одразу. Дану інформацію буде використано під час аналізу коду продукту.

2.2 Дослідження інструментального функціоналу

2.2.1 Структура коду Common Intermediate Language та ILSpy

Common Intermediate Language (надалі CIL) – це проміжна машино-незалежна мова, яка використовується технологіями .NET Framework та Mono як універсальний байт-код. Усі компілятори, які підтримують дані технології, повинні мати здатність транслювати підтримувані мови програмування в CIL.

У даному вигляді код програми значно відрізняється від вхідного коду. CIL за своєю структурою нагадує низькорівневий Assembler. Програмною парадигмою CIL є виконання програмного коду на основі загального стеку: інструкції та їх операнди містяться в стеку та виконуються послідовно. При виконанні інструкції, яка потребує операндів, дані операнди також отримуються із загального стеку та мають бути записані *перед* виконанням даної інструкції. На рисунку 2.2.1 можна розглянути загальну структуру байт-коду CIL.

Список інструкцій CIL містить різноманітні операції над стеком, такі як: запис змінної на стек, арифметика над змінними у стеку, стрибки у різні частини виконуваного коду залежно від змінної у стеку, конвертація змінних, використання

локальних змінних та наданих аргументів (запис та зчитування), логічні операції, обробка винятків та інші.

Кожний рядок програми містить адресу, на яку можна процедурно посилатись за допомогою різних стрибкових інструкцій, найчастіше jmp. Ці інструкції зазвичай використовуються для реалізації різних конструкцій програми, таких як if, for, while, switch тощо.

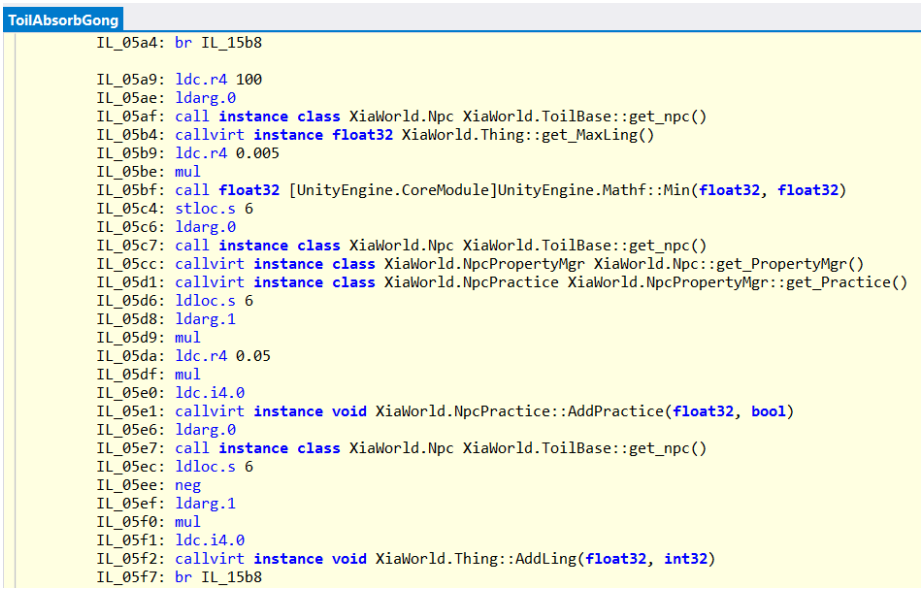

Рисунок 2.2.1 – Фрагмент байт-коду Common Intermediate Language з ПЗ ACS

Таблиця 2.2.1 – Приклад інструкцій CIL

Операційний код	Інструкція	Виконувана дія
0x58	add	Повернути суму двох значень зі стеку
0x27	jmp	Вихід з поточного методу та стрибок у інший
0x20	ldc.i4 <int>	Запис цілого значення на стек
0x14	ldnull	Запис null на стек
0x7A	throw	Виклик винятку
0x00	nop	Пуста операція
0x3B	beq <int>	Націлене розгалуження
0x0C	ldloc <int>	Запис локальної змінної на стек
0x09	ldarg <int>	Запис вхідного аргументу на стек

Повний список доступних операцій налічує більш ніж 100 операційних кодів [3].

При використанні інструкцій, які викликають процедури інших класів, програма очікує наявність у стеку усіх аргументів даної процедури. Розглянемо приклад такого виклику: на рисунку 2.2.1, на адресі «IL_05e1» можна побачити виклик інструкції callvirt, що в свою чергу викликає процедуру «AddParticle()» класу «NpcPractice». Ця процедура вимагає двох аргументів типу float32 та bool. Запис цих аргументів було виконано одразу до інструкції callvirt, де можна побачити виклик математичних операцій mul (множення) які змінюють значення змінної типу float. Тип bool, як такий, не існує в CIL. Замість нього використовуються числові значення, де 0 відповідає false, а будь-яке інше значення – true.

2.2.2 Функціонал бібліотеки Harmony

Бібліотека Harmony дозволяє модифікувати, заміняти та декорувати методи CIL під час виконання програми [4]. Бібліотека не вимагає ніяких додаткових залежностей. Harmony підтримує як 32-, так і 64-бітні архітектури та може використовуватись у проектах, призначених для операційних систем Windows, Mac та Linux. Підтримуються програмні технології .NET та Mono.

Дана бібліотека надає доступ до декількох методів модифікації коду: префіксація, постфіксація, транспліювання, ін’єкція та інші. Кожен метод має свої переваги та недоліки. Як правило, використання будь-якого з цих методів потребує виконання попереднього аналізу програми, оскільки одним з перших кроків модифікації є обрання класу та методу, який буде змінено. Для компілювання нової модифікації необхідно мати доступ до виконавчої бібліотеки програми, що в свою чергу надає можливість отримати повний список публічних методів та класів програми. Однак, дана інформація є дуже обмеженою та недостатньою для зміни

алгоритмів. Harmony не надає інструментарій необхідний для вирішення цієї проблеми. Для цього було використано додаткове програмне забезпечення ILSpy.

Префіксування – це метод, який дозволяє виконання стороннього коду до виконання коду цільового методу. Це дозволяє отримати доступ та змінювати аргументи методу, встановити кінцевий результат методу, або додати початкову частину алгоритму, який буде закінчено під час постфіксувння.

Постфіксування – це метод, який дозволяє виконання стороннього коду після виконання коду цільового методу. Це дозволяє змінювати результат виконання методу або завершувати алгоритми з префіксування.

Трансплюювання надає доступ до повного списку інструкцій методу та дозволяє змінювати будь-які індивідуальні інструкції або аргументи на довільні. Цей метод є найбільш потужним, але в той час і найбільш небезпечним. Оскільки зміна коду відбувається після компілювання – внесені зміни не перевіряються на відповідність чи відсутність помилок. Даний метод потребує спеціалізованих навичок роботи з кодами операцій та обширні програмні знання. Компетентне використання трансплюювання дозволяє програмісту змінювати код програми майже без обмежень.

В загальному, під час трансплюювання бажано не торкатись методів, які керують потоком виконання програми: bne, bqe, br, switch та інші. Ці операції часто посилаються на стрічки які відповідають конкретним рядкам коду, і їх зміна може призвести до недієздатності усієї програми. Якщо внесені зміни не вимагають великої кількості нових інструкцій – то це не буде проблемою, але якщо необхідно створити та виконати великі нові процедури – краще знайти спосіб внесення цієї зміни через префіксування або постфіксування.

Ін’єкції надають доступ до значень методу, які в нормальних умовах не є доступними. Наприклад, «__result» надає доступ до результату методу, «___fields» надає доступ до приватних змінних, «__args[]» надає доступ до аргументів методу та інші.

2.3 Проектування модифікації байт-коду

Під час розробки було вирішено використати Раціональний уніфікований процес проектування, або Rational Unified Process (RUP). В даному методі життєвий цикл проекту, або розробка програмного забезпечення, поділяється на чотири фази. На цих етапах відбуваються різні види діяльності: моделювання, аналіз і проектування, впровадження, тестування та застосування.

Раціональний уніфікований процес (RUP) є ітеративним, тобто повторюваним: всі основні дії процесу повторюються протягом усього проекту. Процес є гнучким, оскільки різні компоненти можуть бути пристосовані до вимог, а фази циклу можна повторювати, поки програмне забезпечення не відповідатиме вимогам і цілям [5].

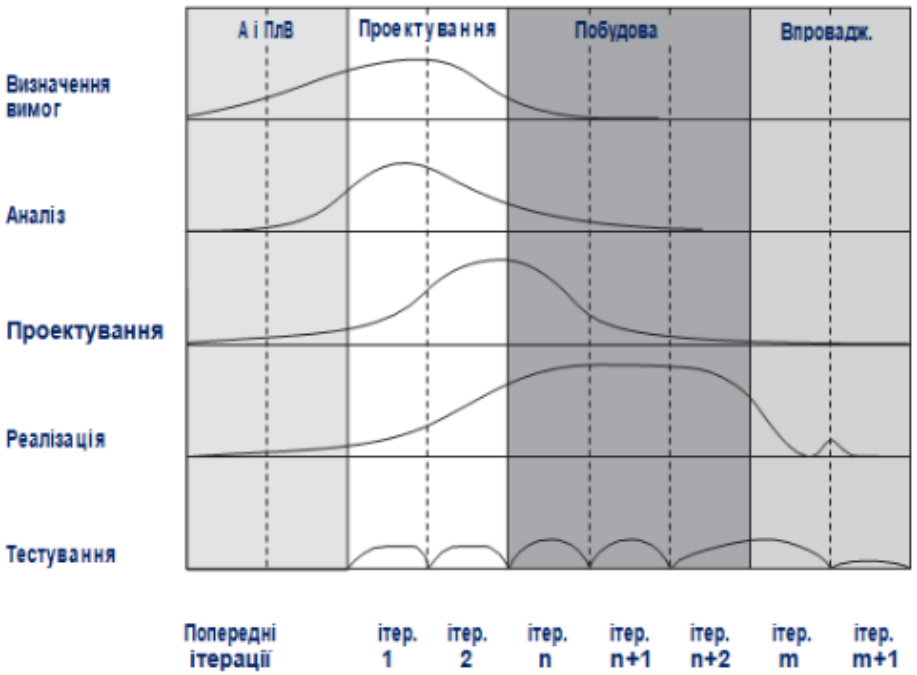

Рисунок 2.3.1 – Робочий потік RUP-проектування

Ітерацій з отримання проміжних варіантів прототипу може бути декілька, в кожній з яких додається функція і повторно моделюється робота прототипу. І так

до тих пір, поки не будуть промодельовані всі функції, задані у вимогах до системи. Фази тестування та пристосування також виконуються ітеративно.

Поступова ітеративна реалізація функціоналу дозволяє уникнути ситуацій, у яких реалізація усіх функцій системи одночасно призводить до громіздкості системи. Швидка реалізація функціональних можливостей системи є головною перевагою даної моделі. Також спрощується внесення змін у зв'язку із заміною окремих функції.

2.3.1 Аналіз декомпільованого коду програмного продукту

Оскільки змінюване програмне забезпечення поширюється лише в скомпільованому виді – необхідно провести початковий аналіз коду з метою знаходження алгоритмів, які необхідно змінити. Задля цього був використаний інструментарій ILSpy, який дозволяє частково декомпілювати програмне забезпечення, яке використовує CIL.

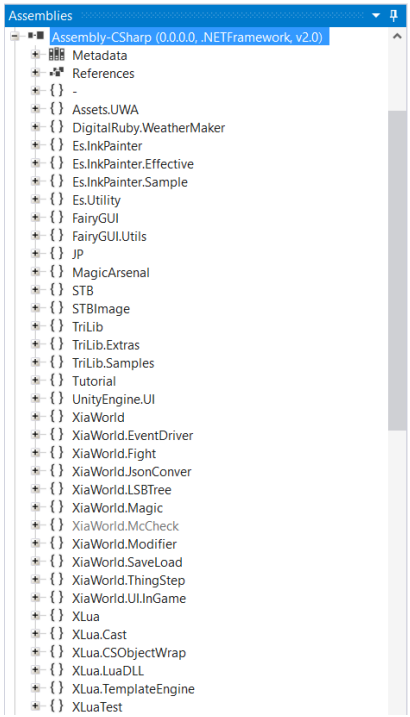

Рисунок 2.3.2 – Список просторів імен ACS

Після відкриття основної бібліотеки ACS (Assembly-CSharp.dll) у ILSpy можна побачити список просторів імен бібліотеки (рис. 2.3.2).

Першочерговим завданням аналізу будь-якого проекту є оцінка компетентності програмістів, які реалізували даний програмний продукт. Дотримання парадигм об’єктно-орієнтованого програмування є майже обов’язковим при конструюванні проектів великого розміру, оскільки в протилежному випадку внесення будь-яких змін буде потребувати більше і більше часу на їх виконання, що в результаті призведе до повного провалу проекту. Доцільне групування класів за їх призначенням та їх відповідне найменування стають незамінними методами проектування.

Деякі з наявних просторів імен налічують лише 1-5 класів, в той час як інші налічують сотні. Невеликі простори можна переглянути дуже швидко: «Asset.UWA», «Es», «FairyGUI», «JP», «STB» – усі відповідальні лише за невелику кількість вторинних утиліт програми, без реалізації основних алгоритмів. У таких випадках ці невеликі частини коду можна вважати неважливими.

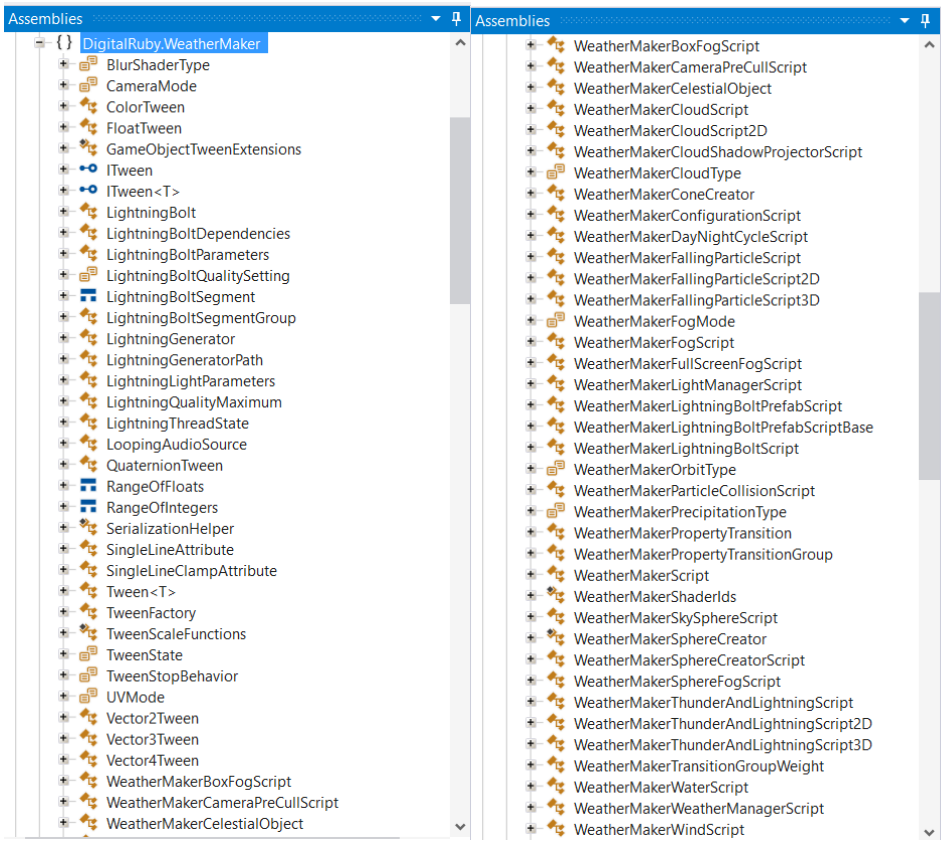

Рисунок 2.3.3 – Класи простору «DigitalRuby.WeatherMaker»

Великі простори імен можна поверхнево оцінити оглядом імен наявних класів. Наприклад, простір «DigitalRuby.WeatherMaker» налічує декілька десятків класів (рис. 2.3.3), але з їх імен можна побачити, що вони найбільш вірогідно відповідають за симуляцію погоди.

Хоча потенційно цей простір міг зайняти велику кількість часу для аналізу, з наявних найменувань класів можна досить впевнено скласти висновок, що ці класи відповідають за симуляцію або відображення погоди, що не є важливим для виконання поставлених вимог.

Простори імен, які починаються з «XLua» відповідають за інтеграцію скриптової мови програмування Lua у проект [6]. Алгоритми створені за допомогою Lua не входять до виконавчої бібліотеки, а постачаються з іншими необхідними файлами продукту. Зазвичай ці файли є легкодоступними для кінцевого користувача, оскільки скриптові мови часто призначені для використання користувачами, а не початковими розробниками.

Як результат, хоча цільова бібліотека має більше десятка просторів, більшість з них можна виключити завдяки програмістам, які проектували даний продукт з дотриманням загальноприйнятих норм найменування та структуризації коду. Алгоритми, в яких ми зацікавлені, найбільш вірогідно знаходяться у просторі «XiaWorld». Хоча він є найбільшим з усіх, навіть тут ми можемо виключити деякі частини:

- «XiaWorld.JsonConver» відповідає за зчитування та запис JSON файлів, які зазвичай використовуються для збереження даних;
- «XiaWorld.UI.InGame» відповідає за елементи інтерфейсу програми;
- «XiaWorld.LSBTree» містить алгоритми для роботи з структурами даних типу «дерево»;
- «XiaWorld.SaveLoad» містить функціонал збереження та завантаження даних програми;
- «XiaWorld.McCheck» містить набір перевантажених функцій, таких як заміна стрічок або методи конвертації типів;

Як висновок: більшість алгоритмів, які відповідають за основну логіку програми, найбільш вірогідно знаходяться у просторі імен XiaWorld.

2.3.2 Проектування архітектури ін’єктованого коду

Хоча основною задачею бібліотеки Harmony є зміна скомпільованого коду – ні бібліотека Harmony, ні стандартний інструментарій .NET Framework не надають доступ до методів початкової ін’єкції бібліотеки у виконавчий код. Існує ряд сторонніх програм, які часто використовуються для виконання цього кроку: Unity Doorstop, UnityAssemblyInjector, MonoJunkie та інші. Використання одного з цих методів є неоманним у випадку, якщо цільове програмне забезпечення не надає доступ до виконання стороннього коду.

ACS був розроблений з інтеграцією скриптової мови Lua. Дана мова програмування часто використовується у подібних проектах з метою надання користувачам можливості створення невеликих параметричних модифікацій. Більшість доступних методів дозволяють лише обмежений контроль над змістом XML файлів, які містять змінювані параметри та описи лише невеликої кількості систем. Однак, один з доступних методів дозволяє виконання процедур сторонніх бібліотек. Цього методу достатньо щоб виконати початкову ін’єкцію бібліотеки в виконавчий код програми.

Також, оскільки ACS використовує багатоплатформний рушій Unity та мову програмування C# - використання тих самих засобів розробки дозволяє уникнути встановлення додаткового програмного забезпечення чи бібліотек, оскільки усі елементи необхідні для роботи модифікації вже будуть присутні в середовищі користувача.

Усі вищеописані деталі повинні бути взяті до уваги при проектуванні будь-якої модифікації.

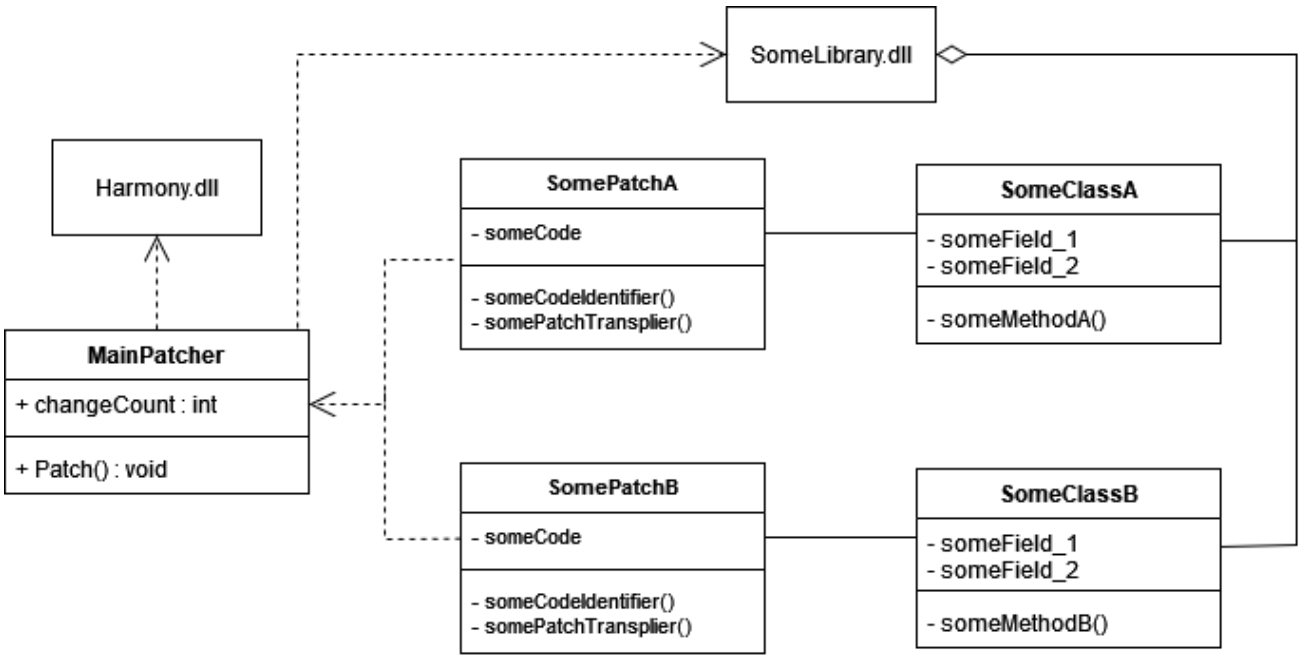

Рисунок 2.3.4 – Узагальнена діаграма класів модифікації

Класова структура модифікації повинна мати головний вхідний клас, який ініціює ін’єкцію коду, та вторинні класи, кожен з яких відповідає змінюваному класу цільового продукту. На рисунку 2.3.4, головному класу відповідає «MainPatcher», вторинні класи: «SomePatchA» та «SomePatchB», початкові класи: «SomeClassA» та «SomeClassB». Головний клас використовує методи бібліотеки Harmony та має доступ до структури класів змінюваної бібліотеки «SomeLibrary.dll», що також містить вищезгадані початкові класи.

Варто зауважити, що кількість початкових та вторинних класів може бути довільною. Задля кращої видимості коду рекомендується створювати по одному вторинному класу на кожен змінюваний початковий клас.

Також варто зазначити, що бібліотека Harmony може виконувати декілька ін’єкцій одночасно, навіть якщо різні ін’єкції намагаються модифікувати один і той самий метод. З цієї причини, замість модифікації коду на основі конкретних адрес інструкцій рекомендовано створювати алгоритми пошуку конкретних фрагментів коду, наприклад порівняння на повну відповідність на основі збереженого фрагменту. Це також допомагає запобігти майбутній непридатності модифікації якщо змінюваний продукт отримає оновлення, яке змінює цільовий клас але не змінює конкретний цільовий алгоритм.

3 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Застосування методології аналізу алгоритмів

Хоча велику частину наявного коду було відхилено як потенційно цільового, частини, що залишились, є досить суттєвими за розміром. Як було зазначено у розділі 2.2.1, успішність та швидкість пошуку необхідних частин коду залежить від поставлених конкретних вимог клієнта та наданої або здобутої інформації про цільовий продукт.

Найбільш наївним методом є перебір усього доступного коду. Хоча для малих за розміром продуктів цей метод займе стільки ж часу, як і більш витончені варіанти, для великих проектів цей метод є неприпустимим, оскільки кількість часу необхідного для аналізу такого об’єму коду перевищує будь-які реалістичні часові обмеження проекту.

Для продуктів, які містять велику кількість елементів інтерфейсу користувача, дуже ефективним методом аналізу є пошук коду, який містить стрічки тексту, які використовуються в елементах інтерфейсу, які відповідають за ініціювання або інформування про статус виконання цільового функціоналу. Оскільки ACS є розважальним продуктом, при його розробці було створено дуже велику кількість таких елементів.

Відповідно до додаткової інформації, яку було надано клієнтом, стрічки тексту що пов’язані з необхідним функціоналом можуть містити наступні фрагменти: Miracle, Third Eye, Marrow Sanitization, Charismatic, Enlightened Awakening, Demigod Statue, Perception, Physique, Charisma, Intelligence, та Luck. Інструментарій ILSpy містить функціонал пошуку стрічок та можливість обмеження цього пошуку до необхідного підтипу: класи, змінні, методи, поля, події, константи, ресурси та інші. Було здійснено відповідний пошук цих стрічок.

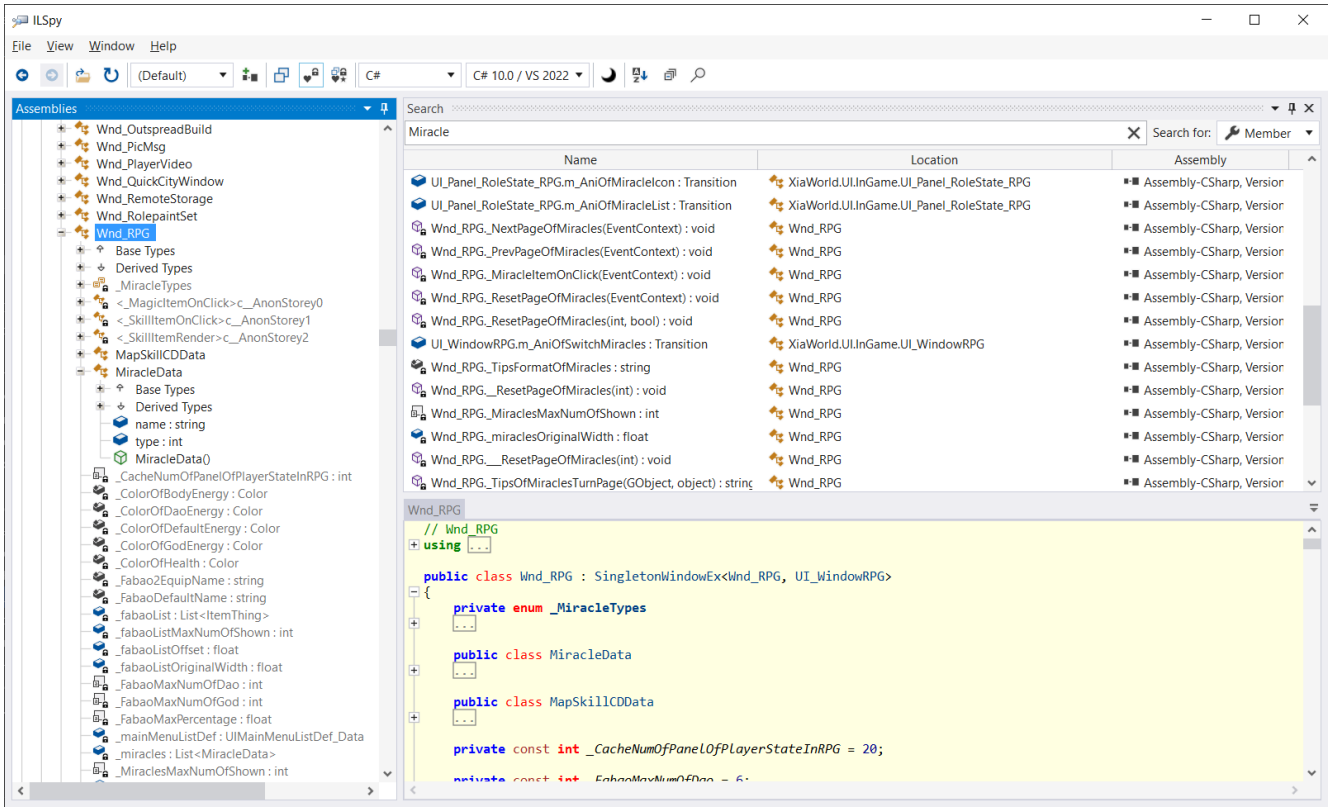

Рисунок 3.1.1 – Пошук стрічки «Miracle» у коді програми

Як можна побачити на рисунку 3.1.1, пошук даної стручки повернув результати з двох різних класів програми. Однак, простір імен «XiaWorld.UI.InGame» було відхилено як потенційно досліджуваного на попередньому етапі аналізу коду, тому клас «UI_WindowRPG» можна ігнорувати, оскільки він належить до вищезгаданого простору імен та містить лише технічну інформацію елементу інтерфейсу без його реалізації. В той же час, можна побачити, що клас «Wnd_RPG» наслідує загальний (або шаблонний) клас «SingletonWindowEx», а одним із шаблонних аргументів цього наслідування є «UI_WindowRPG». З цього можна зробити висновок, що цей клас реалізує елемент інтерфейсу, що відповідає за ініціювання функціоналу Miracle. Хоча він не містить необхідних конкретних алгоритмів, цей клас можна добавити до списку коду, який буде досліджено більш детально у майбутньому.

Пошук стрічок Third Eye, Marrow Sanitization, Charismatic та Enlightened Awakening у коді програми не надав результатів. Можна зробити припущення, що ці стрічки проходять через процес локалізації та зберігаються у XML файлах

продукту. Це припущення підкріплюється аналізом попередньо ідентифікованого класу «Wnd_RPG», оскільки він містить частину коду, відповідального за завантаження та заміну стрічок тексту інтерфейсу, який в коді написано китайською мовою (мова початкових розробників ACS).

Пошук серед усіх XML файлів продукту можна виконати за допомогою програмного забезпечення Notepad++:

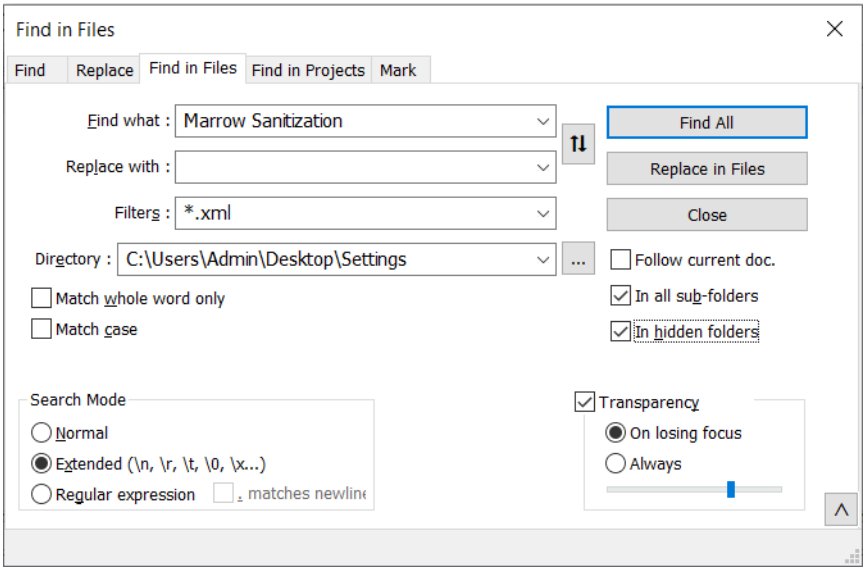

Рисунок 3.1.2 – Структура пошукового запиту

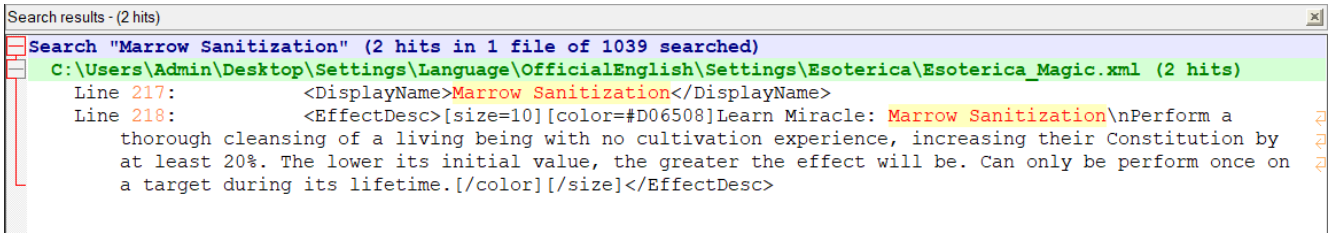

Рисунок 3.1.3 – Результат пошуку стрічки «Marrow Sanitization»

При перегляді повного контексту знайденого результату у файлі, можна побачити, що внутрішнім ім’ям цього набору даних є «Magic_Healing_Physique_1».

```
216     <Text Name="Magic_Healing_Physique_1">
217         <DisplayName>Marrow Sanitization</DisplayName>
218         <EffectDesc>[size=10][color=#D06508]Learn Miracle: Marrow Sanitization\nPerform a thorough
            cleansing of a living being with no cultivation experience, increasing their Constitution
            by at least 20%. The lower its initial value, the greater the effect will be. Can only be
            perform once on a target during its lifetime.[/color][/size]</EffectDesc>
219     </Text>
```

Рисунок 3.1.4 – Фрагмент змісту знайденого файлу

Пошук нової стрічки у коді програми не дає результатів. При повторному пошуку нової стрічки серед XML файлів програми можна знайти нове посилання:

```
Search results - (8 hits)
Search "Magic_Healing_Physique_1" (8 hits in 8 files of 1039 searched)
  C:\Users\Admin\Desktop\Settings\Esoterica\Esoterica_Magic.xml (1 hit)
    Line 792:    <Esoterica Name="Magic_Healing_Physique_1">
  C:\Users\Admin\Desktop\Settings\Language\OfficialEnglish\Settings\Esoterica\Esoterica_Magic.xml (1 hit)
    Line 216:        <Text Name="Magic_Healing_Physique_1">
```

Рисунок 3.1.5 – Результат пошуку стрічки «Magic_Healing_Physique_1»

При перегляді контексту нового файлу, можна знайти початковий опис шуканого функціоналу:

```
C:\Users\Admin\Desktop\Settings\Esoterica\Esoterica_Magic.xml - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Main.lua | new 3 | BodyMiracleDerandomizer.cs | new 2 | Esoterica_Magic.xml | Esoterica_Magic.xml
792     <Esoterica Name="Magic_Healing_Physique_1">
793         <DisplayName>伐毛洗髓之术</DisplayName>
794         <Icon>res/Sprs/ui/icon_xisui01</Icon>
795         <StarLevel>2</StarLevel>
796         <Difficulty>8</Difficulty>
797         <!--<Desc>对一名无修为的生物伐毛洗髓，使他的根骨得到增加，初始值越低效果就越好。目标一生只能被施展一次。</Desc>-->
798         <GLevel>God</GLevel>
799         <Element>None</Element>
800
801         <EffectDesc>
            [size=10][color=#D06508]习得神通：伐毛洗髓\n对一名无修为的生物伐毛洗髓，使他的根骨得到增加，初始值越低效果就越
            好，最少会提高2成根骨。目标一生只能被施展一次。[/color][/size]</EffectDesc>
802         <Magics>
803             <li>Healing_Physique</li>
804         </Magics>
805     </Esoterica>
```

Рисунок 3.1.6 – Фрагмент змісту файлу опису функціоналу програми

Як можна побачити на рисунку 3.1.6, цей фрагмент прив’язує шуканий функціонал до внутрішнього імені «Healing_Physique». При пошуку даної стрічки у коді ACS можна знайти наступну структуру:

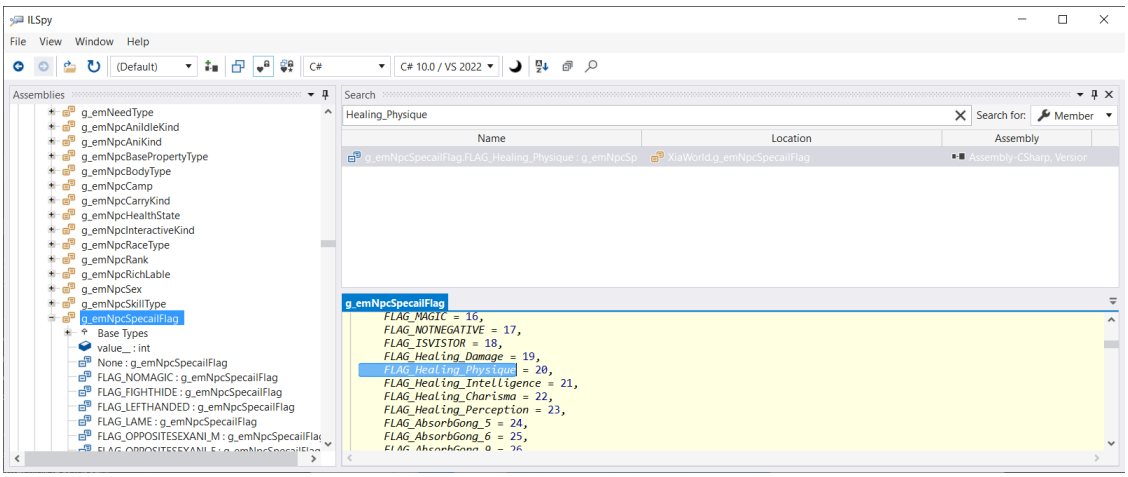

Рисунок 3.1.7 – Знайдений перелік деяких внутрішніх значень програми

Разом з цією стрічкою, можна також побачити аналогічні значення, які закінчуються словами «Intelligence», «Charisma» та «Perception». Якщо повторити процес пошуку стрічки «Marrow Sanitization» зі стрічками «Enlightened Awakening», «Charismatic» та «Third Eye», то ми повернемося до цього ж переліку внутрішніх значень з їх відповідними кінцевими словами. Як результат, ми можемо використати функціонал аналізу цього конкретного переліку та виявити де ці значення використовуються.

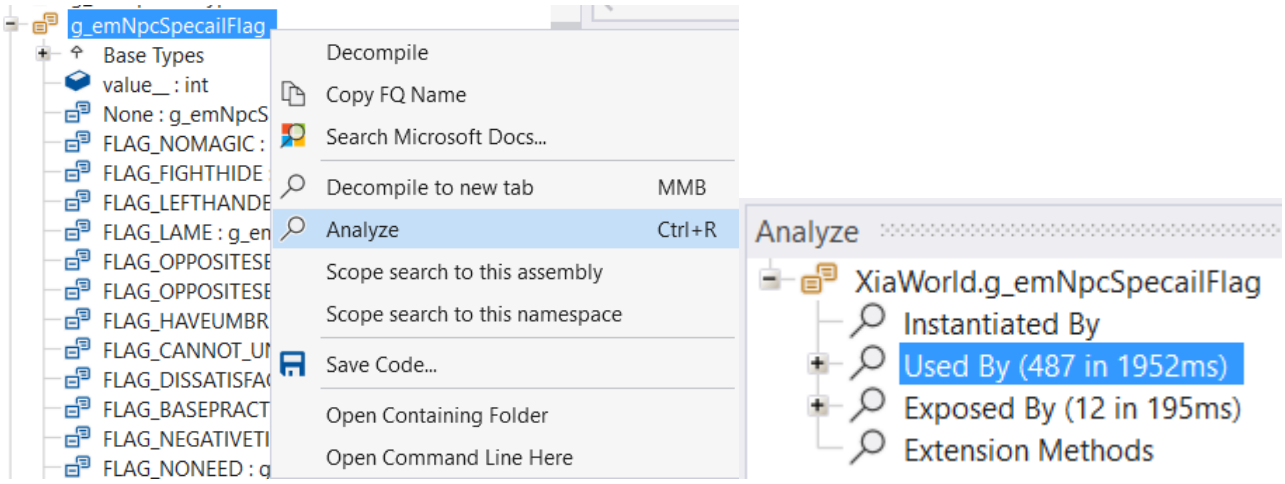

Рисунок 3.1.8 – Ініціювання та результат аналізу переліку даних

Як можна побачити, ці дані використовуються у 487 інших методах програми. Такий великий обсяг коду важко проаналізувати, тому необхідно знайти спосіб відхилити частину отриманих результатів, або знайти спосіб пошуку

використання конкретного значення з цього набору даних. Нажаль, ILSpy не надає функціоналу для виконання такого пошуку. Однак, можна зробити повну декомпіляцію проекту та використати функціонал Notepad++ для пошуку цього конкретного значення в усьому проекті.

Метод повної декомпіляції продукту є досить небажаним. Процес декомпіляції може зайняти деякий час, і як результат отримується велика кількість файлів початкового коду, які досить важко ефективно аналізувати. Можна помилково зробити висновок, що це є найлегшим методом, оскільки ці файли можна відкрити у новому проекті та використати повний функціонал середовища розробки для аналізу коду. Однак, цей підхід створює занадто велику кількість проблем, вирішення яких вимагає великої кількості часу: необхідність у пошуку та використанні усіх бібліотек та їх конкретних версій, які були використані при створенні проекту, відсутність структуризації проекту, необхідність в тривалому налаштуванні середовища розробки тощо. Без вирішення усіх цих проблем, створений проект буде наповнений помилками, які унеможливлять ефективний пошук. У рамках цього дослідження було вирішено використати метод повної декомпіляції лише для пошуку конкретних значень і лише у випадках, коли цей пошук неможливо виконати використовуючи інший функціонал.

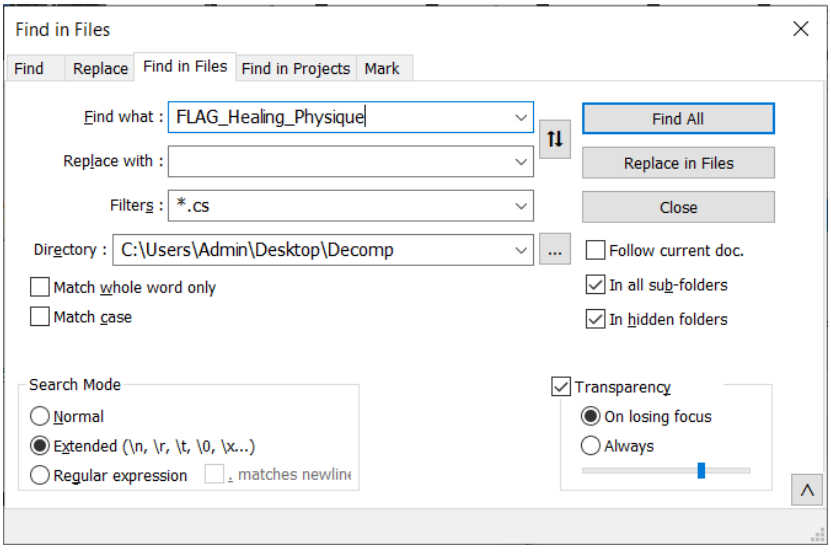

Рисунок 3.1.9 – Пошуковий запит для декомпільованого проекту

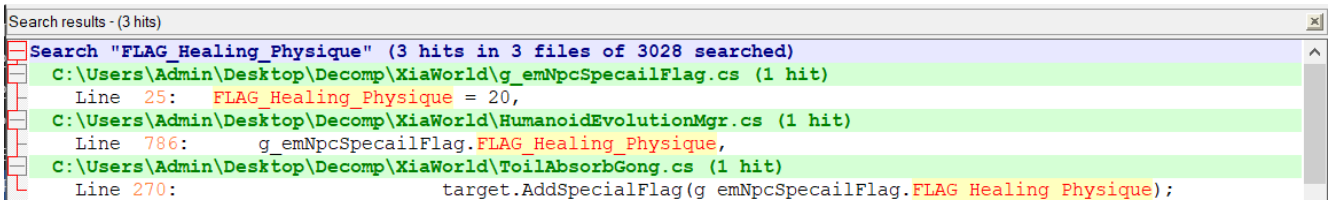

Рисунок 3.1.10 – Результат пошуку декомпільованого проекту

З результату пошуку на рисунку 3.1.10 можна побачити, що шукані дані використовуються лише у двох класах: «HumanoidEvolutionMgr» та «ToilAbsorbGong». Для аналізу цих класів можна повернутись до ILSpy.

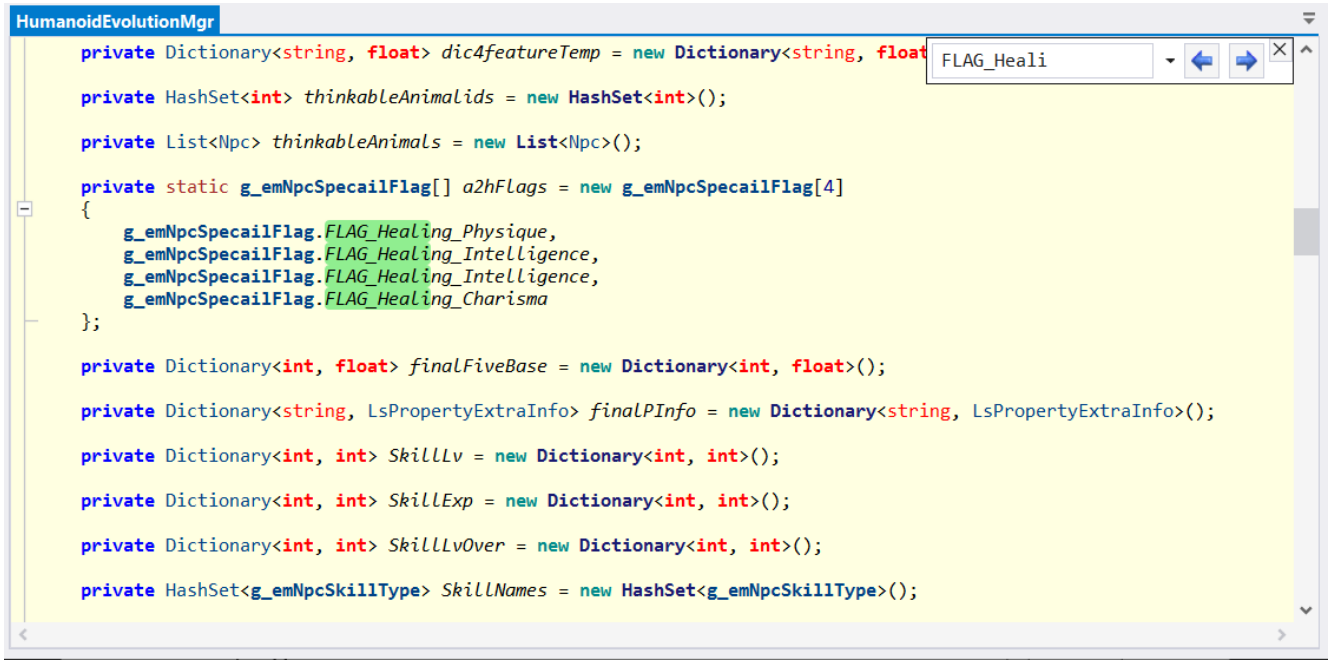

Рисунок 3.1.11 – Фрагмент коду класу «HumanoidEvolutionMgr»

У даному випадку, змінні з переліку використовуються лише для декларації та наповнення статичного масиву класу. Алгоритмічний функціонал відсутній.

В свою чергу, клас «ToilAbsorbGong» містить велику кількість методів, кожен з яких наповнений алгоритмами впровадження деякого функціоналу ACS. Після перегляду цього коду було виявлено фрагмент, який відповідає за одну з цільових функцій – метод «OnStepToil()».

Рисунок 3.1.12 – Фрагмент коду класу «ToilAbsorbGong»

Даний метод містить у собі алгоритми, необхідні для роботи більш ніж 20 елементів продукту. Метод містить у собі конструкцію switch, який залежно від вхідної змінної виконує різні обчислення. Конструкція поділена на 25 частин. Необхідні обчислення було ідентифіковано в частинах 6-9, які збігаються до однакового виконавчого коду у частині 9, яку можна побачити на рисунку 3.1.12.

Іншим цільовим функціоналом, обраним під час створення конкретних вимог проекту, є Demigod Statue. Єдиною додатковою інформацією є те, що цей функціонал модифікує ті ж самі атрибути, що і Miracle, але усі одночасно. З попереднього аналізу ми можемо побачити, що атрибут «FLAG_Healing_Physique» змінюється після виклику методу «AddAddition()» з використанням подібного атрибуту «Physique». Після аналізу переліку атрибутів «g_emNpcBasePropertyType», до якого належить «Physique», можна помітити присутність усіх значень, інформацію про які було надано клієнтом на етапі аналізу вимог:

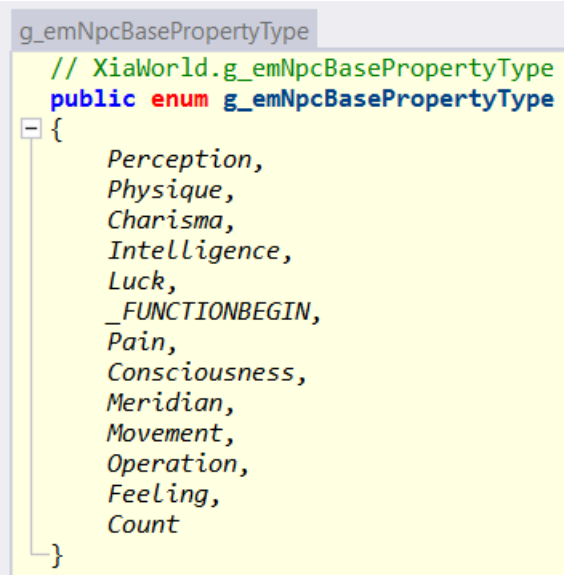

Рисунок 3.1.13 – Перелік значень «g_emNpcBasePropertyType»

Після виконання пошуку використання цих конкретних атрибутів у декомпільованому коді програми, було знайдено наступний фрагмент коду класу «UILogicMode_IndividualCommand»:

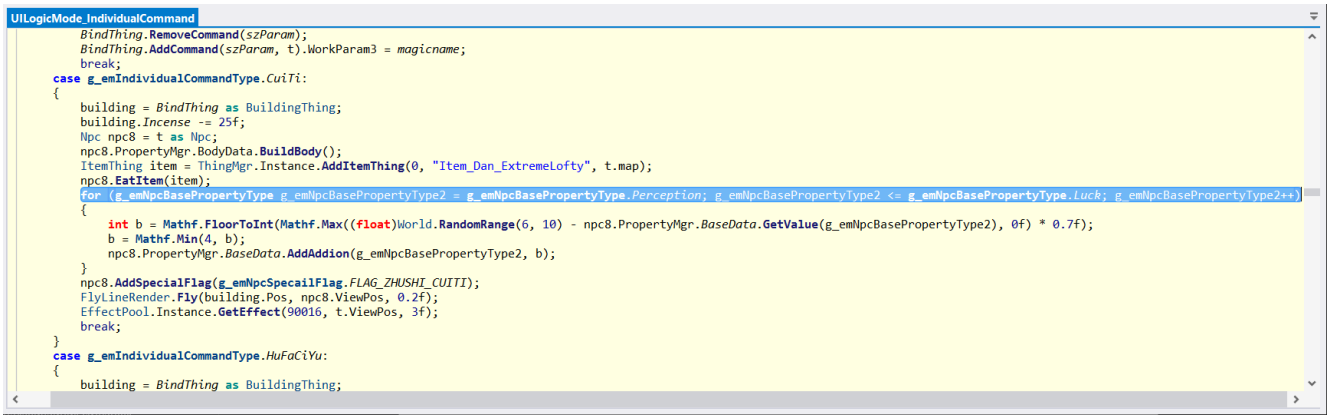

Рисунок 3.1.14 – Фрагмент коду класу «UILogicMode_IndividualCommand»

Тут можна побачити дещо нестандартне використання атрибутів переліку як початкового та кінцевого значень циклу for. Однак даний алгоритм збігається з профілем, наданим клієнтом під час опису функції: зміна значень набору атрибутів з використанням генератору випадкових чисел (метод «World.RandomRange()»).

Хоча виконання пошуку коду, відповідального за функціонал Miracle вимагав декількох повторних пошуків використання специфічних стрічок тексту як

в XML файлах продукту, так і в декомпільованому коді програми, - пошук функціоналу Demigod Statue зайняв набагато менше часу та менше дій.

Проникливий читач міг помітити, що під час цього пошуку стрічку «Demigod Statue» або її варіації не було використано взагалі. Це слугує прикладом того, що у процесі пошуку алгоритмів та аналізу коду будь-якого продукту програміст повинен приділяти увагу структурі коду та помічати особливості його написання. Алгоритм на рисунку 3.1.14 досить подібний до попередньо знайденого алгоритму на рисунку 3.1.12, що надає деякої впевненості у тому, що цей фрагмент коду відповідає шуканому функціоналу. Ця впевненість підкріплюється відсутністю інших подібних методів, які б використовували ідентифіковані атрибути, та може бути остаточно підтверджена зворотнім пошуком серед XML файлів локалізації, більш детальним аналізом коду, або спробою модифікації цього коду та прямого використання цільового функціоналу продукту.

3.2 Реалізація програмної частини

Після створення нового проекту, до списку використовуваних бібліотек було добавлено бібліотеку «Harmony.dll». Це надає доступ до усіх методів та можливостей бібліотеки.

ACS складається з сотень пов’язаних класів та методів, велика кількість яких використовує XML файли задля збереження змінних даних. Більшість програмного функціоналу заходиться в бібліотеці «Assembly-CSharp.dll», що містить майже весь код необхідний для виконання програми. Дана бібліотека була інтегрована в проект з метою отримання доступу до списку класів програми та їх методів.

Хоча велика частина пошуку цільових алгоритмів було виконано на основі реконструйованого коду C#, модифікація коду тим же чином не є можливою. Код вже скомпільовано у готовий продукт, а поширення чи поєднання декількох модифікацій шляхом заміни виконавчого файлу продукту є дуже поганою ідеєю.

Реалізацію можна поділити на чотири етапи:

1. Конструювання нового байт-коду, що потребує повторного аналізу цільового алгоритму у вигляді байт-коду та його розбиття на відповідні фрагменти;
2. Конструювання керуючого коду модифікації;
3. Створення алгоритму процедурного пошуку цільових фрагментів та їх заміни на новий байт-код;
4. Ін’єкція нового байт-коду за допомогою доступних методів.

3.2.1 Конструювання нового байт-коду

У ході використання даного програмного продукту було обрано конкретний функціонал, який необхідно змінити. Функціонал було ідентифіковано як такий, що неможливо модифікувати завдяки редагуванню лише XML файлів продукту. Задачею даної роботи було обрано модифікацію цього функціоналу завдяки трансплюванню коду.

Після тривалого дослідження декомпільованого коду за допомогою ILSpy, цільовий функціонал було ідентифіковано в класах «ToilAbsorbGong», метод «OnStepToil()», та «UILogicMode_IndividualCommand», метод «Apply2Thing()». Оскільки модифікація коду напряму є неможливою, необхідно ідентифікувати дану частину коду в байт-коді програми. Повний байт-код першого методу налічує майже 2000 інструкцій, другого – майже 3000.

Для ідентифікації першого фрагменту було використано аналіз наявної конструкції switch. У декомпільованому коді, усі цільові фрагменти збігаються до однакового коду, однак вони використовують 4 різні значення конструкції switch, перші 3 з яких не містять ніякого коду. Очікується, що у результаті компіляції та автоматичної оптимізації, дана конструкція буде виконувати стрибок до однієї адреси чотири рази поспіль. У цьому випадку достатньо лише знайти інструкцію

switch у байт-коді методу, з якої можна буде отримати конкретну адресу цільового фрагменту.

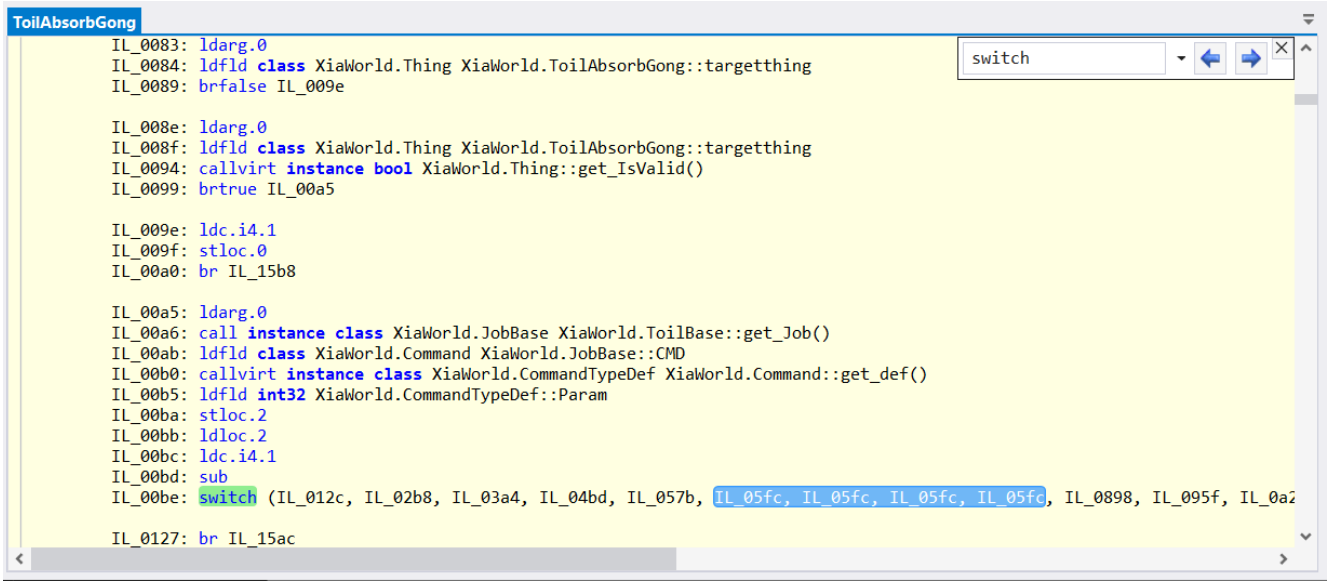

Рисунок 3.2.1 – Шукана конструкція switch у байт-коді продукту

Як результат, можна виконати пошук адреси «IL_05fc», що буде відповідати початку цільового виконавчого коду:

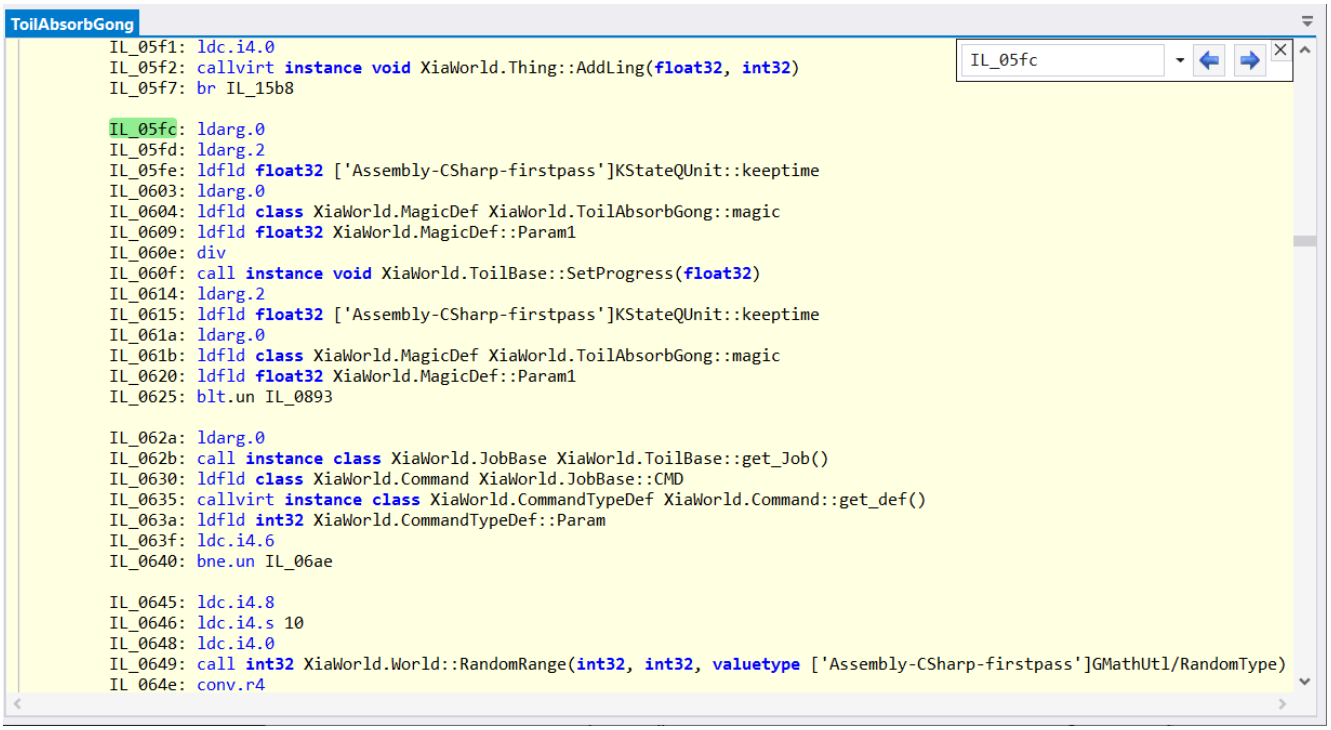

Рисунок 3.2.2 – Фрагмент коду, відповідального за функціонал Miracle

На рисунку 3.2.2 на адресі «IL_0649» можна побачити виклик методу випадкового генератору чисел, а до цього – запис змінних на стек, які будуть використані як аргументи виклику даного методу. Варто зауважити, що цей виклик повторюється ще 3 рази у подальшому коді для різних реалізацій Miracle. Відповідно до поставлених вимог, буде виконано заміну усіх чотирьох реалізацій.

Для ідентифікації другого фрагменту можна використати пошук виклику методу «World.RandomRange()». Хоча ця функція використовується в багатьох частинах коду, конкретну частину легко знайти якщо врахувати аргументи, які використовуються у цільовому алгоритмі.

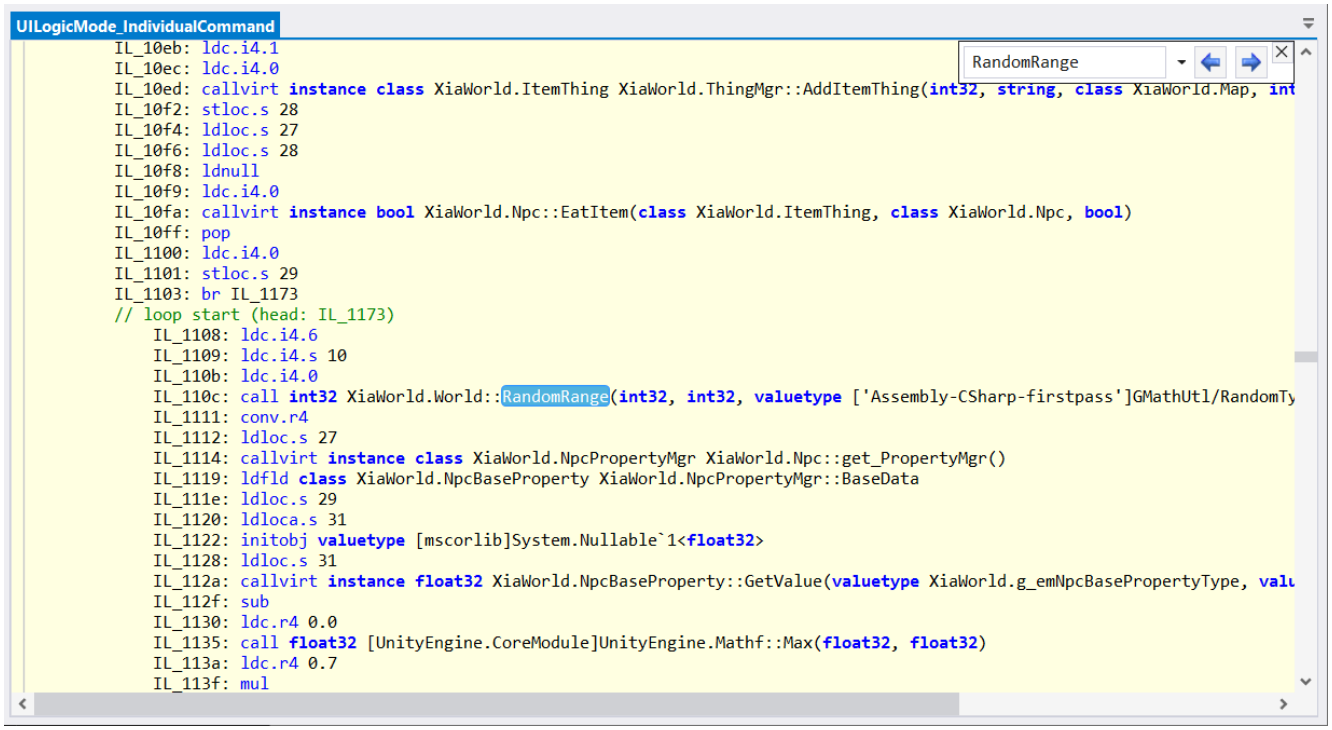

Рисунок 3.2.3 – Байт-код, відповідальний за функціонал Demigod Statue

Є декілька можливих варіантів модифікації. Одним з варіантів є видалення виклику функції генератора випадкових чисел і запису її аргументів на стек, після чого додається виконання запису довільного константного значення на стек. Іншим варіантом є проста заміна вхідних значень функції на однакові, що призведе до постійного вибору цього значення як «випадкового».

Необхідно взяти до уваги те, що дана функція викликається не лише в цьому конкретному коді. Зміна оригінального методу «World.RandomRange()» може призвести до нестабільної роботи продукту, оскільки цей метод викликається більш ніж 300 разів у різних частинах коду.

Було вирішено модифікувати значення аргументів, які використовуються для виклику функції, на однакові. Цього достатньо для виконання поставлених вимог і мінімізує обсяг змінюваного коду, що зменшить вірогідність внесення нестабільних змін до коду програми.

Змінювані фрагменти можна узагальнити наступним чином:

```
IL_1108: ldc.i4.#                                   Запис першого аргументу
IL_1109: ldc.i4.s #                                 Запис другого аргументу
IL_110b: ldc.i4.0                                   Запис третього аргументу
IL_110c: call int32 XiaWorld.World::RandomRange     Виклик функції
```

Варто зауважити, що третій аргумент у даних фрагментах завжди 0, оскільки третій вхідний аргумент методу «World.RandomRange()» належить до типу «GMathUtl.RandomType», однак цей аргумент має значення за замовчуванням, яке використовується у випадку надання лише цілих числових значень у коді C#, де третій аргумент можна не надавати. Однак після компіляції байт-коду, усі аргументи пропущені таким чином заміняються на нуль, який інтерпретується як використання значення за замовчуванням.

Отже, необхідно лише впровадити метод заміни перших двох інструкцій фрагменту на довільні значення. Ці значення можуть бути константними, або змінюваними через конфігураційний файл.

3.2.2 Конструювання коду модифікації

При розробці будь-якої модифікації варто мати на увазі, що залежно від використаної програмної технології та рівня оптимізації коду обраного

початковими розробниками – існуючий код може бути несприятливим до великих змін. Невеликі зміни на рівні індивідуальних інструкцій, наприклад заміну значень примітивних типів даних, легко впровадити за допомогою наявного функціоналу бібліотеки Harmony. Однак, якщо є необхідність у зміні виклику процедури, використанні додаткових циклічних конструкцій або більш глибокій зміні алгоритмів – нові частини коду краще реалізувати у новому класі, в той час як початковий код модифікується лише з метою інтеграції нового функціоналу. У даній предметній галузі бажано мінімізувати втручання у існуючий код, оскільки дуже важко передбачити ефект зміни можуть мати на стан усієї програми.

Програмна технологія .NET Framework є досить надійною, бібліотека Harmony вносить зміни досить проникливо, а необхідні зміни не є комплексними Враховуючи це, було прийнято рішення реалізувати процедурний пошук замінюваних фрагментів коду у процесі транспліювання.

Необхідно створити вхідний клас проекту, а також два додаткові класи, які будуть реалізувати модифікацію методів продукту. Ім’ям основного класу (а також простору імен модифікації) було обрано «BodyShapingMiracleDerandomizer». Усі класи є статичними, оскільки немає потреби у існуванні більш ніж одного екземпляра. Другорядні класи реалізують метод доступу до локальних змінних, які зберігають нові значення програми, а головний клас викликає ці методи зі значеннями, які будуть завантажуватись з конфігураційного файлу.

Після цього, відповідно до документації бібліотеки Harmony, необхідно виконати завантаження бібліотеки в асамблею програми. Це можна виконати за допомоги стандартної бібліотеки «System.Reflection» та методу «Assembly.LoadFrom(string)».

Ініціювання модифікації відбувається шляхом створення екземпляру класу Harmony, де вхідним аргументом конструктора є внутрішнє найменування модифікації. Метод «PatchAll()» новоствореного об’єкту почне процес модифікації.

Одночасно з цим, головний клас програми містить локальну змінну count, яка буде зберігати кількість змінених викликів методів програми. Оскільки розмір модифікації є невеликим а сама змінна не буде використовуватись для виконання

внутрішньої логіки програми, було вирішено не створювати додаткових методів для взаємодії з цією змінною. Замість цього було використано модифікатор доступу public. Таким чином другорядні класи програми зможуть напряму змінювати це значення в залежності від кількості змінених інструкцій.

Усе це відбувається в межах конструкції try. У випадку відсутності помилок у процесі модифікації викликається метод «KLog.Dbg(string)» продукту, що створює запис у журналі подій про успішне інтегрування нового коду та кількість змінених методів. У разі виникнення непередбачених помилок виконується запис про зміст помилки.

```
public static class BodyShapingMiracleDerandomizer {
    public static int count = 0;
    public static void Patch(int BodyMiracle_min, int BodyMiracle_max, int
    AlchemyStatue_min, int AlchemyStatue_max) {
        try {
            ToilAbsorbGong_Patch.SetValues(BodyMiracle_min, BodyMiracle_max);
            UILogicMode_IndividualCommand_Patch.SetValues(AlchemyStatue_min,
            AlchemyStatue_max);
            Assembly.LoadFrom(Path.Combine(Path.GetDirectoryName(Assembly.
            GetExecutingAssembly().Location), "0Harmony.dll"));
            Harmony harmony = new Harmony(
            "Remilian.BodyShapingMiracleDerandomizer");
            harmony.PatchAll();
            KLog.Dbg("[BodyShapingMiracleDerandomizer] Loaded! Lines affected: "
             + count.ToString() + ". New value ranges:");
            KLog.Dbg("[BodyShapingMiracleDerandomizer]    Shaping Miracles: " +
            BodyMiracle_min.ToString() + " - " + BodyMiracle_max.ToString());
            KLog.Dbg("[BodyShapingMiracleDerandomizer]    Remold: " +
            AlchemyStatue_min.ToString() + " - " + AlchemyStatue_max.ToString
            ());
        }
        catch (Exception e) {
            KLog.Dbg("[BodyShapingMiracleDerandomizer] Exception caught: " + e.
            ToString());
        }
    }
}
```

Рисунок 3.2.4 – Реалізація головного класу модифікації

Другорядні класи повинні реалізовувати як новий-байт код, так і пошук початкових фрагментів коду. Цей пошук можливий лише під час процесу трансплюювання, який надає доступ до повного списку інструкцій продукту. Для

цього необхідно використати анотації класів, а також реалізувати метод «Transplier()».

Анотація класу повинна містити інформацію про метод, який буде модифіковано, а також інформацію про клас, у якому даний метод міститься. Для позначення методу достатньо його найменування, але для позначення цільового класу бажано використати оператор typeof.

Інструкції байт-коду представлені класом «CodeInstruction» бібліотеки Harmony, а повний список можна отримати за допомогою будь-якого контейнера, що імплементує стандартний інтерфейс «IEnumerable». У даному проекті було використано контейнер List.

Оскільки ціллю даних класів є модифікування значень методу «World.RandomRange()», є доцільним виконання пошуку виклику даного методу у байт-коді. Ідентифікацію конкретних частин коду можливо виконати на основі змінних, які використовуються як аргументи виклику цього методу, які записуються на стек перед його виконанням. Для цього доцільно використати метод «CodeInstruction.Calls(MethodInfo)», який повертає значення типу bool в залежності від відповідності наданого методу до методу, що викликається відповідною інструкцією.

Аргумент MethodInfo доцільно створити використовуючи стандартний метод «System.Type.GetMethod()», який повертає інформацію про довільний метод у виконуваній асамблеї на основі найменування методу та, залежно від використаного варіанту, деякої додаткової інформації. Якщо метод є публічним та не містить перезавантажених варіантів, то достатньо лише найменування. Однак «World.RandomRange()» реалізовано у двох варіантах: один приймає два аргументи типу int, інший – два аргументи типу float. Модифіковані частини коду використовують лише перший варіант, тому необхідно надати додаткову інформацію. Для цього було обрано масив Type[], який буде містити набір типів, які відповідають типам аргументів бажаної реалізації шуканого методу.

```
[HarmonyPatch(typeof(ToilAbsorbGong), "OnStepToil")]
public static class ToilAbsorbGong_Patch {
    private static int min = 10;
    private static int max = 10;

    public static void SetValues(int newmin, int newmax) {
        min = newmin;
        max = newmax;
    }

    static IEnumerable<CodeInstruction> Transpiler(IEnumerable<CodeInstruction> instructions) {
        var codes = new List<CodeInstruction>(instructions);
        var entrymethod = typeof(World).GetMethod("RandomRange",
            new Type[] { typeof(int), typeof(int), typeof(GMathUtl.RandomType) }
            );
        int count = 0;
        for (int i = 0; i < codes.Count; i++) {
            if (codes[i].Calls(entrymethod)
                && (i - 3) >= 0
                && codes[i - 1].opcode == OpCodes.Ldc_I4_0
                && codes[i - 2].opcode == OpCodes.Ldc_I4_S
                && codes[i - 2].operand.Equals((SByte)10)
                && codes[i - 3].opcode == OpCodes.Ldc_I4_8) {
                //KLog.Dbg("[BodyShapingMiracleDerandomizer] Match found at: " + i.ToString());
                codes[i - 2].operand = (SByte)max;
                codes[i - 3].opcode = OpCodes.Ldc_I4_S;
                codes[i - 3].operand = (SByte)min;
                count++;
                //KLog.Dbg("[BodyShapingMiracleDerandomizer] Operands after change: " +
                codes[i - 3].operand.ToString() + "-" + codes[i - 2].operand.ToString());
            }
        }
        BodyShapingMiracleDerandomizer.count += count;
        return codes;
    }
}
```

Рисунок 3.2.5 – Реалізація додаткового класу «ToilAbsorbGong_Patch»

Модифікація байт-коду відбувається шляхом перебору усього списку інструкцій даного методу. У процесі цього перебору відбувається перевірка на виклик методу «World.RandomRange(int,int,RandomType)», після чого відбувається перевірка на відповідність використаних аргументів до тих, що були використані у цільовому фрагменті коду. При успішній перевірці відбувається заміна інструкції «ldc_i4_8», яка є оптимізованим варіантом запису конкретного цілого числа на стек, на «ldc_i4_s», що дозволяє запис довільного числа типу SByte. Також, операнди нових інструкцій заміняються на раніше отримані нові значення. При успішному знаходженні та заміні фрагменту коду – змінна count збільшується на 1. Як завершальний етап транспліювання дана змінна передається до головного класу, де вона буде використана для запису до журналу подій.

Метод «Apply2Thing()» модифікується аналогічно. Єдиними відмінностями є одна з інструкцій шуканого фрагменту («ldc_i4_6» замість «ldc_i4_8») та нові значення змінних, які встановлюються головним класом модифікації.

```
[HarmonyPatch(typeof(UILogicMode_IndividualCommand), "Apply2Thing")]
public static class UILogicMode_IndividualCommand_Patch {
    private static int min = 10;
    private static int max = 10;
    public static void SetValues(int newmin, int newmax) {
        min = newmin;
        max = newmax;
    }
    static IEnumerable<CodeInstruction> Transpiler(IEnumerable<CodeInstruction> instructions) {
        var codes = new List<CodeInstruction>(instructions);
        var entrymethod = typeof(World).GetMethod("RandomRange",
            new Type[] { typeof(int), typeof(int), typeof(GMathUtl.RandomType) }
            );
        int count = 0;
        for (int i = 0; i < codes.Count; i++) {
            if (codes[i].Calls(entrymethod)
                && (i - 3) >= 0
                && codes[i - 1].opcode == OpCodes.Ldc_I4_0
                && codes[i - 2].opcode == OpCodes.Ldc_I4_S
                && codes[i - 2].operand.Equals((SByte)10)
                && codes[i - 3].opcode == OpCodes.Ldc_I4_6)
            {
                //KLog.Dbg("[BodyShapingMiracleDerandomizer] Match found at: " + i.ToString());
                codes[i - 2].operand = (SByte)max;
                codes[i - 3].opcode = OpCodes.Ldc_I4_S;
                codes[i - 3].operand = (SByte)min;
                count++;
                //KLog.Dbg("[BodyShapingMiracleDerandomizer] Operands after change: " +
                codes[i - 3].operand.ToString() + "-" + codes[i - 2].operand.ToString());
            }
        }
        BodyShapingMiracleDerandomizer.count += count;
        return codes;
    }
}
```

Рисунок 3.2.6 – Реалізація класу «UILogicMode_IndividualCommand_Patch»

3.2.3 Ін’єкція коду за допомогою Lua та огляд створеної системи

Інтегрування модифікації у виконавчий код обраного програмного продукту потребує використання скриптованої мови програмування Lua. Інтерпретатор мови є вільно поширюваним. Редагування коду можливе з використанням будь-якого текстового редактора. У ході даної роботи був використаний редактор Notepad++.

Як було помічено у розділі 2.3.2: існують методи ін’єкції, які не потребують взаємодії з методами цільового продукту. Однак вони вимагають використання

сторонніх бібліотек, присутність яких збільшує розмір модифікації. З однієї сторони, якщо цільовий продукт має відкритий функціонал який можна використати замість збільшення обсягу модифікації – чому б не використати його. З іншої, якщо розробник вже ознайомлений з функціоналом додаткових бібліотек – їх використання може пришвидшити виконання проекту.

Оскільки ACS надає доступ до методів інтегрованої бібліотеки XLua – було вирішено використати її методи взаємодії з асамблеєю ACS для виконання початкової ін'єкції модифікації. Реалізацію примітивного функціоналу конфігураційного файлу також було вирішено виконати за допомогою Lua. Оскільки скриптові файли виконуються лише під час завантаження стандартних модифікацій ACS – також необхідно створити файл «info.json», присутність якого вимагається для розпізнання програми як модифікації, а також використовується сервісом цифрової дистрибуції Steam для поширення модифікацій серед користувачів.


```
{"Name":"BodyShapingMiracleDerandomizer","DisplayName":"Body Shaping Miracle
Derandomizer","GameVersion":1.22,"Version":1.1,"Desc":"[作者]\r\n
Remilian\r\n[简介]\r\nRemoves randomness from the body reshaping miracles such as
Third Eye, Marrow Sanitization, Charismatic, and Enlightened Awakening.\nGuarantees
the best roll each time.\n\n\nValues are changeable in a config file in 'saves'
folder.\nVanilla values: min=8, max=10.\nMod values: min=10, max=10.\nIf you want
to guarantee an average roll - change both values to 9.","Author":"Remilian",
"UIPackage":null,"ID":"2785210384","Flag":446983575,"Tags":["Function",
"ExcludeRealmsHeaven","Non-Chinese"],"IncreasedDesc":"1.0: Initial release",
"IsInsideCreat":false,"Previews":[],"ParentModInfo":null}
```


Рисунок 3.2.7 – Вміст файлу «info.json» модифікації

Ін'єкція здійснюється шляхом реалізації методу «OnBeforeInit()», який виконується ACS під час завантаження та виконання файлу «Main.lua» кожної модифікації. Даний метод має доступ до функціоналу XLua. Код даного методу виконує отримання доступу до необхідних класів асамблеї продукту, відновлення шляху до виконавчого файлу програми, завантаження бібліотеки модифікації згідно з відновленим шляхом, а також ініціювання процесу модифікації через головний клас створеної бібліотеки.

```
local BodyShapingMiracleDerandomizer = GameMain:GetMod("BodyShapingMiracleDerandomizer")
local settings = {BodyMiracleMin=10,BodyMiracleMax=10,AlchemyStatueMin=10,AlchemyStatueMax=10,Unsafe=0}
local setting_present = {}

function BodyShapingMiracleDerandomizer:OnBeforeInit()
  BodyShapingMiracleDerandomizer:LoadSettings()
  xlua.private_accessible(CS.XLua.LuaEnv)
  xlua.private_accessible(CS.XLua.ObjectTranslator)
  local thisData = CS.ModsMgr.Instance:FindMod("BodyShapingMiracleDerandomizer", nil, true)
  local thisPath = thisData.Path
  local mllFile = CS.System.IO.Path.Combine(thisPath, "BodyShapingMiracleDerandomizer.dll")
  local asm = CS.System.Reflection.Assembly.LoadFrom(mllFile)
  CS.XiaWorld.LuaMgr.Instance.Env.translator.assemblies:Add(asm)
  CS.BodyShapingMiracleDerandomizer.BodyShapingMiracleDerandomizer.Patch(
    settings["BodyMiracleMin"],settings["BodyMiracleMax"],
    settings["AlchemyStatueMin"],settings["AlchemyStatueMax"])
end
```

Рисунок 3.2.8 – Метод «OnBeforeInit()» файлу Main.lua

Реалізація взаємодії з файлом конфігурації виконується у методі «LoadSettings()», який викликається попередньо створеним методом. Зазвичай, рекомендовано використання популярних бібліотек, які надають можливість роботи з поширеними форматами збереження даних, наприклад JSON та XML. Однак, оскільки основною ціллю даного файлу є лише надання доступу до змінюваних значень модифікації більш просунутим користувачам – було вирішено не використовувати існуючі бібліотеки для його реалізації.

```
function BodyShapingMiracleDerandomizer:LoadSettings()
  local file = io.open(".\\saves\\BodyShapingMiracleDerandomizer.cfg", "r")
  if file == nil then
    BodyShapingMiracleDerandomizer:NewSettings()
  else
    local raw = file:read("*all");
    for file_key, file_value in string.gmatch(raw, "(%a+)%s*=%s*(%d+)") do
      for setting_key, setting_value in pairs(settings) do
        if file_key == setting_key then
          local value = tonumber(file_value)
          if value ~= nil then
            settings[setting_key] = file_value
          end
          setting_present[setting_key] = 1
          break
        end
      end
    end
    file:close()
    BodyShapingMiracleDerandomizer:UpdateSettings()
  end
  BodyShapingMiracleDerandomizer:Safety()
  return;
end
```

Рисунок 3.2.9 – Реалізація методу «LoadSettings()»

Метод «LoadSettings()» імплементує даний функціонал у досить простій формі: перевіряється наявність конфігураційного файлу у деякій директорії продукту. При його відсутності – цей файл створюється і в нього записується стандартні змінні модифікації. Ці змінні мають значення за замовчуванням, які об’явлені на початку файлу (рис. 3.2.8). Після відкриття та зчитування файлу використовується стандартний метод «string.gmatch(string,string)», який дозволяє поділити зміст файлу на частини відповідно до наданого регулярного виразу. Зчитані змінні порівнюються з очікуваними значеннями та використовуються у подальшому процесі модифікації.

Також реалізовано дві захисні функції для роботи з конфігураційним файлом. Метод «UpdateSettings()» порівнює змінні у збереженому файлі зі змінними, наявність яких очікується програмою. При відсутності змінної у файлі – новий запис з цією змінною додається до кінця файлу.

```
function BodyShapingMiracleDerandomizer:UpdateSettings()
  local file = io.open(".\\saves\\BodyShapingMiracleDerandomizer.cfg", "a")
  for key,value in pairs(settings) do
    local present = 0
    for key_p, value_p in pairs(setting_present) do
      if key == key_p then
        present = 1
        break
      end
    end
    if present == 0 then
      file:write("\n",key," = ",value)
    end
  end
  file:close()
  return
end

function BodyShapingMiracleDerandomizer:NewSettings()
  local file = io.open(".\\saves\\BodyShapingMiracleDerandomizer.cfg", "w")
  for key,value in pairs(settings) do
    file:write(key," = ",value,"\n")
  end
  file:close()
  return
end
```

Рисунок 3.2.10 – Реалізація методів «UpdateSettings()» та «NewSettings()»

Метод «Safety()» існує для зменшення вірогідності використання непридатних значень для модифікації. Після зчитування значень конфігураційного файлу отримані значення проходять через процес «затискання» між мінімальним та максимальним значеннями. Також, файл містить додаткову змінну «Unsafe». По замовчуванню ця змінна має значення 0. Якщо користувач змінив це значення на будь-яке інше – мінімальне та максимальне значення розширюються.

```
function BodyShapingMiracleDerandomizer:Safety()
  local _minBody = 8
  local _maxBody = 10
  local _minAlch = 6
  local _maxAlch = 10
  if settings["Unsafe"] ~= 0 then
    _minBody = 0
    _maxBody = 255
    _minAlch = 0
    _maxAlch = 255
  end
  settings["BodyMiracleMin"] = math.max(math.min(_maxBody,settings["BodyMiracleMin"]),_minBody)
  settings["BodyMiracleMax"] = math.max(math.min(_maxBody,settings["BodyMiracleMax"]),_minBody)
  settings["AlchemyStatueMin"] = math.max(math.min(_maxAlch,settings["AlchemyStatueMin"]),_minAlch)
  settings["AlchemyStatueMax"] = math.max(math.min(_maxAlch,settings["AlchemyStatueMax"]),_minAlch)
  if (settings["BodyMiracleMin"] > settings["BodyMiracleMax"]) then
    settings["BodyMiracleMin"], settings["BodyMiracleMax"] = settings["BodyMiracleMax"], settings["BodyMiracleMin"]
  end
  if (settings["AlchemyStatueMin"] > settings["AlchemyStatueMax"]) then
    settings["AlchemyStatueMin"], settings["AlchemyStatueMax"] = settings["AlchemyStatueMax"], settings["AlchemyStatueMin"]
  end
end
```

Рисунок 3.2.11 – Реалізація методу «Safety()»

Варто зауважити, що початкові мінімальне та максимальне значення відповідають тим, що були використані як початкові аргументи змінюваних викликів методу «World.RandomRange()». Вторинні значення відповідають можливим значенням типу SByte, який використовується для запису нових змінних інструкцією «ldc_i4_s».

Також можна помітити, що дана реалізація є досить громіздкою та містить код, який бажано відокремити як новий метод. Цей код був створений як тимчасовий та підлягав заміні, однак був випадково опублікований разом з оновленням модифікації. Оскільки цей метод не стосується основного функціоналу – було вирішено відкласти оновлення цієї частини до наступного оновлення основного функціоналу програми.

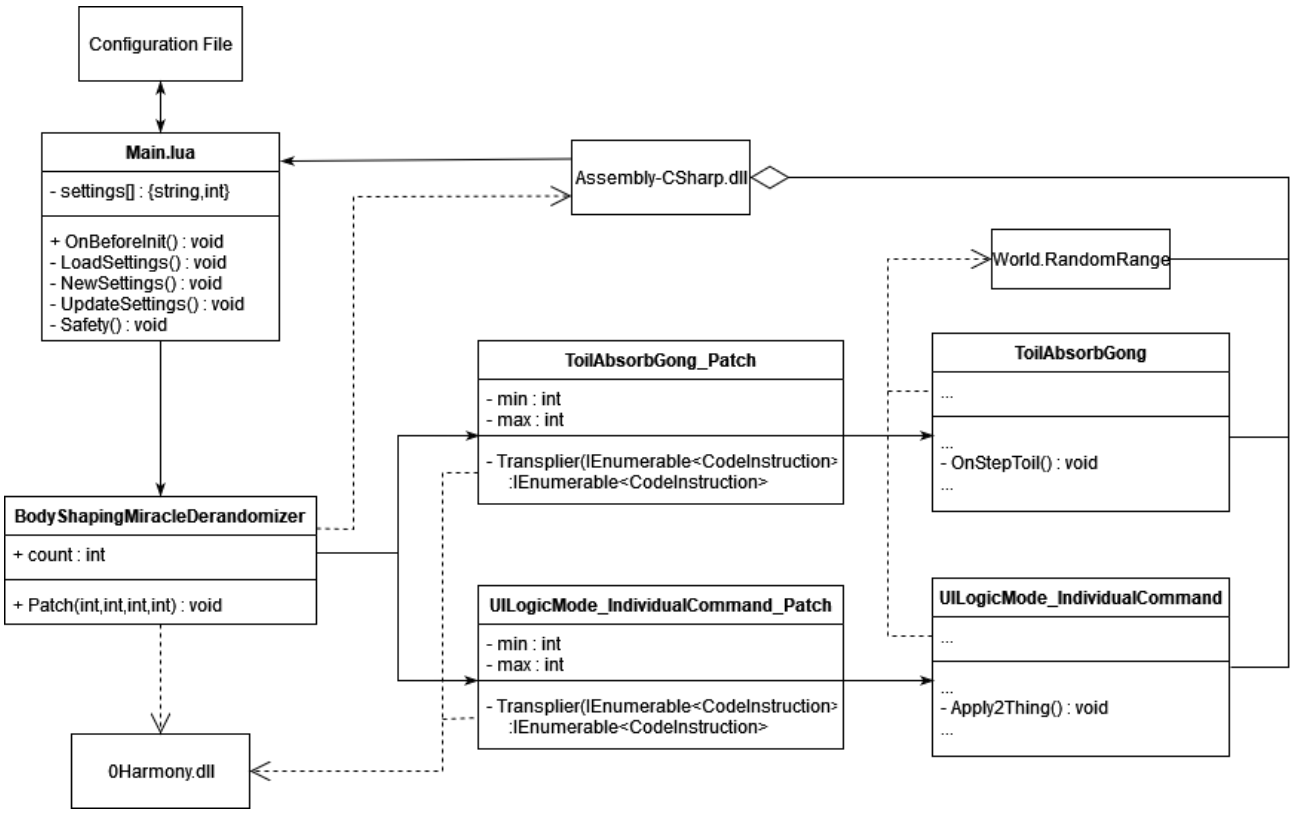

Рисунок 3.2.12 – Кінцева діаграма класів модифікації

Як можна побачити з даної діаграми, повний процес модифікації вимагає детального планування. У процесі реалізації спроектованої архітектури проекту необхідно адаптуватись до виявлених обмежень та нових вимог, які можуть з’явитись під час аналізу цільового програмного забезпечення. У порівнянні з початковою спрощеною діаграмою на рисунку 2.3.4 розділу 2.3.2 – комплексність проекту дещо збільшилась.

Послідовність виконуваних процедур можна описати за допомогою діаграми послідовності на рисунку 3.2.13. Дана діаграма описує лише частину виконуваного коду, який відповідальний за інтеграцію модифікації до ACS.

Під час виконання програми, більшість алгоритмів якої міститься у бібліотеці «Assembly-CSharp.dll», виконується завантаження та виконання файлів Lua, що належать до стандартних доповнень програми. Файл, відповідальний за завантаження модифікації, виконує зчитування даних з відповідного конфігураційного файлу.

Далі виконується метод бібліотеки Harmony, що ініціює проведення модифікацію байт-коду програми. Раніше зчитані дані використовуються під час ініціалізації «_Patch» класів проекту, які в свою чергу проводять пошук та модифікацію конкретних фрагментів байт-коду ACS. Після завершення інтеграції подальше виконання програми відбувається без змін.

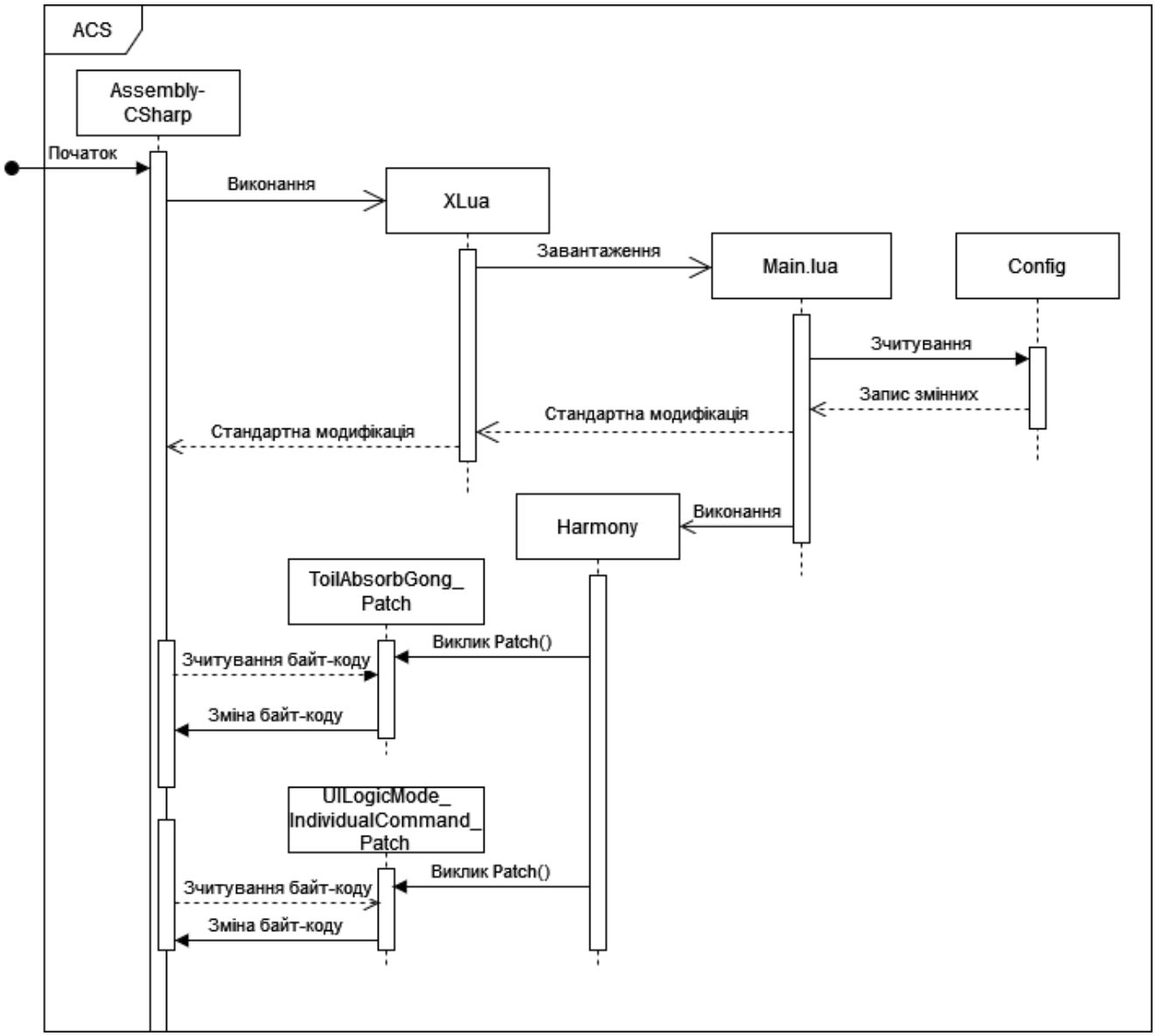

Рисунок 3.2.13 – Діаграма послідовності інтеграції модифікації

3.3 Тестування та оцінка якості

Тестування та оцінка якості модифікації відбувалась використовуючи два методи: перегляд журналу подій ACS та аналіз створених записів, та публікація модифікації за допомогою сервісу Steam з подальшим аналізом статистики використання та відгуків користувачів.

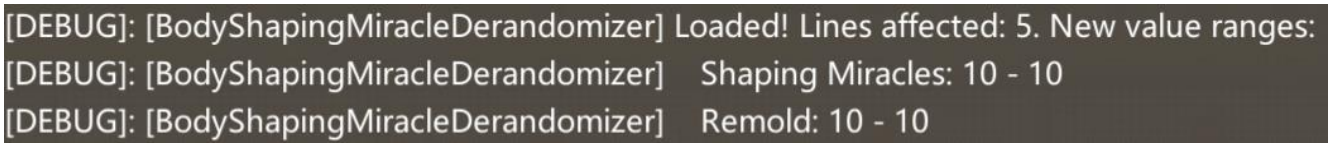

Рисунок 3.3.1 – Журнал подій ACS після завантаження модифікації

Відповідно до запису у журналі – модифікацію було успішно завантажено. П’ять стрічок, що були змінені, відповідають чотирьом стрічкам функціоналу Miracle та одній стрічці функціоналу Demigod Statue.

Поширення таких невеликих модифікацій, зазвичай, виконується використовуючи сторонні сервіси поширення програмного забезпечення. Часто такі сервіси або спеціалізуються ексклюзивно на модифікаціях, наприклад сервіси Mod.io та Nexusmods, або мають обширну підтримку поширення модифікацій, наприклад Steam Workshop. Для публікації даного продукту було обрано Steam, який також поширює ACS. На рисунку 3.3.2 можна побачити узагальнену взаємодію між розробниками модифікації та кінцевими користувачами. Сервіси поширення виконують посередницьку роль у даному обміні.

Також, контакт з великою кількістю користувачів надає можливість підтвердити коректну роботу функціоналу модифікації. Оскільки апаратне та програмне забезпечення користувачів варіюється – існує можливість, що створене програмне забезпечення буде несумісним з середовищем виконання кінцевого користувача. Однак використання програмної технології .NET Framework мінімізує цей ризик.

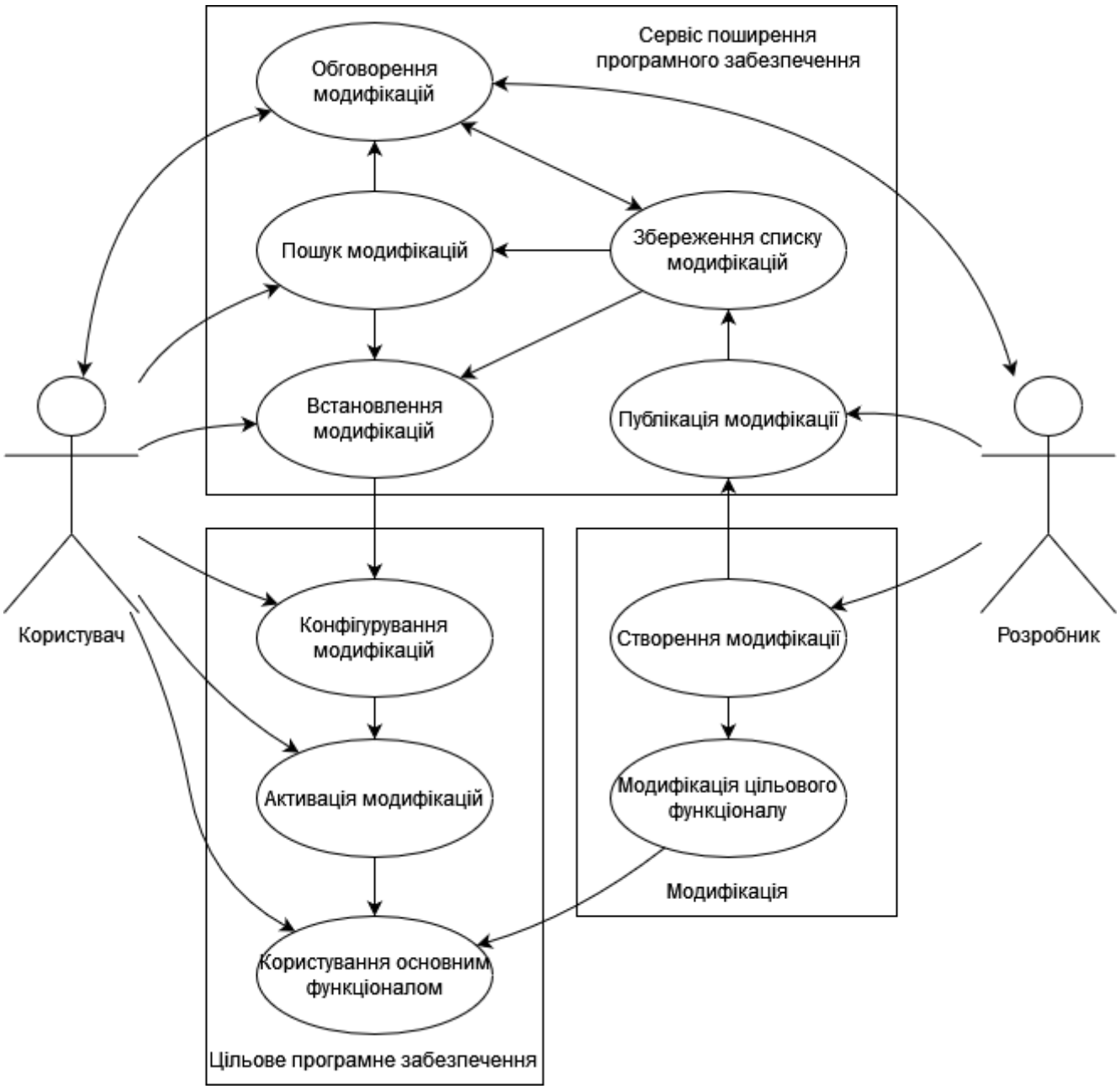

Рисунок 3.3.2 – Зв’язок між кінцевим користувачем та розробником при використанні сервісів поширення програмних продуктів

Відповідно до статистики, наданої сервісом Steam з часу публікації модифікації та на момент виконання даного проекту: 651 унікальних користувачів завантажило дану модифікацію, 596 з яких досі нею користуються [7]. Також відсутні коментарі щодо непрацюючого функціоналу або помилок при завантаженні. На основі цього можна скласти небезпідставний висновок, що функціонал реалізовано відповідно до поставлених вимог та без помилок.

Рисунок 3.3.3 – Статистика опублікованої модифікації

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Дослідження можливостей редагування байт-коду CIL здійснювалось з використанням портативних пристроїв та персональних комп'ютерів. Відповідно, в ході виконання експлуатації необхідно працювати з комп'ютерною технікою. Тому надзвичайно важливим фактором безпеки праці є дотримання правил користування технікою, норм та правил охорони праці.

Питання охорони праці регулюються певними законодавчими та нормативно-правовими актами, які, зокрема, визначають обов'язки роботодавця із забезпечення робітникам комфортних та безпечних умов для здійснення роботи. Ці обов'язки, а також права робітників на таких умовах праці передбачені частиною 2 ст.2 і частиною 1 ст.21 КЗпП, а також ст.13 Закону України «Про охорону праці» [8], у яких визначаються основні положення з реалізації конституційного права робітників. Існує цілий ряд вимог, які визначають специфіку заходів з охорони праці при роботі з персональним комп'ютером. Законодавчі та нормативно-правові акти, які за участі відповідних органів державної влади регулюють відносини між роботодавцем та робітником з питань безпеки, гігієни праці та виробничого середовища, а також встановлюють єдиний порядок організації охорони праці в Україні. На їх основі розроблені чисельні документи: правила, інструкції, норми, державні санітарні правила та інші нормативно-правові документи, якими мають керуватись роботодавці та які регламентують певні питання щодо конструкції електронно-обчислювальної техніки, та особливостей їх розміщення.

На сьогодні основними документами, які регламентують питання охорони праці при використанні працівниками персональних комп'ютерів, можна вважати наступні підзаконні акти:

- ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку» [9];

- ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» [10];
- НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [11];

У відповідності з цими актами, при розробці будь-якого програмного забезпечення, необхідно вжити всіх необхідних заходів з охорони праці та розробити відповідні документи.

Крім цього, в залежності від кількості працівників в ІТ компанії та від кількості працівників задіяних в проекті по розробці програмного забезпечення, служба охорони праці виглядає наступним чином:

- В ІТ компанії або проекті з кількістю працюючих 50 і більше осіб, роботодавець створює службу охорони праці відповідно до типового положення, що затверджується центральним органом виконавчої влади, що забезпечує формування державної політики у сфері охорони праці.
- В ІТ компанії або проекті з кількістю працюючих менше 50 осіб, функції служби охорони праці можуть виконувати в порядку сумісництва особи, які мають відповідну підготовку.
- В ІТ компанії або проекті з кількістю працюючих менше 20 осіб, для виконання функцій служби охорони праці можуть залучатися сторонні спеціалісти на договірних засадах, які мають відповідну підготовку.

Служба охорони праці згідно із законодавством підпорядковується безпосередньо роботодавцеві. Проте, роботодавець може доручити функціональне управління (кураторство) діяльністю служби іншій посадовій особі, скажімо, головному інженерові, заступникові директора з охорони праці тощо. Покладення таких обов’язків потрібно закріпити наказом або відобразити в посадовій інструкції уповноваженої особи. Робота служби охорони праці підприємства має здійснюватися відповідно до плану роботи та графіків обстежень, затверджених роботодавцем.

Функції служби охорони праці:

- Підготовка проектів наказів (розпоряджень) з питань охорони праці і внесення їх на розгляд роботодавцю.
- Проведення спільно з представниками інших структурних підрозділів і за участю представників професійної спілки підприємства або, за її відсутності, уповноважених найманими працівниками осіб із питань охорони праці перевірок дотримання працівниками вимог нормативно-правових актів з охорони праці.
- Проведення з працівниками вступного інструктажу з питань охорони праці та супутніх інструктажів.
- Ведення обліку та проведення аналізу причин виробничого травматизму, професійних захворювань, аварій на виробництві, заподіяної ними шкоди.
- Забезпечення належного оформлення і зберігання документації з питань охорони праці.
- Складання звітності з охорони праці за встановленими формами.
- Складання за участю керівників підрозділів підприємства переліків професій, посад і видів робіт, на які повинні бути розроблені інструкції з охорони праці, що діють в межах підприємства, надання методичної допомоги під час їх розроблення.
- Інформування працівників про основні вимоги законів, інших нормативно-правових актів та актів з охорони праці, що діють в межах підприємства.
- Розгляд питань про підтвердження наявності небезпечної виробничої ситуації, що стала причиною відмови працівника від виконання дорученої роботи відповідно до законодавства (у разі необхідності).
- Забезпечення підрозділів нормативно-правовими актами з охорони праці та актами з охорони праці, що діють в межах підприємства, посібниками, навчальними матеріалами з цих питань.

4.2 Безпека в надзвичайних ситуаціях

Радіоактивні речовини широко застосовуються в різних галузях промисловості, а також в науково-дослідних роботах. Тому важливого значення набувають знання характеру впливу іонізуючих випромінювань на організм людини та правил поводження з радіоактивними речовинами. Іонізуючі випромінювання, проникаючи в організм людини, можуть стати причиною важких захворювань. Можливі такі захворювання, як променева хвороба, лейкемія, злоякісні пухлини, захворювання шкіри. Можуть виникнути і генетичні наслідки, що приводять до захворювань в послідуючих поколіннях. Небезпека полягає в тому, що дія радіоактивних випромінювань не спостерігається до того часу, поки не з'явиться явне ушкодження або захворювання.

Система радіаційної безпеки вирішує дві функціональні задачі:

- Зниження рівня опромінення персоналу і населення до допустимих норм;
- Створення ефективної системи радіаційного контролю.

Для забезпечення радіаційної безпеки необхідно створювати такі умови використання радіоактивних речовин, при яких доза опромінення буде мінімальною і безпечною для здоров'я. Інакше кажучи, повинні бути прийняті всі міри для обмеження рівня опромінення персоналу і населення, але разом з тим забезпечені умови розвитку галузей енергетики, медицини та наукових досліджень.

Діючі "Норми радіаційної безпеки України" (НРБУ-97) включають систему принципів, критеріїв, нормативів та правил, виконання яких є обов'язковою нормою в політиці держави щодо забезпечення протирадіаційного захисту людини та радіаційної безпеки [12]. НРБУ-97 розроблені у відповідності до основних положень Конституції (254к/96-ВР) та Законів України "Про забезпечення санітарного та епідемічного благополуччя населення" (4004-12), "Про використання ядерної енергії та радіаційну безпеку" (39/95-ВР), "Про поводження з радіоактивними відходами" (255/95-ВР).

НРБУ-97 описують наступні основні принципи радіаційної безпеки:

- Неперевищення встановленої основної дозової границі опромінення;
- Зниження дози опромінення до можливого більш низького рівня;
- Виключення будь-якого необґрунтованого опромінення.

При встановленні основних дозових границь НРБУ-97 виділяють наступні категорії опромінених осіб:

- Категорія А - персонал (професійні працівники) - особи, які постійно або тимчасово працюють безпосередньо із джерелами іонізуючих випромінювань;
- Категорія Б - особи, які не працюють безпосередньо з джерелами випромінювання, але розташування робочих місць може попадати під вплив радіоактивних речовин та інших джерел випромінювання, які застосовуються на підприємствах і установах;
- Категорія В - населення області, краю, республіки, держави.

Для кожної категорії осіб встановлено три класи нормативів:

1. Основні дозові границі (ОДГ);
2. Допустимі рівні (ДР);
3. Контрольні рівні (КР).

Для категорії А в якості основної дозової границі встановлюється гранична допустима еквівалентна доза за рік (ГДД), а для категорії Б -границя еквівалентної дози за рік (ГД). Гранично допустима доза (ГДД) - найбільше значення індивідуальної еквівалентної дози за рік, яке при рівномірному впливі на протязі 50 років не викликає в стані здоров'я персоналу (категорія А) небажаних змін, що виявляються сучасними методами. Гранична доза (ГД) - гранична еквівалентна доза за рік для обмеженої частини населення (категорія Б); гранична доза встановляється меншою ніж ГДД. Вимірювання граничних доз виконується у біологічному еквіваленті раду (бер).

Таким чином, для осіб категорії А індивідуальна доза за рік не повинна перевищувати значення ГДД: 5 бер/рік для всього тіла та кісткового мозку; 150 бер/рік для м'язів, печінки, нирок, шлунково-кишкового тракту, легень, очей; 300 бер/рік для шкіряного покрову, рук, ніг, а для осіб категорії Б значення ГД: 0.5, 1.5, 30 бер/рік відповідно.

Сумарна доза, накопичена людиною до 30 років у всякому разі не повинна перевищувати 60 бер.

В зв'язку з тим, що при хронічному опроміненні в малих дозах (на рівні ГДД) біологічний ефект зумовлений тільки сумарною дозою опромінення, отриманого за багато років, норми регламентують тільки річну ГДД, тобто не накладають обмежень на рівень опромінення за робочий день, тиждень, квартал. Це означає, що дозволяється в необхідних випадках одноразове опромінення в дозі, яка рівна ГДД. Безумовно, що в подальшому необхідно організувати роботу цих осіб таким чином, щоб сумарна доза за рік не перевищувала ГДД.

Дозові норми, встановлені НРБУ-97, не враховують дозу, яку отримує пацієнт при медичному обстеженні, а також дозу, обумовлену природнім радіаційним фоном.

Природній радіаційний фон випромінювання - це іонізуюче випромінювання, яке складається з космічного випромінювання та випромінювання природніх радіоактивних речовин (на поверхні Землі, в атмосфері, в продуктах харчування, у воді, в організмі людини). Природній фон зовнішнього випромінювання складає 40...200мР/рік. Відзначимо, що індивідуальні дози опромінення персоналу підприємств та об'єктів атомної промисловості і ядерної енергетики в Україні всюди нижче встановленої ГДД. В працівників, які мають справу з джерелами іонізуючих випромінювань, середня доза утримується на рівні 0.1 ГДД, тобто 0.5 бер.

Організація роботи з радіоактивними речовинами та джерелами випромінювання виконується відповідно до основних санітарних правил забезпечення радіаційної безпеки України [13].

При роботі з радіоактивними речовинами і джерелами іонізуючих випромінювань першочергове значення має правильна організація праці, яка забезпечує безпеку обслуговуючого персоналу та всього населення в цілому. Правильно організувати роботу з радіоактивними речовинами - це означає передбачити комплекс заходів, що забезпечують радіаційну безпеку До таких заходів відносяться: захист від зовнішніх потоків випромінювання, відповідне планування і обробка приміщень, організація відповідного радіаційного контролю та санітарно-пропускного режиму, забезпечення необхідних умов транспортування радіоактивних речовин, збору та захоронення радіоактивних відходів, застосування засобів індивідуального захисту, тощо.

В залежності від річного споживання радіоактивних речовин підприємства поділяються на три категорії:

1. З річним споживанням більше 100 кг;
2. З річним споживанням від 10 до 100 кг;
3. З річним споживанням до 10 кг.

На обладнанні, контейнерах, транспортних засобах, приладах та апаратах, приміщеннях, призначених для роботи з радіоактивними речовинами і джерелами іонізуючих випромінювань повинні бути знаки радіаційної небезпеки. Адміністрація підприємства повинна розробити детальні інструкції щодо порядку проведення робіт, обліку, зберігання та видачі джерел випромінювання, збору та знищенню радіоактивних відходів, організації та проведенню радіаційного (дозиметричного) контролю. Всі працюючі повинні бути ознайомлені з цими інструкціями, навчені безпечним методам праці і зобов'язані скласти відповідний техмінімум. Всі особи, які поступають на роботу, повинні проходити попередній, а потім періодичні медичні обстеження.

При роботі із закритими джерелами, тобто радіоактивними джерелами випромінювання, устрій яких виключає попадання радіоактивних речовин в оточуюче середовище, персонал може бути опромінений тільки зовнішніми потоками випромінювання.

В залежності від умов випромінювання, характеру та місцезнаходження джерела застосовуються різні заходи та методи захисту від опромінення:

- Захист за часом ;
- Захист відстанню;
- Екранування джерел випромінювання;
- Індивідуальні засоби захисту;
- Радіопротектори.

Віддалення персоналу від джерела випромінювання особливо корисно, оскільки доза та її потужність обернено пропорційні квадрату відстані. Відстань на якій можна працювати з джерелами іонізуючого випромінювання на протязі певного часу, визначається по формулам з урахуванням допустимої потужності дози або дози опромінення, або по довжині пробігу випромінювання в повітрі. Для збільшення відстані між працюючим і джерелом випромінювання в лабораторній практиці широко застосовується дистанційне управління операціями з радіоактивними речовинами.

Безпеку робіт з іонізуючим випромінюванням забезпечують також захисні екрани, товщина яких розраховується на основі законів послаблення випромінювання у речовині екрану. Товщину захисного екрану можна визначити по довільним таблицям та номограмам.

Захист від внутрішнього опромінення вимагає виключення контакту із радіоактивними речовинами у відкритому вигляді, попередження попадання їх в середину організму, в повітря робочої зони, а також попередження радіоактивного забруднення рук, одягу, поверхонь, приміщень і обладнання.

ВИСНОВКИ

В ході даного дослідження було успішно використано бібліотеку Harmony для модифікування байт-коду програмного продукту Amazing Cultivation Simulator. Використаний інструметальний функціонал ILSpy дозволив ідентифікувати частини коду відповідальні за конкретні алгоритми.

Використані методи роботи вимагають детального аналізу модифікованого програмного забезпечення. Розмір проекту та методологія розробки використана початковими розробниками впливає на ефективність даного методу модифікації, оскільки пошук та ідентифікація необхідних компонентів може зайняти велику кількість часу. Однак, при наявності відповідних знань та навичок щодо використання операційних кодів CIL – сама модифікація є досить швидкою.

Задля ефективного виконання аналізу існуючого коду також необхідні обширні знання та навички у багатьох галузях інженерії програмного забезпечення: обізнаність з парадигмами проектування ПЗ, здатність розпізнавати стандартні конструкції та методи написання коду, ознайомленість з методами використання різних форматів збереження даних, компетентне використання наявного інструментарію, вміння пошуку та застосування документації використовуваних бібліотек та програмних технологій тощо.

Описані методи модифікації можуть бути використані у великій кількості ситуацій. Спеціалізоване програмне забезпечення, теоретично, може бути модифіковано з використанням подібної методології та відносно невеликою кількістю витрат. Ігровий сектор залишається найбільш впливовим у сфері модифікації програмного забезпечення, але ці ж самі методи та інструменти можуть бути використані у інших галузях.

Сучасні методи публікації програмного забезпечення дозволяють легко поширювати відносно невеликі за розміром модифікації серед великої кількості користувачів. Сервіси цифрової дистрибуції у поєднанні з тісними спільнотами дозволяють навіть початківцям швидко отримати відгуки та пропозиції щодо їх продуктів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Most used programming languages among developers worldwide as of 2022. URL: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/
2. Most used libraries and frameworks among developers, worldwide, as of 2022. URL: https://www.statista.com/statistics/793840/worldwide-developer-survey-most-used-frameworks/
3. CIL Instruction Set at C# Online.NET (CSharp-Online.NET). URL: http://en.csharp-online.net/CIL_Instruction_Set
4. Introduction to Harmony. URL: https://harmony.pardeike.net/articles/intro.html;
5. Kruchten, Philippe (2004-05-01). The Rational Unified Process: An Introduction. Addison-Wesley. ISBN 9780321197702
6. xLua programming solution for C# (Unity, .Net, Mono). URL: https://github.com/Tencent/xLua
7. Body Shaping Miracle Derandomizer, Steam Workshop. URL: https://steamcommunity.com/sharedfiles/filedetails/?id=2785210384
8. Закон України «Про охорону праці». URL: https://zakon.rada.gov.ua/laws/show/2694-12#Text
9. Постанова «Санітарні норми виробничого шуму, ультразвуку та інфразвуку». URL: https://zakon.rada.gov.ua/rada/show/va037282-99#Text
10. Постанова «Санітарні норми мікроклімату виробничих приміщень». URL: https://zakon.rada.gov.ua/rada/show/va042282-99#Text
11. Наказ «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text
12. Постанова «Про введення в дію Державних гігієнічних нормативів "Норми радіаційної безпеки України (НРБУ-97)"». URL: https://zakon.rada.gov.ua/rada/show/v0062282-97#Text

13. Наказ «Про затвердження державних санітарних правил "Основні санітарні правила забезпечення радіаційної безпеки України"». URL: https://zakon.rada.gov.ua/laws/show/z0552-05#Text
14. ECMA-335 Common Language Infrastructure (CLI). URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-335/
15. Harmony API Documentation. URL: https://harmony.pardeike.net/api/index.html
16. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 «Інженерія Програмного Забезпечення», М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк – Тернопіль: ТНТУ, 2020, 51с
17. Державний стандарт України. ДСТУ 3008-95 Документація. Звіти в сфері науки і техніки. Структура і правила оформлення
18. Науково-методичний посібник щодо змістовного наповнення основного розділу (Проєктування) кваліфікаційної роботи (другий освітній кваліфікаційний рівень «Магістр») для спеціальності 121 – інженерія програмного забезпечення, М.Р. Петрик, І.Я. Мудрик – Тернопіль: ТНТУ, 2022.
19. Comparison of solving algorithms for a mathematical model of filtration-diffusion transfer in the medium of spherical moisture-saturated microporous particles / Mykhaylo Petryk; Dmytro Mykhalyk; Oksana Petryk // Scientific Journal of TNTU. — Tern.: TNTU, 2021. — Vol 101
20. High-Performance Supercomputer Technologies of Simulation and Identification of Nanoporous Systems with Feedback for n-Component Competitive Adsorption / M. R. Petryk, I. V. Boyko, O. M. Khimich & M. M. Petryk // Cybernetics and Systems Analysis volume 57, pages 316–328 (2021)

ДОДАТКИ

ДОДАТОК А Технічне завдання

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку кваліфікаційної роботи
«Дослідження можливостей редагування байт-коду CIL з метою аналізу та модифікації алгоритмів скомпільованого програмного забезпечення»

Розробники:
виконавець ст. гр. СПм-61
Грибун Ігор Євгенович

_____
(підпис)

керівник кваліфікаційної роботи:
Петрик Михайло Романович

_____
(підпис)

Тернопіль 2022

ЗМІСТ

1. ПІДСТАВИ ДО РОЗРОБКИ

Розробка проводиться у відповідності до графіку навчального плану підготовки магістрів за спеціальністю 121 «Інженерія програмногозабезпечення».

Тема кваліфікаційної роботи: «Дослідження можливостей редагування байт-коду CIL з метою аналізу та модифікації алгоритмів скомпільованого програмного забезпечення».

Термін виконання: до «___» _____ 2022р.

2. ПРИЗНАЧЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Програмний продукт призначений для модифікування алгоритмів скомпільованого програмного забезпечення. Розробка даного продукту демонструє використання методології аналізу коду існуючого продукту. Використані методи публікації програмного забезпечення дозволяють оцінити техніко-економічний потенціал досліджень даної галузі.

3. ВИМОГИ ДО ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Функціональні вимоги

Система передбачає одну роль: користувач.

Користувач повинен мати змогу завантажити модифікацію використовуючи сервіс цифрової дистрибуції Steam. Користувач повинен мати доступ до конфігураційного файлу модифікації, який представлений текстовим файлом, який знаходиться у директорії кінцевого продукту.

3.2 Технічні вимоги

Вимоги до апаратного та програмного забезпечення користувача: операційна система, що підтримується програмною технологією .NET Framework, інтерпретатор байт-коду програмної технології .NET Framework, оригінал цільового програмного продукту.

Додаткове програмне забезпечення: сервіс цифрової дистрибуції Steam.

3.3 Програмні вимоги

Розробка системи виконується з використанням програмної технології .NET Framework та мов програмування C# та Lua.

Додаткова використана бібліотека: Harmony.

4. ЕТАПИ РОЗРОБКИ

Розробка інформаційної системи проводиться відповідно до наступного плану:

1. Аналіз предметної галузі;
2. Аналіз конкретних вимог системи;
3. Дослідження обраного інструментального функціоналу;
4. Проектування модифікації байт-коду;
5. Застосування методології аналізу алгоритмів;
6. Конструювання нового байт-коду;
7. Конструювання коду модифікації;

8. Реалізація ін’єкції коду;
9. Огляд створеної інформаційної системи;
10. Тестування та оцінка якості продукту.

Виконання кожного етапу проекту погоджується з керівником проекту.

5. СУПРОВІДНА ДОКУМЕНТАЦІЯ

Для інформаційної системи повинні бути розроблені наступні документи:

- Завдання на кваліфікаційну роботу;
- Пояснювальна записка до кваліфікаційної роботи;
- Презентація проекту;
- Рецензія на проект;
- Диск з проектом.

Пояснювальна записка до проекту оформляється згідно діючих вимог до нормоконтролю проектів.

6. ПОРЯДОК ЗДАЧІ ПРОЕКТУ

Розроблена інформаційна система повинна відповідати вимогами, що складаються з перерахованих у п.3.1 цього документу.

Для здачі проекту необхідно підготувати весь перелік документів зазначений у п.5 цього документу.

Приймання проекту проводиться спеціально створеною комісією в термін зазначені в п.1 цього документу.

7. ВІДМІТКИ ПРО ВИКОНАННЯ ЕТАПІВ ТА ЗМІНИ В ПРОЕКТІ

Назва етапу	Відмітка*
Аналіз предметної області	
Архітектура системи	
Проектування системи	
Використання системи	
Супровідна документація	

* відмітки про виконання етапу ставляться керівником проекту

ДОДАТОК Б Публікація у науковому виданні

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

X НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

7–8 грудня 2022 року

ТЕРНОПІЛЬ
2022

УДК 004.41
І. Грибун
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АНАЛІЗ ТА МОДИФІКАЦІЯ БАЙТ-КОДУ СКОМПІЛЬОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

UDC 004.41
I. Hrybun

ANALYSIS AND MODIFICATION OF BYTECODE IN COMPILED SOFTWARE

Модифікація існуючого програмного забезпечення є перспективним напрямком дослідження в сучасній сфері розробки програмних продуктів. Великий обсяг створюваного програмного забезпечення та мінливість вимог користувачів іноді призводить до ситуацій, у яких потреби клієнтів змінюються після завершення розробки. Зазвичай, клієнт може легко зробити запит до розробників щодо внесення змін. Однак, існують ситуації у яких це є або неможливим, або небажаним. У таких випадках найняття спеціаліста для внесення мінімальних змін є допустимим варіантом.

Поширення багатоплатформених та швидкодійних програмних технологій, наприклад .NET Framework або Mono, залишається досить суттєвим [1]. Оскільки вони використовують компіляцію до проміжного байт-коду – їх модифікація потребує спеціалізованих знань щодо аналізу та зміни їх алгоритмів.

Метою даного дослідження є створення загальної методології пошуку та аналізу коду, відповідального за функціонал існуючого програмного продукту, та використання методів зміни даного коду відповідно до потреб клієнта. Належне застосування цих методів у поєднанні з усвідомленням загальновживаних структур та парадигм програмування дозволяє набагато швидше орієнтуватись у коді великих проектів. Також важливим є використання ефективних інструментів відповідно до поставленої задачі: програмне забезпечення ILSpy є висококласним продуктом, ціллю якого є детальний аналіз байт-коду Common Intermediate Language, який використовується платформою .NET Framework [2].

Також доцільним є порівняння даної методології та систем порівняння математичних та програмних моделей [3]. Деякі методи використані у процесі аналізу цих моделей можуть бути перепрофільовані для аналізу структур скомпільованого байт-коду.

Література

1. Most used libraries and frameworks among developers, worldwide, as of 2022. URL: https://www.statista.com/statistics/793840/worldwide-developer-survey-most-used-frameworks/.
2. ILSpy: .NET Decompiler with support for PDB generation, ReadyToRun, Metadata (&more). URL: https://github.com/icsharpcode/ILSpy.
3. Petryk M., Mykhalyk D., Petryk O. Comparison of solving algorithms for a mathematical model of filtration-diffusion transfer in the medium of spherical moisture-saturated microporous particles. Scientific Journal of TNTU. Tern.: TNTU, 2021. Vol. 101. No. 1. P. 15–21.

ДОДАТОК В Лістинг коду

Лістинг 1: Main.lua

```
local BodyShapingMiracleDerandomizer =
GameMain:GetMod("BodyShapingMiracleDerandomizer")
local settings =
{BodyMiracleMin=10,BodyMiracleMax=10,AlchemyStatueMin=10,AlchemyStatueMax=10,Unsaf
e=0}
local setting_present = {}

function BodyShapingMiracleDerandomizer:OnBeforeInit()
  BodyShapingMiracleDerandomizer:LoadSettings()
  xlua.private_accessible(CS.XLua.LuaEnv)
  xlua.private_accessible(CS.XLua.ObjectTranslator)
  local thisData = CS.ModsMgr.Instance:FindMod("BodyShapingMiracleDerandomizer",
nil, true)
  local thisPath = thisData.Path
  local mllFile = CS.System.IO.Path.Combine(thisPath,
"BodyShapingMiracleDerandomizer.dll")
  local asm = CS.System.Reflection.Assembly.LoadFrom(mllFile)
  CS.XiaWorld.LuaMgr.Instance.Env.translator.assemblies:Add(asm)
  CS.BodyShapingMiracleDerandomizer.BodyShapingMiracleDerandomizer.Patch(
    settings["BodyMiracleMin"],settings["BodyMiracleMax"],
    settings["AlchemyStatueMin"],settings["AlchemyStatueMax"])
end

function BodyShapingMiracleDerandomizer:LoadSettings()
  local file = io.open(".\\saves\\BodyShapingMiracleDerandomizer.cfg", "r")
  if file == nil then
    BodyShapingMiracleDerandomizer:NewSettings()
  else
    local raw = file:read("*all");
    for file_key, file_value in string.gmatch(raw, "(%a+)%s*=%s*(%d+)") do
      for setting_key, setting_value in pairs(settings) do
        if file_key == setting_key then
          local value = tonumber(file_value)
          if value ~= nil then
            settings[setting_key] = file_value
          end
          setting_present[setting_key] = 1
          break
        end
      end
    end
    file:close()
    BodyShapingMiracleDerandomizer:UpdateSettings()
  end
  BodyShapingMiracleDerandomizer:Safety()
  return;
end

function BodyShapingMiracleDerandomizer:Safety()
  local _minBody = 8
  local _maxBody = 10
  local _minAlch = 6
  local _maxAlch = 10
  if settings["Unsafe"] ~= 0 then
    _minBody = 0
    _maxBody = 255
    _minAlch = 0
```

```
    _maxAlch = 255
  end
  settings["BodyMiracleMin"] =
math.max(math.min(_maxBody,settings["BodyMiracleMin"]),_minBody)
  settings["BodyMiracleMax"] =
math.max(math.min(_maxBody,settings["BodyMiracleMax"]),_minBody)
  settings["AlchemyStatueMin"] =
math.max(math.min(_maxAlch,settings["AlchemyStatueMin"]),_minAlch)
  settings["AlchemyStatueMax"] =
math.max(math.min(_maxAlch,settings["AlchemyStatueMax"]),_minAlch)
  if (settings["BodyMiracleMin"] > settings["BodyMiracleMax"]) then
    settings["BodyMiracleMin"], settings["BodyMiracleMax"] =
settings["BodyMiracleMax"], settings["BodyMiracleMin"]
  end
  if (settings["AlchemyStatueMin"] > settings["AlchemyStatueMax"]) then
    settings["AlchemyStatueMin"], settings["AlchemyStatueMax"] =
settings["AlchemyStatueMax"], settings["AlchemyStatueMin"]
  end
end

function BodyShapingMiracleDerandomizer:UpdateSettings()
  local file = io.open(".\\saves\\BodyShapingMiracleDerandomizer.cfg", "a")
  for key,value in pairs(settings) do
    local present = 0
    for key_p, value_p in pairs(setting_present) do
      if key == key_p then
        present = 1
        break
      end
    end
    if present == 0 then
      file:write("\n",key," = ",value)
    end
  end
  file:close()
  return
end

function BodyShapingMiracleDerandomizer:NewSettings()
  local file = io.open(".\\saves\\BodyShapingMiracleDerandomizer.cfg", "w")
  for key,value in pairs(settings) do
    file:write(key," = ",value,"\n")
  end
  file:close()
  return
end
```

Лістинг 2: BodyMiracleDerandomizer.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Reflection;
using System.Reflection.Emit;
using HarmonyLib;
using XiaWorld;



namespace BodyShapingMiracleDerandomizer {
    public static class BodyShapingMiracleDerandomizer {
        public static int count = 0;
```

```
        public static void Patch(int BodyMiracle_min, int BodyMiracle_max, int
AlchemyStatue_min, int AlchemyStatue_max) {
            try {
                ToilAbsorbGong_Patch.SetValues(BodyMiracle_min, BodyMiracle_max);
                UILogicMode_IndividualCommand_Patch.SetValues(AlchemyStatue_min,
AlchemyStatue_max);

Assembly.LoadFrom(Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly
().Location), "0Harmony.dll"));
                Harmony harmony = new
Harmony("Remilian.BodyShapingMiracleDerandomizer");
                harmony.PatchAll();
                KLog.Dbg("[BodyShapingMiracleDerandomizer] Loaded! Lines affected:
" + count.ToString() + ". New value ranges:");
                KLog.Dbg("[BodyShapingMiracleDerandomizer]    Shaping Miracles: "
+ BodyMiracle_min.ToString() + " - " + BodyMiracle_max.ToString());
                KLog.Dbg("[BodyShapingMiracleDerandomizer]    Remold: " +
AlchemyStatue_min.ToString() + " - " + AlchemyStatue_max.ToString());
            }
            catch (Exception e) {
                KLog.Dbg("[BodyShapingMiracleDerandomizer] Exception caught: " +
e.ToString());
            }
        }
    }


    [HarmonyPatch(typeof(ToilAbsorbGong), "OnStepToil")]
    public static class ToilAbsorbGong_Patch {
        private static int min = 10;
        private static int max = 10;

        public static void SetValues(int newmin, int newmax) {
            min = newmin;
            max = newmax;
        }

        static IEnumerable<CodeInstruction>
Transpiler(IEnumerable<CodeInstruction> instructions) {
            var codes = new List<CodeInstruction>(instructions);
            var entrymethod = typeof(World).GetMethod("RandomRange",
                new Type[] { typeof(int), typeof(int), typeof(GMathUtl.RandomType)
}
                );
            int count = 0;
            for (int i = 0; i < codes.Count; i++) {
                if (codes[i].Calls(entrymethod)
                    && (i - 3) >= 0
                    && codes[i - 1].opcode == OpCodes.Ldc_I4_0
                    && codes[i - 2].opcode == OpCodes.Ldc_I4_S
                    && codes[i - 2].operand.Equals((SByte)10)
                    && codes[i - 3].opcode == OpCodes.Ldc_I4_8) {
                    //KLog.Dbg("[BodyShapingMiracleDerandomizer] Match found at: "
+ i.ToString());
                    codes[i - 2].operand = (SByte)max;
                    codes[i - 3].opcode = OpCodes.Ldc_I4_S;
                    codes[i - 3].operand = (SByte)min;
                    count++;
                    //KLog.Dbg("[BodyShapingMiracleDerandomizer] Operands after
change: " + codes[i - 3].operand.ToString() + "-" + codes[i -
2].operand.ToString());


                }
```

```
            }
            BodyShapingMiracleDerandomizer.count += count;
            return codes;
        }
    }

    [HarmonyPatch(typeof(UILogicMode_IndividualCommand), "Apply2Thing")]
    public static class UILogicMode_IndividualCommand_Patch {
        private static int min = 10;
        private static int max = 10;
        public static void SetValues(int newmin, int newmax) {
            min = newmin;
            max = newmax;
        }
        static IEnumerable<CodeInstruction>
Transpiler(IEnumerable<CodeInstruction> instructions) {
            var codes = new List<CodeInstruction>(instructions);
            var entrymethod = typeof(World).GetMethod("RandomRange",
                new Type[] { typeof(int), typeof(int), typeof(GMathUtl.RandomType)
}
                );
            int count = 0;
            for (int i = 0; i < codes.Count; i++) {
                if (codes[i].Calls(entrymethod)
                    && (i - 3) >= 0
                    && codes[i - 1].opcode == OpCodes.Ldc_I4_0
                    && codes[i - 2].opcode == OpCodes.Ldc_I4_S
                    && codes[i - 2].operand.Equals((SByte)10)
                    && codes[i - 3].opcode == OpCodes.Ldc_I4_6)
                {
                    //KLog.Dbg("[BodyShapingMiracleDerandomizer] Match found at: "
+ i.ToString());
                    codes[i - 2].operand = (SByte)max;
                    codes[i - 3].opcode = OpCodes.Ldc_I4_S;
                    codes[i - 3].operand = (SByte)min;
                    count++;
                    //KLog.Dbg("[BodyShapingMiracleDerandomizer] Operands after
change: " + codes[i - 3].operand.ToString() + "-" + codes[i -
2].operand.ToString());
                }
            }
            BodyShapingMiracleDerandomizer.count += count;
            return codes;
        }
    }
}
```

Лістинг 3: info.json


```
{"Name":"BodyShapingMiracleDerandomizer","DisplayName":"Body Shaping Miracle
Derandomizer","GameVersion":1.23,"Version":1.1,"Desc":"[作者]\r\n    Remilian\r\n[
简介]\r\n[作者]\n    Remilian\n[简介]\nRemoves randomness from the body reshaping
miracles such as Third Eye, Marrow Sanitization, Charismatic, and Enlightened
Awakening.\nNow also includes Remold ability of a certain Demigod
Statue.\nGuarantees the best roll each time.\n\nValues are changeable in a config
file in 'saves'
folder.","Author":"Remilian","UIPackage":null,"ID":"2785210384","Flag":446983575,"
Tags":["Function","ExcludeRealmsHeaven","Non-Chinese","Non-
Chinese"],"IncreasedDesc":"1.1: Included Remold ability of a Demigod Statue of an
ascended disciple of Primordial Alchemy Law\n1.0: Initial
release","IsInsideCreat":false,"Previews":[],"ParentModInfo":null}
```

ДОДАТОК Г Диск з даними проекту