МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет ім. І. Франка
Факультет електроніки та комп'ютерних технологій
Кафедра оптоелектроніки та інформаційних технологій

Допустити до захисту
Завідувач кафедри
_____ проф. Кушнір О. С.
«____» _____ 2023 р.

Кваліфікаційна робота

Бакалавр
(освітній ступінь)

«АНАЛІЗ ПРОГРАМНОГО КОДУ ВІДЕОРУШІЯ UNITY»

Виконав:
студент IV курсу групи ФеП-41
спеціальності 121 — Інженерія
програмного забезпечення
_____ **Р. ПАРХОМЧУК**

Науковий керівник:
ас. Б. І. ГОРОН
_____
«____» _____ 2023 р.

Рецензент:
доц. Н. Є. ФТОМИН
_____

Львів 2023

Зміст

Анотація

У роботі проведено лінгвістичний аналіз комп'ютерного коду відеорушія Unity, написаного на мові програмування C#. Окрім основного референс-коду Unity, розглянуті сторонні бібліотеки, написані на основі відеорушія Unity. Досліджено виконання законів Ціпфа, Парето, Гіпса для окремих текстів: найбільшого тексту з бібліотеки, а також об'єднаного тексту, сформованого з усіх менших текстів референс-коду Unity. Крім того код Unity було розглянуто, як корпус текстів, досліджено його флуктуаційну поведінку та виконання закону Тейлора. Виявлено, що основні лінгвістичні залежності, які виконуються для текстів, написаних на природних мова, виконуються також для комп'ютерного коду.

Abstract

In the paper, a linguistic analysis of the computer code of the Unity video engine, written in the C# programming language, was carried out. In addition to the main Unity reference code, third-party libraries written based on the Unity video engine are considered. The implementation of the laws of Zipf, Pareto and Heaps for individual texts was investigated: the largest text from the library, as well as the combined text formed from all the smaller texts of the Unity reference code were used for these purposes. In addition, the Unity code was considered as a corpus of texts, its fluctuation behavior and the fulfillment of Taylor's law were investigated. It was found that the main linguistic dependencies, which are fulfilled for texts written in natural languages, are also fulfilled for computer code.

Розділ 1

Теорія

1.1 Закони Ціпфа

Закони Ціпфа описують статистичні закономірності в розподілі величини частоти виникнення певних подій. Ці закони були відкриті в 20-х роках XX століття економістом Вільямом Стенлі Ціпфом, який виявив, що розподіл частоти виникнення подій в багатьох сферах життя, таких як розміри міст, кількість слів у тексті чи доходи фірм, може бути наближено законом степені.

Перший закон Ціпфа, що частота виникнення події f (тут f — відносна частота, тобто $f = F/N$, де F — абсолютна частота слова, а N — кількість слів у тексті) зворотно пропорційна її рангу r. Формула цього закону виглядає наступним чином:

$$f(r) = \frac{A_1}{r^\alpha}, \tag{1.1}$$

де α — степеневий коефіцієнт, зазвичай $\alpha \approx 1$.

Другий закон Ціпфа пов'язує ймовірність випадання деякого слова p (або більш загально — деякої події) з її частотою, тобто, яка ймовірність у тексті натрапити на слово певної частоти:

$$p(f) = \frac{B}{F^\beta}, \tag{1.2}$$

де β — степеневий коефіцієнт. Зазвичай $\beta \approx 2$.

Закон Парето — інтегральна форма другого закону Ціпфа, де $P(F)$ — кумулятивна густина ймовірності (тобто ймовірність натрапити на слово з частотою меншою або рівною F:

$$P(f) = \frac{C}{F^k}, \tag{1.3}$$

тут k — степеневий коефіцієнт.

Наведені степеневі коефіцієнти, як відомо з теорії, пов'язані такими залежностями: $\beta = 1 + 1/\alpha$, а $k = 1/\alpha$.

1.2 Поправка Мандельброта

Відомо, що зазвичай закон Ціпфа 1.1 погано описує слова з низькими рангами (дуже часті), а також з високими (дуже рідкісні, наприклад, hapax legomena, слова, які трапляються один раз). Тому Мандельброт запропонував поправку до цього закону, виходячи з міркувань про деяку фрактальну природу текстів і мови. Закон Ціпфа-Мандельброта має вигляд:

$$f(r) = \frac{A_2}{(r_0 + r)^\alpha}, \tag{1.4}$$

де A_2, r_0 і α — константи.

1.3 Закон Гіпса

Емпіричний закон Г.С. Гіпса (H. S. Heaps) в комп'ютерній лінгвістиці встановлює зв'язок між обсягом документа і обсягом словника унікальних слів, які містяться у цьому документі. Здавалося б, що словник унікальних слів має насичуватися, і його обсяг повинен стабілізуватися при збільшенні обсягу тексту. Однак, за законом Гіпса, це не так.

Згідно з емпіричним законом Гіпса, для будь-якого тексту обсяг словника унікальних слів (позначений як V) залежить від кількості унікальних слів у цьому тексті (позначена як L) за формулою:

$$V(L) = DL^\gamma \tag{1.5}$$

де D і γ є емпіричними параметрами. Значення D зазвичай лежить у діапазоні від 10 до 100 для європейських мов, а значення γ знаходиться у діапазоні від 0.4 до 0.6.

Цей закон відображає те, що обсяг словника унікальних слів зростає пропорційно до потужності тексту, але не насичується. З іншими словами, зі збільшенням кількості унікальних слів у тексті, обсяг словника також зростає, і ця залежність підкріплюється емпіричними параметрами D і γ, які можуть бути різними для різних мов або текстових корпусів.

Закон Гіпса є одним зі способів моделювання і розуміння статистичних властивостей тексту в комп'ютерній лінгвістиці та обробці природної мови. Цей закон може бути використаний для оцінки обсягу словника унікальних слів в текстових корпусах, розробки лексичних ресурсів та інших завдань, пов'язаних з аналізом тексту.

1.4 Закон Тейлора

Існує два закони, які пов'язують середньоквадратичне відхилення від середнього значення певної характеристики тексту від його розмірів та середнього значення. Зазвичай це стосується частоти певного елемента в тексті, але також його застосовують і до розмірів словника V і позначають як dV. Тут ми звертатимемося саме до визначення цих законів через словник, оскільки таким чином ми їх використовуємо в роботі.

Так залежність середньоквадратичного відхилення розмірів словника dV від розмірів тексту L має такий вигляд:

$$dV \propto L^{\alpha_1}, \tag{1.6}$$

де α_1 — степеневий коефіцієнт. Зазвичай α_1 лежить між значенням 0.5 і 1.0, коли $\alpha_1 = 0.5$, то це означає, що в тексті нема далекосяжних кореляцій, якщо ж $\alpha_1 > 0.5$ — це вказує на наявність цих далекосяжних кореляцій. Рівняння 1.6 ще називають флуктуаційним скейлінґом.

Залежність середньоквадратичного відхилення розмірів словника dV від розмірів словника V описується через закон Тейлора:

$$dV \propto V^{\alpha_2}, \tag{1.7}$$

де α_2 — степеневий коефіцієнт. Подібно до флуктуаційного скейлінґу 1.6, якщо α_2 лежить між значенням 0.5 і 1.0, коли $\alpha_2 = 0.5$, то це означає, що в тексті нема далекосяжних кореляцій, якщо ж $\alpha_2 > 0.5$ — це вказує на наявність цих далекосяжних кореляцій.

1.5 Лінгвістичні дослідження текстів комп'ютерних програм

На сьогодні існує ряд праць, в яких з точки зору кількісної лінгвістики були проаналізовані тексти комп'ютерних програм. Зокрема [1] авторами досліджувалися системи, написані мовою програмування Java (Jena, Jakarta-Tomcat, Ant, Swing, jEdit, jetty, jHotDraw, DrJava, Protégé, Cocoon, JavaCC, JUnit), у [2] було розглянути одразу три мови програмування: Java, C та C++, у [3] ще більше — крім вищезгаданих також Smalltalk, Matlab, PHP, Haskell, OCaml, Scheme, а також мови розмітки HTML, TeX і XML. Саме у [3] мови програмування порівнювали не тільки самі по собі, але і з точки зору парадигм програмування, які за ними стояли. Хоча треба зауважити, що дослідження проводилися не так з перспективи саме лінгвістики, як з перспективи software science.

Натомість у [4] оптика вже більш подібна до оптики теорії складних систем, однак [4] стосується не слів і текстів, а розподілу розмірів файлів у різних системах.

Розділ 2

Методика

2.1 Unity

Unity - це популярний інтегрований розвитоковий середовище (IDE) для створення ігор та інтерактивних додатків. Воно надає розширені засоби для розробки, включаючи графіку, фізику, звук, штучний інтелект, анімацію та інші функціональні можливості. Unity підтримує різноманітні платформи, такі як ПК, консолі, мобільні пристрої і віртуальна реальність (VR), що робить його популярним в галузі ігрової розробки та суміжних сферах.

Основною методикою, пов'язаною з Unity, є розробка на основі компонентів (Component-Based Development). Ця методика передбачає побудову ігрових об'єктів з використанням компонентів, які представляють певну функціональну одиницю. Компоненти можуть включати логіку, візуальні ефекти, фізичні властивості, звук, управління та інші аспекти ігрового об'єкта. Розробка на основі компонентів дозволяє легко комбінувати і перевикористовувати функціональні елементи, спрощує процес розробки та забезпечує більшу гнучкість.

Основні складові методики Unity включають наступні аспекти:

- Сцена (Scene): Сцена є основним простором, в якому відбувається гра. Вона може містити ігрові об'єкти, фон, освітлення, камери та інші елементи. Сцени дозволяють організовувати та керувати різними етапами гри.

- Ігрові об'єкти (Game Objects): Ігрові об'єкти є базовими будівельними блоками гри. Вони можуть мати компоненти, які надають їм функціональність. Наприклад, ігровий об'єкт "Гравець"може мати компоненти для керування рухом, анімації та здоров'я.

- Компоненти (Components): Компоненти представляють функціональні одиниці, які можуть бути додані до ігрових об'єктів. Вони визначають поведінку, властивості та взаємодію об'єктів. Наприклад, компонент "Рух"може визначати спосіб переміщення ігрового об'єкта, а компонент "Модель фізики— властивості фізичної моделі об'єкта.

- Скрипти (Scripts): У Unity використовуються скрипти на мові програмування C# для створення логіки і взаємодії об'єктів. Скрипти можуть бути прикріплені до ігрових об'єктів і виконувати різні дії, такі як керування рухом, обробка подій, зміна параметрів тощо.

- Графічний інтерфейс (GUI): Unity надає засоби для створення графічного інтерфейсу, такого як кнопки, текстові поля, вікна тощо. Графічний інтерфейс може використовуватись для створення головного меню, налаштувань гри, ігрових інтерфейсів та інших елементів, які забезпечують взаємодію користувача з грою.

Це лише загальна оглядова інформація про методику Unity. Розробка ігор використовуючи Unity може бути досить складним і комплексним процесом, що вимагає знань програмування, дизайну і розуміння принципів геймдизайну. Якщо ви зацікавлені у глибшому вивченні Unity, рекомендую розглянути документацію Unity, онлайн-курси або літературу, присвячену розробці ігор на цій платформі. Open-source (відкритий код) означає, що програмне забезпечення, таке як Unity, має відкритий доступ до свого вихідного коду. Це означає, що користувачі можуть переглядати, змінювати і розповсюджувати код без обмежень, відповідно до умов ліцензії.

Для Unity це означає, що розробники мають можливість доступу до вихідного коду цього інтегрованого розвиткового середовища. Це дозволяє розробникам змінювати і адаптувати функціональність Unity під свої потреби. Вони можуть створювати власні розширення, плагіни, засоби розробки та інші додатки, що розширюють можливості Unity.

Open-source також означає, що існує велика спільнота розробників, які приєднуються до проекту Unity, вносять свої внески, допомагають вдосконалювати його та надають підтримку іншим користувачам. Це створює багато різних контриб'юторів, які працюють над вдосконаленням функціональності Unity, виправленням помилок, створенням нових функцій та інструментів.

Open-source підхід дозволяє розробникам співпрацювати, обмінюватися

знаннями та ідеями, розробляти плагіни та розширення, що сприяє швидкому росту і вдосконаленню Unity. Відкритий код також сприяє інноваціям та широкому застосуванню Unity в різних галузях, включаючи ігрову розробку, віртуальну реальність, доповнену реальність та інші інтерактивні додатки.

Загалом, використання open-source підходу для Unity дозволяє розробникам мати більшу свободу, гнучкість та контроль над розробкою своїх ігор та інтерактивних додатків, а також сприяє активному розвитку та вдосконаленню цього популярного розвиткового середовища.

До розгляду ми взяли кілька пакетів: сирцевий код `Unity` для референсів UnityCsReference, сирцевий код демонстраційного шутера від першого лиця FPS Sample, сирцевий код для рендеру за допомогою Unity через бравзер UnityRenderStreaming, сирцевий код демонстраційної гри для конференції Unity 2016 Tokyo AnotherThread, сирцевий код пакету для синхронізації розробки моделей у сторонніх програмах для створення цифрового контенту (DCC) з Unity в реальному часі MeshSync, сирцевий код програми для аналізу даних логів ProfileReader, сирцевий код системи для вторинних анімацій UnityChainSpringBone та сирцевий код демонстраційної гри для конференції Unite 2017 Tokyo WaveShooter.

2.2 Синтаксис мови програмування C#

При аналізі програмного коду відеоігор, зокрема відеоігор на платформі Unity, можуть зустрічатись різні мови програмування, залежно від конкретного проекту та використовуваних технологій. Основною мовою програмування для розробки ігор на платформі Unity є C#. Однак, також можуть бути використані інші мови та скриптові мови, зокрема:

C#: Це мова програмування, яка є основною для розробки ігор на Unity. C# використовується для створення скриптів, компонентів, логіки гри, взаємодії з об'єктами та багатьох інших аспектів розробки ігор в Unity.

JavaScript (UnityScript): Unity також підтримує JavaScript як одну з мов програмування для розробки ігор. В Unity використовується власна реалізація JavaScript, відома як UnityScript. Хоча UnityScript все ще підтримується, рекомендується використовувати C# як основну мову програмування, оскільки UnityScript вважається застарілим і його підтримка може бути припинена в майбутньому.

Shader languages: Для розробки шейдерів, які контролюють візуальний вигляд графіки в іграх, Unity використовує спеціальні мови програмування шейдерів, такі як HLSL (High-Level Shader Language) для платформ Windows і Xbox, GLSL (OpenGL Shading Language) для платформи OpenGL, або CG (C for Graphics) - мова програмування, яка підтримується різними графічними API.

Python: В деяких випадках Python може бути використаний для автоматизації рутинних завдань, створення скриптів для обробки даних, роботи з зовнішніми інструментами або розширення функціональності Unity за допомогою різних плагінів і бібліотек.

Інші мови: Залежно від конкретних потреб проекту, можуть бути використані інші мови програмування, такі як C++, Boo, Lua, Visual Basic, або навіть мови скриптів, які підтримуються окремими плагінами або фреймворками.

Вибір мови програмування для аналізу програмного коду відеоігор залежить від конкретної теми дослідження та потреб аналізу. Бажано вибрати мови програмування, які широко використовуються в індустрії геймдеву та мають достатню кількість ресурсів і документації для дослідження та аналізу.

C# — це універсальна багатопарадигмальна мова програмування, що включає строгу типізацію, лексичну область видимості, імператив, декларативний, функціональний, універсальний об'єкт-орієнтовані (на основі класів) та компонентно-орієнтовані дисципліни програмування. Він був розроблений Microsoft приблизно в 2000 році в рамках ініціативи .NET, а потім затверджений в якості міжнародного стандарту Ecma (ECMA-334) і ISO (ISO / IEC 23270: 2018). Mono - це назва безкоштовного проекту з відкритим вихідним кодом для розробки компілятора і середовища виконання для мови. C# є одним з мов програмування, розроблених для Common Language Infrastructure (CLI).

C# був розроблений Андерсом Хейлсбергом, а його командою розробників в даний час керує Мадс Торгерсен. Остання версія — 8.0, випущена в 2019 році разом з Visual Studio 2019 версії 16.3.

Під час розробки .NET Framework бібліотеки класів спочатку були написані з використанням компілятора системи керованого коду під назвою «Simple Managed C» (SMC). У січні 1999 року Андерс Хейлсберг сформував команду для створення нової мови під назвою Cool, який позначав «C-подібна об'єктно-орієнтована мова». Microsoft розглядала збереження назви «Cool» в якості остаточної назви мови, але вирішила не робити цього з причин, пов'язаних з товарними знаками. До того часу, коли проект .NET був публічно оголошено на конференції професійних розробників в липні 2000 року, мову був перейменований

в C#, а бібліотеки класів і середовище виконання ASP.NET були портований на C#.

Хейлсберг — головний дизайнер і головний архітектор C# в Microsoft, раніше займався проектуванням Turbo Pascal, Embarcadero Delphi (раніше CodeGear Delphi, Inprise Delphi і Borland Delphi) і Visual J++. У своїх інтерв'ю і технічних статтях він заявив, що недоліки в більшості основних мов програмування (наприклад, C++, Java, Delphi і Smalltalk) призвели до основ Common Language Runtime (CLR), що, в свою чергу, призвело до розробки сама мова C#. Джеймс Гослінг, який створив мову програмування Java в 1994 році, і Білл Джой, співзасновник Sun Microsystems, творця Java, назвали C# «імітацією» Java; Далі Гослінг сказав, що «C# - це свого роду Java з надійністю, продуктивністю і безпекою видалений». Клаус Крефт і Анджеліка Лангер (автори книги про потоках C ++) заявили у своєму блозі, що «Java і C#» це майже ідентичні мови програмування. Нудне повторення без інновацій », « Навряд чи хто-небудь буде стверджувати, що Java або C# є революційними мовами програмування, які змінили спосіб написання програм », а« C# багато запозичив у Java - і навпаки Тепер, коли C# підтримує упаковки і розпаковування, у нас буде дуже схожа функція в Java. "У липні 2000 року Хейлсберг сказав, що C#"не є клоном Java "і"набагато ближче до C ++ "по своєму дизайну. З часу випуску C# 2.0 в листопаді 2005 року мови C# і Java розвивалися за все більш і більш розбіжним траєкторіях, ставши двома зовсім різними мовами. Одним з перших основних відхилень стало додавання узагальнень до обох мов з абсолютно різними реалізаціями. C# використовує reification для надання «першокласних» універсальних об'єктів, які можна використовувати як будь-який інший клас, з генерацією коду, що виконується під час завантаження класу. Крім того, в C#, додано кілька основних функцій для програмування функціонального стилю, кульмінацією яких є розширення LINQ, випущені в C# 3.0, і підтримуюча його структура лямбда-виразів, методів розширення і анонімних типів. Ці функції дозволяють програмістам C# використовувати методи функціонального програмування, такі як замикання, коли це вигідно для їх застосування. Розширення LINQ і функціональний імпорт допомагають розробникам скоротити обсяг стандартного коду, який включається в загальні завдання, такі як запити до бази даних, аналіз файлу XML або пошук в структурі даних, зміщуючи акцент на реальну логіку програми, щоб поліпшити читаність і ремонтопридатність.

По своєму дизайну C# є мовою програмування, який безпосередньо відображає основну інфраструктуру спільної мови (CLI). Більшість його внутрі-

шніх типів відповідають типам значень, реалізованих середовищем CLI. Однак специфікація мови не містить вимог до генерації коду компілятора, тобто не вказує, що компілятор C# має орієнтуватися на середовище виконання спільної мови, або генерувати загальний проміжний мова (CIL), або генерувати будь-який інший конкретний формат. Теоретично, компілятор C# може генерувати машинний код, як традиційні компілятори C++ та Fortran.

C# підтримує строго типізовані оголошення неявних змінних з ключовим словом var, а також неявно типізовані масиви з ключовим словом new [], за яким слід ініціалізатор колекції.

C# підтримує суворий логічний тип даних, bool. Оператори, які приймають умови, такі як while і if, вимагають вирази типу, що реалізує оператор true, такого як логічний тип. Хоча C ++ також має логічний тип, він може вільно перетворюватися в цілі і з цілих чисел, а вирази, наприклад, якщо (a) вимагають тільки, щоб a був перетворений в bool, дозволяючи a бути int або покажчиком. C# забороняє цей підхід «цілочисельне значення істина або брехня» на тій підставі, що примус програмістів використовувати вирази, які повертають саме bool, може запобігти певні типи помилок програмування, таких як if (a = b) (використання присвоювання = замість рівності ==).

C# більш безпечний для типів, ніж C++. Єдиними неявними перетвореннями за замовчуванням є ті, які вважаються безпечними, наприклад, розширення цілих чисел. Це застосовується під час компіляції, під час JIT і, в деяких випадках, під час виконання. Не відбувається неявного перетворення між логічними значеннями і цілими числами, а також між членами перерахування і цілими числами (за винятком літерала 0, який може бути неявно перетворений в будь-який перераховується тип). Будь кероване перетворення повинне бути чітко позначено як явне або неявне, на відміну від конструкторів копіювання C ++ та операторів перетворення, які за замовчуванням неявні.

C# має явну підтримку коваріації і контраваріантності в універсальних типах, на відміну від C ++, який має деяку ступінь підтримки контраваріантності просто завдяки семантиці повертаються типів віртуальних методи.

Учасники перерахування розміщуються у своїй області видимості.

Мова C# не допускає глобальних змінних або функцій. Всі методи і члени повинні бути оголошені всередині класів. Статичні члени відкритих класів можуть замінювати глобальні змінні і функції.

Локальні змінні не можуть приховувати змінні вміщує блоку, на відміну від C та C++.

C# має підтримку строго типізованих покажчиків на функції через ключове слово делегат. Як і псевдо-C ++ фреймворк Qt, в C# є семантика, зокрема навколишнє події стилю публікації-підписки, хоча C# використовує для цього делегати.

Керована пам'ять не може бути звільнена; замість цього він автоматично збирається. Збірка сміття вирішує проблему витоків пам'яті, позбавляючи програміста від відповідальності за звільнення пам'яті, яка більше не потрібна.

На відміну від C ++, C# не підтримує множинне успадкування, хоча клас може реалізовувати будь-яку кількість інтерфейсів. Це було дизайнерське рішення провідного архітектора мови, щоб уникнути ускладнення і спростити архітектурні вимоги у всьому CLI. При реалізації декількох інтерфейсів, які містять метод з однаковою сигнатурою, і. тобто два методу з одним і тим же ім'ям, що приймають параметри одного й того ж типу в одному і тому ж порядку, C# дозволяє реалізовувати кожен метод в залежності від того, через який інтерфейс викликається цей метод, або, як Java, дозволяє реалізувати метод один раз і мати один виклик за викликом через будь-який з інтерфейсів класу.

Однак, на відміну від Java, C# підтримує перевантаження операторів. Тільки найбільш часто перевантажені оператори в C ++ можуть бути перевантажені в C#.

C# має можливість використовувати LINQ через .NET Framework. Розробник може запросити будь-який об'єкт IEnumerable <T>, документи XML, набір даних ADO.NET і бази даних SQL. Використання LINQ в C# дає такі переваги, як підтримка Intellisense, потужні можливості фільтрації, безпека типів з можливістю перевірки помилок компіляції і узгодженість даних для запиту з різних джерел. Є кілька різних мовних структур, які можна використовувати з C# з LINQ, і вони є виразами запитів, лямбда-виразами, анонімними типами, неявно типізованими змінними, методами розширення і ініціалізаторами об'єктів У серпні 2001 року корпорації Microsoft, Hewlett-Packard і Intel Corporation виступили співавторами в уявленні специфікації для C#, а також інфраструктури спільної мови (CLI) в організацію по стандартизації Ecma International. У грудні 2001 року ECMA випустила специфікацію мови ECMA-334 C#. C# став стандартом ISO у 2003 році (ISO / IEC 23270: 2003 Інформаційні технології. Мови програмування — C#). ECMA раніше прийняла еквівалентні специфікації як друге видання C# в грудні 2002 року.

У червні 2005 року ECMA схвалив видання 3 специфікації C# і оновило ECMA-334. Доповнення включали в себе часткові класи, анонімні методи,

нульові типи і універсальні шаблони (щось схоже на шаблони C ++). У липні 2005 року ECMA представила в ICO / МЕК JTC 1 за допомогою прискореного процесу останнього стандарти та відповідні ТЗ. Цей процес зазвичай займає 6-9 місяців.

Визначення мови C# і CLI стандартизовані у відповідності зі стандартами ISO і Ecma, які забезпечують розумну та недискримінаційну ліцензійну захист від патентних претензій.

Microsoft погодилася не пред'являти позов розробникам програмного забезпечення з відкритим вихідним кодом за порушення патентів в некомерційних проектах в частині структури, охоплюваній OSP. Microsoft також погодилася не застосовувати патенти на продукти Novell щодо платять клієнтів Novell, за винятком списку продуктів, в яких явно не згадується C#.NET або реалізація Novell .NET (The Mono Project). Тим не менш, Novell стверджує, що Mono не порушує жодних патентів Microsoft. Microsoft також уклала конкретну угоду про відмову у захисту патентних прав, пов'язаних з плагіном браузера Moonlight, який залежить від Mono, якщо він отриманий через Novell Microsoft очолює розробку еталонного компілятор C# з відкритим вихідним кодом і набору інструментів, що раніше носили кодова назва "Roslyn". Компілятор, який повністю написаний на керованому коді (C#), був відкритий, а функціональність відображена як API. Це дозволяє розробникам створювати інструменти рефакторінгу та діагностики.

Інші компілятори C# (деякі з яких включають реалізацію інфраструктури спільної мови і бібліотек класів .NET):

- Проект Mono надає компілятор C# з відкритим вихідним кодом, повну реалізацію загальномовної інфраструктури з відкритим вихідним кодом, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і майже повну реалізацію пропрієтарних бібліотек класів Microsoft .NET до .NET 3.5. Починаючи з Mono 2.6, планів по впровадженню WPF не існує; WF планується до більш пізнього випуску; і є лише часткові реалізації LINQ to SQL і WCF.

- Проект DotGNU (в даний час припинений) також надав компілятор C# з відкритим вихідним кодом, майже повну реалізацію інфраструктури спільної мови, включаючи необхідні бібліотеки інфраструктури, як вони зазначені в специфікації ECMA, і підмножина деяких залишилися пропрієтарних класів Microsoft .NET. бібліотеки .NET 2.0 (не документовані або

не включені в специфікацію ECMA, але включені в стандартний дистрибутив Microsoft .NET Framework). Загальна мовна інфраструктура загального ресурсу Microsoft під кодовою назвою «Rotor» забезпечує реалізацію спільного джерела середовища CLR і компілятора C#, ліцензованого тільки для освітніх і дослідницьких цілей, а також підмножина необхідних бібліотек інфраструктури Common Language Infrastructure в специфікації ECMA (вгору в C# 2.0 і підтримується тільки в Windows XP).

•

2.3 Токенізація тексту

За структурою програма складається з двох частин: `preprocessor.py` спершу обробляє програмний код таким чином, щоб позбутися всіх зайвих символів (для текстів природної мови це можуть бути знаки пунктуації, знаки нового рядка тощо) і виділити пробілами будь-що, що ми визначаємо як словоформу. Таким чином у випадку програмних кодів символи «.», «;», «» і тому подібні виділені пробілами, як окремі слова. Решта скриптів програми вже виходять з того, що текст був оброблений препроцесором і приведений до вигляду, в якому будь-що, що ми вважаємо словоформою, виділене пробілами з обох боків. Таким чином ми можемо змінювати препроцесори залежно від того, з яким текстом працюємо (до прикладу, для генних послідовностей чи MIDI-записів визначення «словоформ» будуть інакшими), але решту програми не доведеться модифіковувати. Так `static_characteristics.py` відповідає за перевірку законів Ціпфа, Парето та Гіпса, звісно, до кінця автоматизувати цю перевірку майже неможливо (адже ширина ціпфового режиму від тексту до тексту різниться), але можна завжди вже після побіжного такого аналізу оцінити, наскільки ці закони виконуються або ні. А програма `corpus_analyser.py` відповідає за перевірку законів, які стосуються флуктуації в корпусі текстів, зокрема, закону Тейлора 1.7. Поки що `corpus_analyser.py` досліджує тільки флуктуації словника, як цілого, але надалі плануємо додати також флуктуаційний аналіз окремих слів, заданих з консолі або через графічний інтерфейс. У коді використані методи сторонніх бібліотек, написаних на Python, основні з яких — pandas, matplotlib, numpy та scipy.

У цьому дослідженні ми повністю ігнорували коментарі тому, що, як показують [5] насправді не має сенсу змішувати дослідження двох, відмінних по

синтаксису та функціях мов: програмного коду та природної мови коментарів. Результату аналізу такої змішаної мови без чіткої нульової гіпотези складно осмислювати.

Розділ 3

Результати

3.1 Аналіз окремих текстів

У [1] спостережено, що найбільш частим ідентифікатором для Java є `String`, який займає 1.53 % тексту. Виходить, що саме класом `Sring` найбільше користуються програмісти, а отже — саме цей клас рекомендують як кандидата на оптимізацію.

На рис. 3.1 зображена частотно-рангова залежність слів для тексту `EditorGUI.c` частини програмного пакету `Unity Reference Code`. Червоною пунктирною лінією зображена апроксимація залежності законом Ціпфа 1.1, синьою — законом Ціпфа–Мандельброта 1.4. Для закону Ціпфа отримано $\alpha = 1.32$ і $A_1 = 0.524$ з коефіцієнтом кореляції Пірсона $r = 0.992$, для закону Ціпфа–Мандельброта — $\alpha = 1.39$, $A_2 = 0.634$ і $r_0 = 2.9$. Як і очікувалося, закон Ціпфа–Мандельброта доволі непогано описує залежність частоти від рангу для слів низьких рангів (високої частоти), проте недооцінює частоти для середніх та високих рангів. Натомість закон Ціпфа добре описує якраз середні ранги, низькі — дещо гірше, та все ж краще, ніж закон Ціпфа–Мандельброта. Співвідношення N_2/N_1 для `EditorGUI.csv` складає 63 %. Частка hapax legomena в усьому тексті становить $\sim 1.3\%$ (якщо брати кількість усіх слів, а не кількість унікальних слів, відносно унікальних слів hapax legomena складає частку у 29 %).

На рис. 3.2 зображено розподіл густини ймовірності для слів тексту `EditorGUI.csv`. Оригінальні частоти було біновано за допомогою естіматора Фрідмана-Діаконіса [6], після чого відкинуто всі біни, в які увійшло нуль частот. На виході отримано 150 точок. Червоною пунктирною лінією зображено апроксимацію розподілу густини ймовірності розподілом Ціпфа (другий закон Ціпфа) 1.2: $\beta = 1.644$, $B = 7.691 \times 10^{-9}$, коефіцієнт кореляції Пірсона $r = 0.929$.

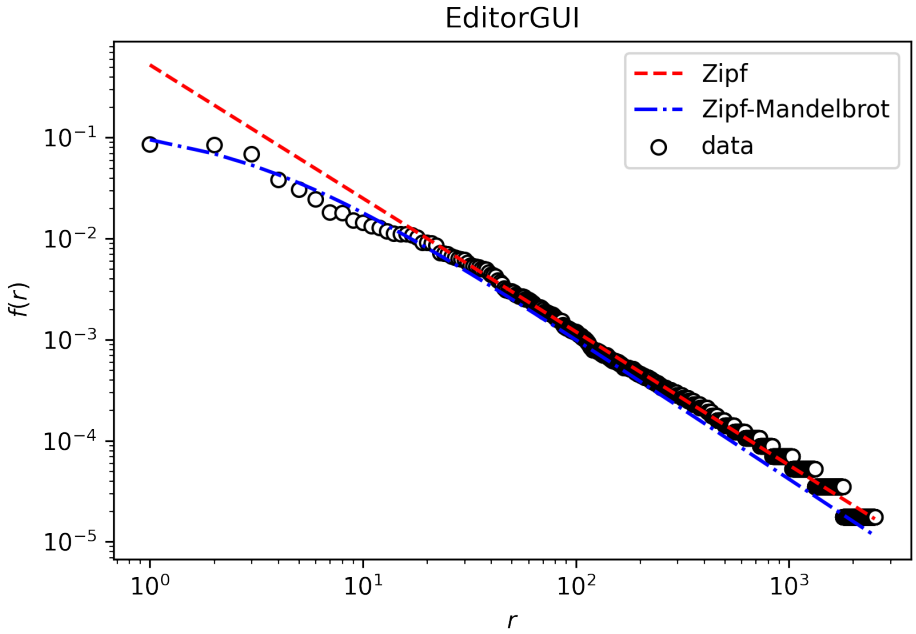

Рис. 3.1: Частотно-рангова залежність слів тексту `EditorGUI.csv`.

r	слово	частота
1	.	4862
2	(	4799
3	;	3885
4	=	2174
5	{	1741
6	position	1398
7	static	1032
8	label	1017
9	[	859
10	Rect	822

Табл. 3.1: Перші десять найбільш вживаних слів у `EditorGUI.csv`.

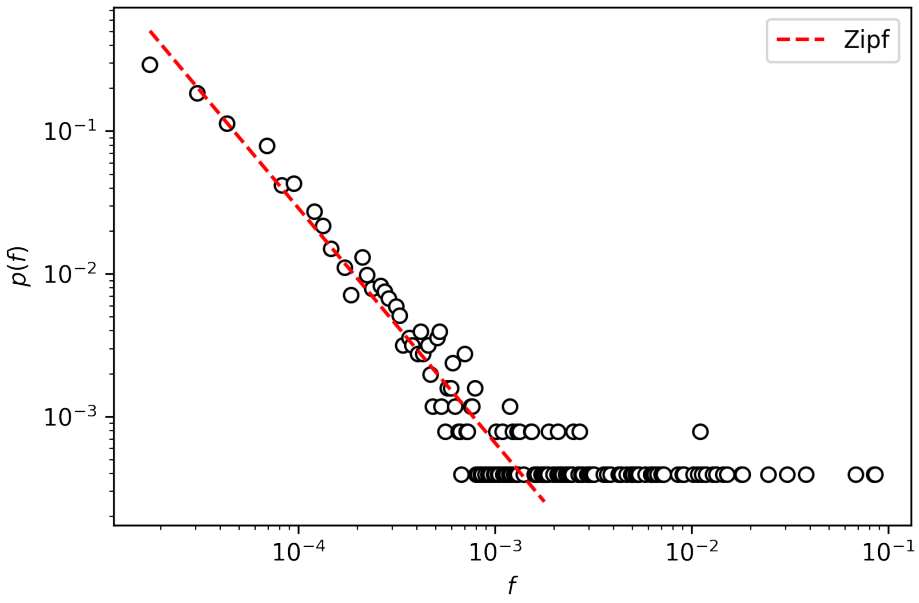

Рис. 3.2: Розподіл маси ймовірності для слів тексту `EditorGUI.csv`.

На рис. 3.3 зображено розподіл кумулятивної густини ймовірності для слів тексту `EditorGUI.csv`. Червоною пунктирною лінією зображено апроксимацію розподілу густини ймовірності розподілом Парето 1.3: $k = 0.842$, $C = 1.28 \times 10^{-4}$ і коефіцієнт кореляції Пірсона $r = 0.988$.

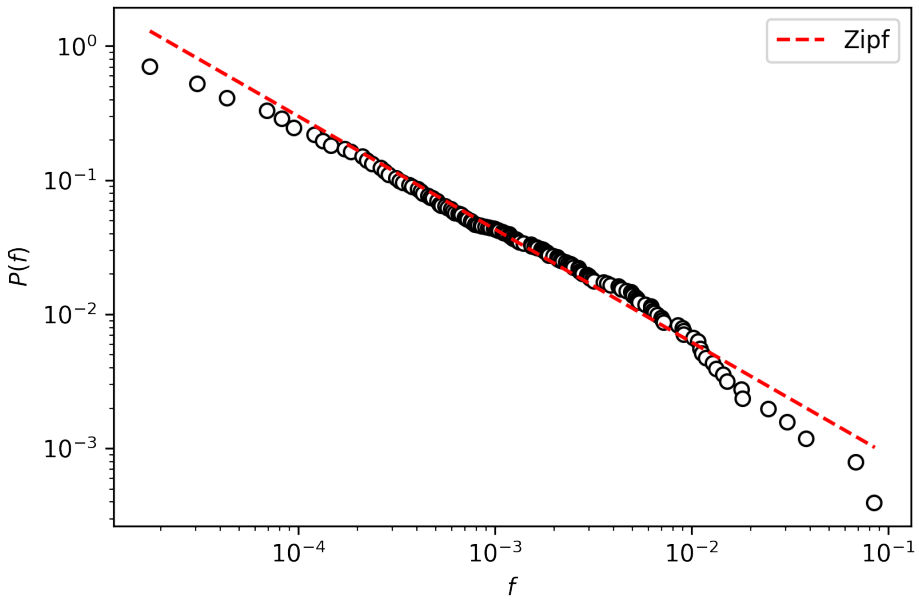

Рис. 3.3: Розподіл кумулятивної густини ймовірності для слів тексту `EditorGUI.csv`.

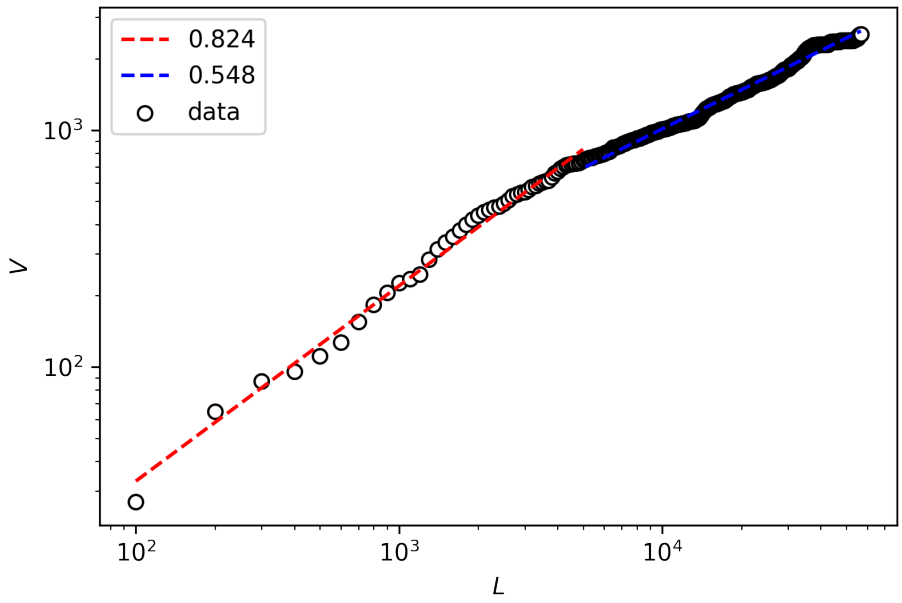

Рис. 3.4: Ріст словника V з ростом загальної довжини тексту L для `EditorGUI.csv`, червоним зображений режим швидкого росту з $\gamma \sim 0.8$, а синім — режим сповільненого росту з $\gamma \sim 0.57$.

На рис. 3.4 зображений ріст словника відносно росту довжини тексту для `EditorGUI.csv`. Раніші дослідження (зокрема див. [5]) розбігалися в однозначній оцінці степеня γ у законі Гіпса, подібно до природної мови він, щонайменше, варіюється від мови до мови. Проте тут ми навіть на довжині одного тексту не змогли отримати однозначну відповідь, тому розділили його два режими, які начебто видимі неозброєним оком: на початку словник росте доволі швидко, $\gamma \sim 0.8$, пізніше ріст доволі різко сповільнюється до того, що $\gamma \sim 0.57$. Подібну ситуацію описано в статті [7], яка стосується відхилень від законів Ціпфа та Гіпса у мовах з обмеженим словником (досліджувалися тексти, написані китайською, японською та корейською мовами). Автори виділяють два режими: лінійний приріст на початку, тоді логарифмічний приріст, який виходить на насичення. Подібні лінійний та логарифмічний режими можна побачити і на наших даних, якими розділили, хоч ми і апроксимували обидва режими прямою лінією, це було більше для ілюстрації їхньою відмінності.

Наслідуючи аналіз у [5] та [8] ми не зупинилися на аналізі одного окремого тексту, але також проаналізували увесь текст програмного пакету, як один текст. У [5] таким чином зокрема аналізували увесь текст пакету `Swing`, написаного на мові програмування Java. Це не знімає ніяких обмежень на гомогенність, розподіл на окремі тексти в рамках одного програмного пакету доволі умовний

і виникає здебільшого, як наслідок реорганізації коду таким чином, щоб його було зручно читати і розуміти іншим програмістам. Тому об'єднання програмного пакету в один текст в якомусь сенсі можна сприймати, як повернення до його природного вигляду.

На рис. 3.5 зображена частотно-рангова залежність для об'єднаного тексту, створеного зі всіх частин сирцевого коду рушія `Unity`. Апроксимація законом Ціпфа 1.1 показує, що для цього тексту $\alpha = 1.09$, $A_1 = 0.1183$ і коефіцієнт кореляції Пірсона $r = 0.987$. Апроксимація законом Ціпфа–Мандельброта 1.4 повертала не дуже реалістичні значення (наприклад, $\alpha \sim 3.4$), тому тут не наведена. Ми пов'язуємо таку невдачу з тим, що ціпфів лінійний режим набагато довший для такого великого тексту, ніж для меншого `EditorGUI.csv`, тому формула Ціпфа–Мандельброта зіткнулася з проблемами, усе ще доволі реалістично описуючи частотну поведінку слів з малими рангами.

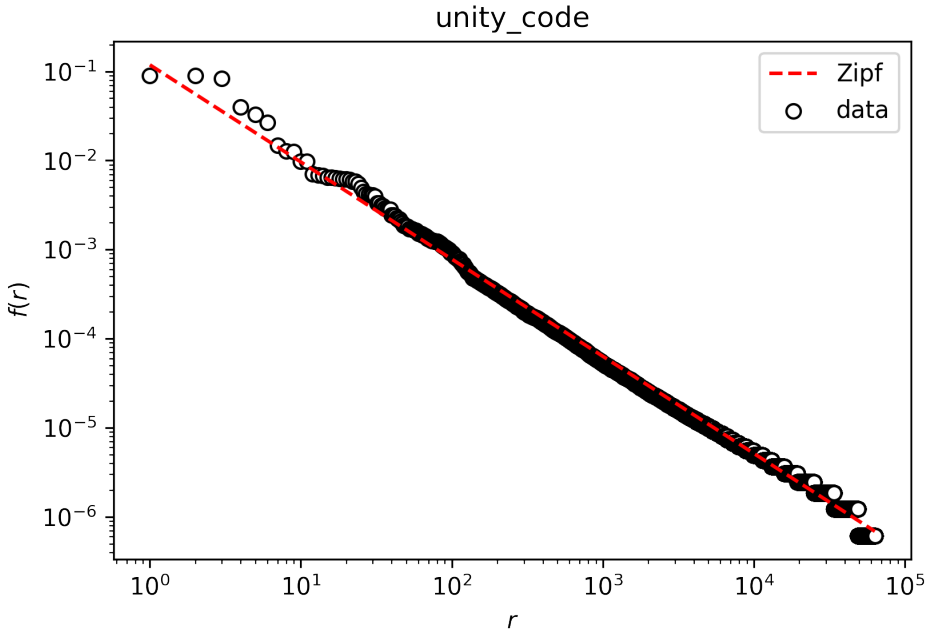

Рис. 3.5: Частотно-рангова залежність для об'єднаного `Unity`.

У табл. 3.2 наведені перші десять слів з найбільшою частотою в об'єднаному тексті `Unity`.

На рис. 3.6 зображений розподіл густини ймовірності об'єднаного тексту, створеного зі всіх частин сирцевого коду рушія `Unity`. Як і у випадку `EditorGUI.csv` бінування відбувалося за допомогою естіматора Фрідмана–Діаконіса [6], після чого біни з нульовою кількістю частот відкидалися. У результаті отримано 570 точок. Апроксимація формулою розподілу Ціпфа 1.2 дала $\beta = 1.99$, $B = 5.1083 \times 10^{-13}$ і коефіцієнт кореляції Пірсона $r = 0.969$.

r	слово	частота
1	.	146277
2	(	145567
3	;	134720
4	=	64696
5	{	53587
6	if	23928
7	public	20630
8	[	20524
9	return	15929
10	static	15857

Табл. 3.2: Перші десять найбільш вживаних слів в об'єднаному тексті `Unity`.

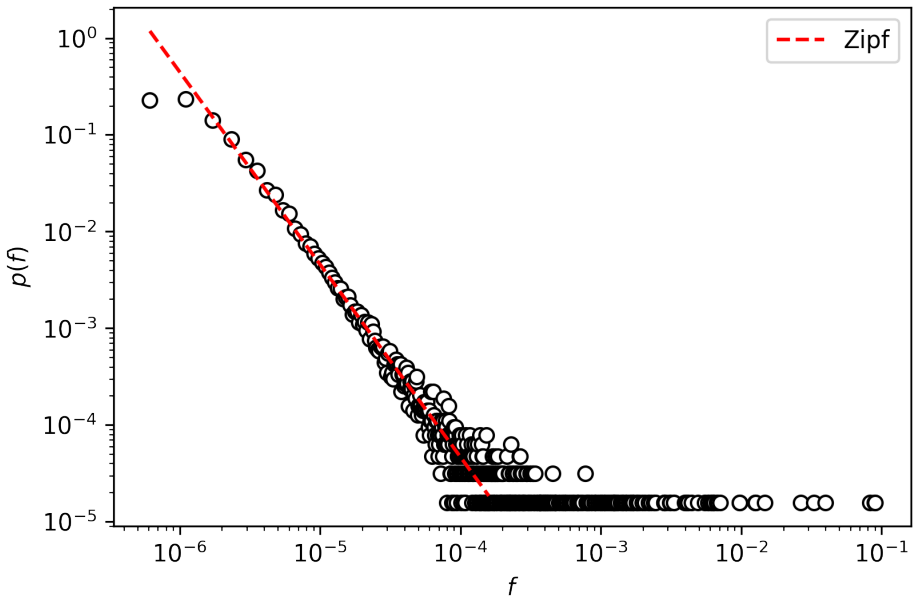

Рис. 3.6: Розподіл густини ймовірності слів з об'єднаного `Unity`.

На рис. 3.7 зображений розподіл кумулятивної густини ймовірності для об'єднаного тексту, створеного зі всіх частин сирцевого коду рушія `Unity`. Апроксимація формулою Парето 1.3 дала такі значення параметрів: $k = 0.913$, $C = 2.013 \times 10^{-6}$ і коефіцієнт кореляції Пірсона $r = 0.999$.

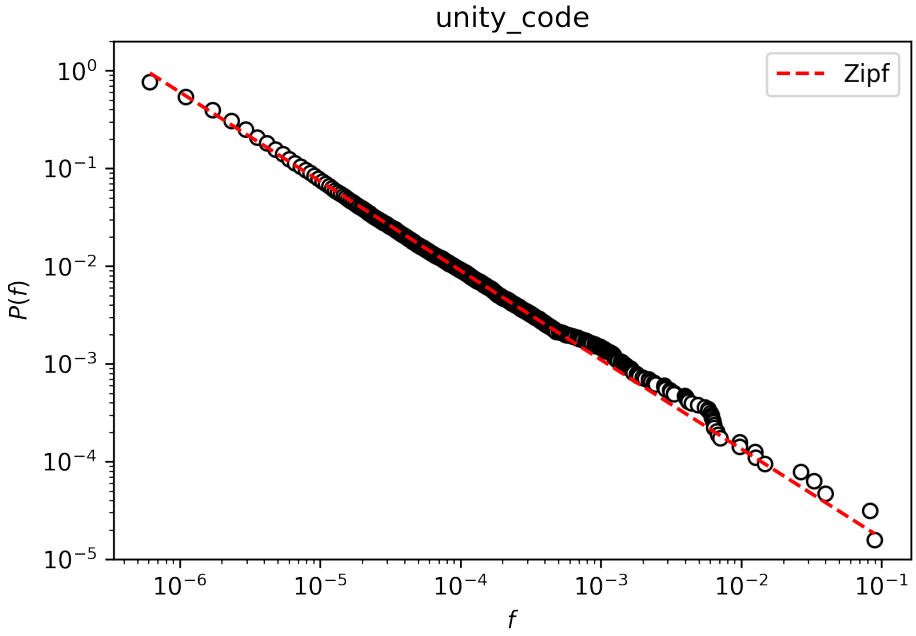

Рис. 3.7: Розподіл кумулятивної густини ймовірності слів з об'єднаного `Unity`.

Цікаво, що для об'єднаного тексту закон Гіпса (зображений на рис. 3.8) виконувався доволі добре і однозначно, окремих режимів росту не було виявлено. Десь так, як описувалося в роботі [5], з $\gamma \sim 0.8$ (що навіть більше, ніж отримано для Java) словник тексту програми росте швидше, ніж звично для природної мови (наприклад, для англійської γ становить порядку 0.6).

Як бачимо, зі збільшенням розмірів тексту параметри з формул 1.1, 1.2 та 1.3 наближаються до своїх значень, передбачених теорією. Хоча цікаво те, що вже і сам `EditorGUI.csv` доволі великий текст, проте, мабуть, недостатньо, щоб повертати статистично надійні дані.

3.2 Аналіз корпусу

У цій частині роботи ми йдемо слідами дослідження [9], в якому досліджувалися флуктуації частот лінгвістичних елементів у текстовій базі. Ми обрали тільки частину роботи, яка стосується не дослідження частоти окремого слова, а лише дослідження флуктуаційної поведінки розмірів словника унікальних слів,

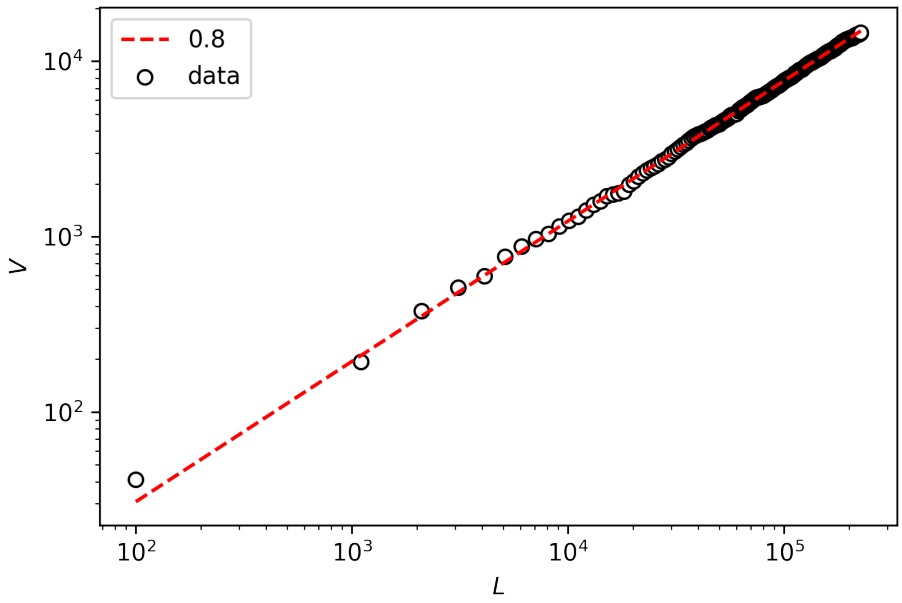

Рис. 3.8: Закон Гіпса для об'єднаного тексту `Unity`.

подібно як у [10]. На рис. 3.9 зображені емпіричні залежності розміру словника V від довжини тексту L для корпусу в подвійному логарифмічному масштабі. Для подальшого аналізу корпус був розбитий на біни по 50 текстів у кожному.

Так на рис. 3.10 зображена залежність розмірів середнього словника $\bar{V}$ від розмірів тексту L. Фактично, це той самий закон Гіпса, тому під час апроксимації ми користувалися тієї ж формулою 1.5. Цікаво, що отримали результат, відмінний від того, що спостерігали на прикладі об'єднаного тексту `Unity`, якщо там був $\gamma \sim 0.8$, то тут $\gamma = 0.627$ — менше значення, характерне для аналітичних мов. Ймовірно, що це пов'язано з тим, що в корпус входили не лише текст з бібліотеки `Unity`, а також з кількох інших бібліотек, перерахованих вище.

На рис. 3.11 зображена залежність середньоквадратичного відхилення від розмірів середнього словника dV від розмірів тексту L. Ми бачимо, що $\alpha_1 = 0.739$, тобто лежить між 0.5 та 1, що вказує на те, що в системі існують далекосяжні кореляції [9].

На рис. 3.12 зображена залежність середньоквадратичного відхилення від розмірів середнього словникав dV від розмірів середнього словника $\bar{V}$. Як відомо, ця залежність описується законом Тейлора 1.7. Також показник α_2 пов'язує між собою показники γ та α_1 таким співвідношенням: $\alpha_2 = \gamma/\alpha_1$. Як бачимо, $\alpha_2 = 1.18$ отримане як наслідок процедури апроксимації та таке «теоретично» оцінене $\alpha_2 = 1.16$ лежать доволі близько. Таке значення α_2 вказує на повторюваність певних граматичних структур у тексті, що доволі очікувано для тексту,

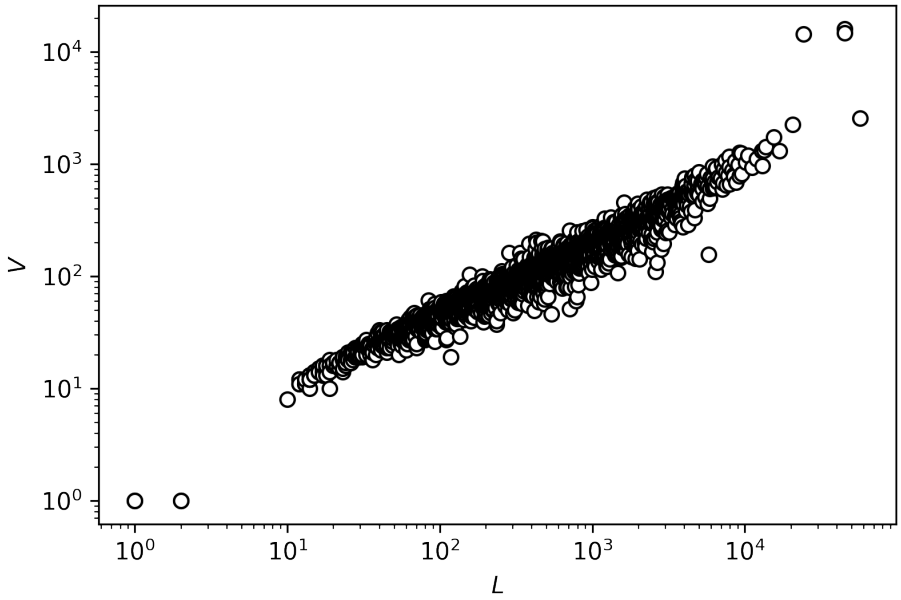

Рис. 3.9: Емпірична залежність $V(L)$ для текстів з корпусу

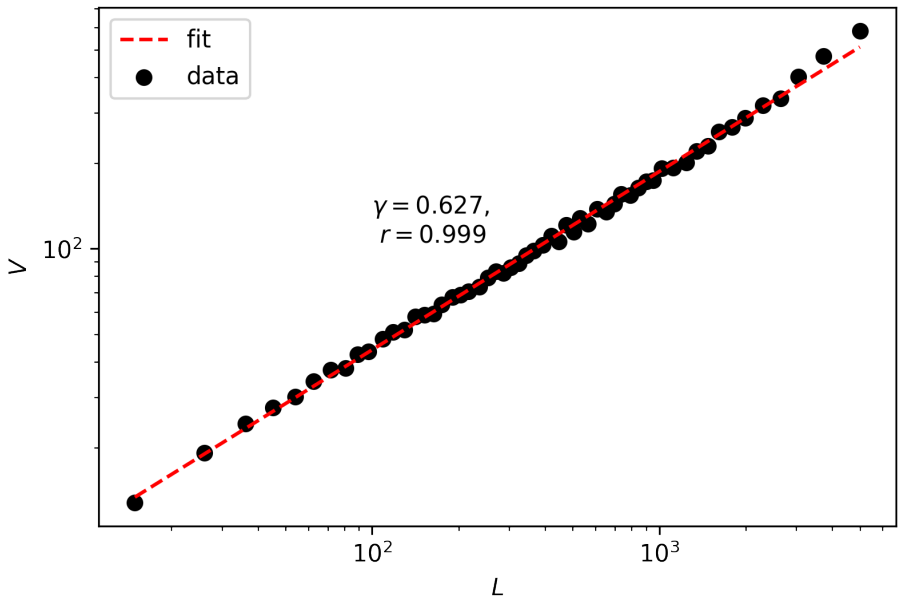

Рис. 3.10: Залежність $\bar{V}$ від L

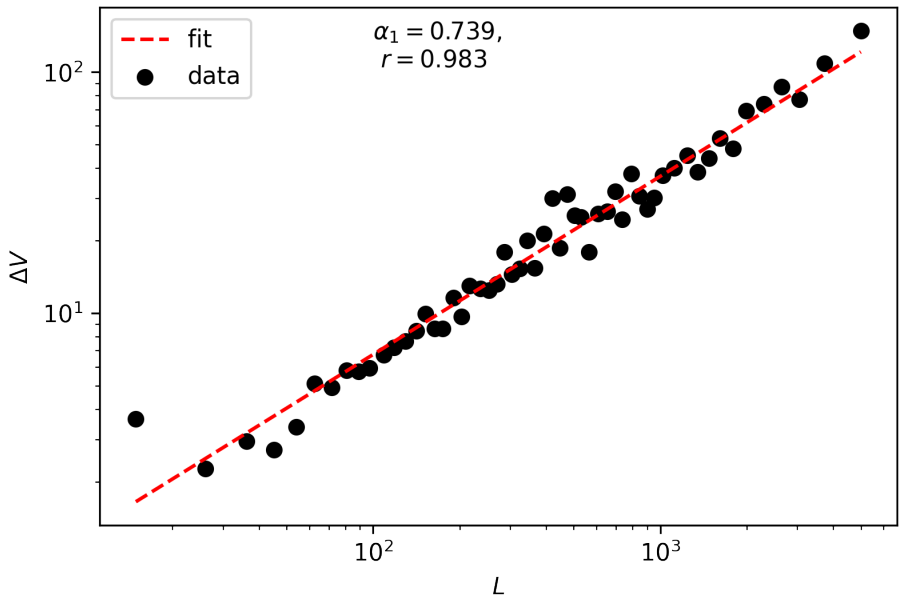

Рис. 3.11: Залежність dV від L

написаного мовою програмування. Зазвичай у цих мов не дуже гнучка граматика (вони і не потребують надто гнучкої граматики), відповідно, подібні граматичні патерни повсюди повторюються в текстах.

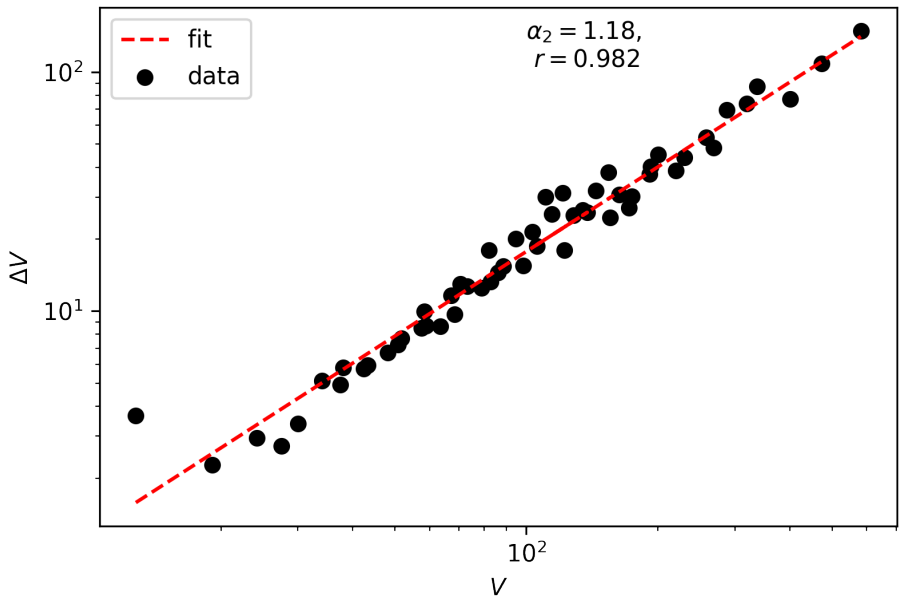

Рис. 3.12: Залежність dV від $\bar{V}$

Результати аналізу корпусу в короткій формі зібрані в таб. 3.3.

степінь	значення	коефіцієнт кореляції
γ	0.627	0.999
α_1	0.739	0.983
α_2	1.18	0.982

Табл. 3.3: Результати аналізу корпусу

Висновки

В ході виконання роботи було створено корпус текстів, який складався з основного референс-коду відеорушія Unity, а також кодів декількох сторонніх бібліотек, написаних для або на основі відеорушія Unity. Усі ці тексти написані мовою програмування C#, яка раніше було не надто досліджена з точки зору комп'ютерної лінгвістики (хоча деякі результати, отримані, наприклад, для мови програмування Java з деякою осторогою, але можна перенести на C# через загальну подібність цих мов). Загалом фінальний корпус складається з приблизно 2800 текстів.

Подальший аналіз складався з перевірки виконання законів Ціпфа, Парето та Гіпса для окремих текстів, а також флуктуаційного аналізу та перевірки виконання закону Тейлора для корпусу текстів. Для цього було написано код мовою Python, і загалом розроблено деякий проєкт екосистеми для лінгвістичного аналізу, яка не завершена, але її завершення не було метою роботи.

Аналіз показав, що закони Ціпфа, Парето, Гіпса і Тейлора виконуються для комп'ютерних текстів подібно, як і для текстів, які написані природною мовою. Цікавим виявився степеневий коефіцієнт зі закону Тейлора, який для корпусу комп'ютерних текстів становить приблизно 1.18, тоді як для природної мови його значення зазвичай лежить між 0.5 та 1. Таке значення вказує на те, що комп'ютерний код доволі регулярний і певні конструкції повторюються там зі сталою постійністю, що не дивно, адже для мов програмування не притаманна гнучкість синтаксичних конструкцій, тому ті не гнучкі конструкції, що існують, повторюються постійно.

Список використаних джерел

1. *Zhang H.* Exploring Regularity in Source Code: Software Science and Zipf's Law // 2008 15th Working Conference on Reverse Engineering. — IEEE, 10.2008. — С. 101—110. — DOI: 10.1109/WCRE.2008.37.
2. *Zhang H.* Discovering power laws in computer programs // Information Processing and Management. — 2009. — Т. 45. — С. 477—483. — DOI: 10.1016/j.ipm.2009.02.001.
3. *Pierret D., D. P.* An empirical exploration of regularities in open-source software lexicons // 2009 IEEE 17th International Conference on Program Comprehension. — IEEE, 05.2009. — С. 228—232. — DOI: 10.1109/ICPC.2009.5090047.
4. *Downey A.* The structural cause of file size distributions // SIGMETRICS '01: Proceedings of 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems. — 06.2001. — С. 328—329. — DOI: 10.1145/378420.378824.
5. Статистичні закономірності лінгвістики комп'ютерних програм / О. С. Кушнір [та ін.] // VIII українсько-польська науково-практична конференція ЕлІТ-2016. — 2016. — С. 90—95.
6. *Freedman D., Diaconis P.* On the Histogram as a Density Estimator: L_2 theory // Z. Wahrscheinlichkeitstheorie verw. Gebiete. — 1981. — Т. 57. — С. 453—476. — DOI: https://doi.org/10.1007/BF01025868.
7. *Lü L., Zhang Z.-K., Zhao T.* Deviation of Zipf's and Heap's Laws in Human Languages with Limited Dictionary Sizes // Scientific Reports. — 2013. — Т. 3. — С. 1082. — DOI: 10.1038/srep01082.
8. *Montemurro M.* Beyond the Zipf–Mandelbrot law in quantitative linguistics // Physica A: Statistical Mechanics and Its Applications. — 2001. — Т. 300. — С. 567—578. — DOI: 10.1016/S0378-4371(01)00355-7.

9. Флуктуації частот лінгвістичних елементів у текстовій базі / О. С. Кушнір [та ін.] // IX українсько-польська науково-практична конференція ЕлІТ-2017. — 2017. — С. 91—94.

10. *Gerlach M.*, *Altmann E.* Scaling laws and fluctuations in the statistics of word frequencies // New Journal of Physics. — 2014. — Т. 16. — С. 113010. — DOI: 10.1088/1367-2630/16/11/113010.