метадані

Заголовок
2023_ФеП-41с_ПАРХОМЧУК_Р.В._текст.pdf

Автор
Роман Пархомчук

Науковий керівник / Експерт
Богдан Горон

підрозділ
Факультет електроніки та комп'ютерних технологій

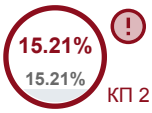
Перелік можливих спроб маніпуляцій з текстом

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	1478
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	28
Парафрази (SmartMarks)	a	141

Обсяг знайдених подібностей

Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



10
Довжина фрази для коефіцієнта подібності 2

7836
Кількість слів

52976
Кількість символів

Подібності за списком джерел

Прокручуйте список та аналізуйте, особливо, фрагменти, які перевищують КП 2 (позначено жирним шрифтом). Скористайтеся посиланням "Позначити фрагмент" та перегляньте, чи є вони короткими фразами, розкиданими в документі (випадкові схожості), численними короткими фразами поруч з іншими (мозаїчний плагіат) або великими фрагментами без зазначення джерела (прямий плагіат).

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	2021_ФеІ-41_Олексів_М-Д.Я._текст.pdf [redacted] 6/7/2021 The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)	125	1.60 %
2	2021_ФеІ-41_Олексів_М-Д.Я._текст.pdf [redacted] 6/7/2021 The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)	102	1.30 %
3	2021_ФеІ-41_Олексів_М-Д.Я._текст.pdf [redacted] 6/7/2021 The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)	91	1.16 %

4	2021_ФеІ-41_Олексів_М-Д.Я._текст.pdf [redacted] 6/7/2021 The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)	50	0.64 %
5	https://uk.pediatrendo.com/146922-c-sharp-programming-language-URLOWK	42	0.54 %
6	2021_ФеІ-41_Олексів_М-Д.Я._текст.pdf [redacted] 6/7/2021 The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)	42	0.54 %
7	https://uk.pediatrendo.com/146922-c-sharp-programming-language-URLOWK	38	0.48 %
8	https://uk.pediatrendo.com/146922-c-sharp-programming-language-URLOWK	37	0.47 %
9	https://uk.pediatrendo.com/146922-c-sharp-programming-language-URLOWK	37	0.47 %
10	https://uk.pediatrendo.com/146922-c-sharp-programming-language-URLOWK	34	0.43 %

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (6.80 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	2021_ФеІ-41_Олексів_М-Д.Я._текст.pdf [redacted] 6/7/2021 **The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)**	**516 (11)**	**6.58 %**
2	2022_ФеС_42_Ліпковський_НР_робота.pdf [redacted] 6/14/2022 **The Ivan Franko National University (Факультет електроніки та комп'ютерних технологій)**	**17 (2)**	**0.22 %**

з програми обміну базами даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з Інтернету (9.83 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	**https://uk.pediatrendo.com/146922-c-sharp-programming-language-URLOWK**	**539 (28)**	**6.88 %**
2	**https://ela.kpi.ua/bitstream/123456789/27033/7/Karpenko_magistr.pdf**	**74 (4)**	**0.94 %**
3	**https://essuir.sumdu.edu.ua/bitstream/123456789/86866/1/Mucharova_mag_rob.pdf**	**70 (4)**	**0.89 %**
4	**https://hongyihuang.wordpress.com/page/2/**	**38 (4)**	**0.48 %**
5	**https://krs.chmnu.edu.ua/jspui/bitstream/123456789/1968/1/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20401%20%D0%A7%D0%B5%D1%80%D0%BD%D0%B5%D0%BD%D0%BA%D0%BE%20%D0%9E%D0%BB%D0%B5%D0%BA%D1%81%D0%B0%D0%BD%D0%B4%D1%80%20%D0%AE%D1%80%D1%96%D0%B9%D0%BE%D0%B2%D0%B8%D1%87.pdf**	**27 (1)**	**0.34 %**
6	**https://www.maths.usyd.edu.au/u/pubs/publist/pubs2014.html**	**17 (1)**	**0.22 %**

7	https://github.com/rudrapratapsingh176/Matplotlib-and-seaborn/blob/main/Matplotlib.pyplot%20and%20Seaborn.py	5 (1)	0.06 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)

МІНІСТЕРСТВООСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет ім. І. Франка
Факультет електроніки та комп'ютерних технологій
Кафедра оптоелектроніки та інформаційних технологій
Кваліфікаційна робота
Бакалавр
(освітній ступінь)
«АНАЛІЗ ПРОГРАМНОГО КОДУ ВІДЕОРУШІЯ UNITY»
Виконав:
студент IV курсу групи ФеП-41
спеціальності 121
_____ Р. ПАРХОМЧУК
Науковий керівник:
ас. Б. І. ГОРОН

_____
«____» _____ 2023 р.
Рецензент:
доц. Н. Є. ФТОМИН

_____
м. Львів — 2023 р.

Зміст
Анотація 3
Abstract 4
1 Теорія 5
1.1 Закони Ціпфа 5
1.2 Поправка Мандельброта 6
1.3 Закон Гіпса 6
1.4 Закон Тейлора 7
1.5 Лінгвістичні дослідження текстів комп'ютерних програм 8
2 Методика 9
2.1 Unity 9
2.2 Синтаксис мови програмування C# 11
2.3 Токенізація тексту 17
3 Результати 19
3.1 Аналіз окремих текстів 19
3.2 Аналіз корпусу 25
Висновки 30
Список використаних джерел 31
А Програмний код preprocessor.py 33
Б Програмний код corpus_analyser.py 36
В Програмний код static_characteristics.py 40
1

ЗМІСТ
2
Г Програмний код heaps.py 43
Д Програмний код merger.py 45

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 було досліджено код Unity, як корпус текстів, його флуктуаційну поведінку та виконання закону Тейлора. Виявлено,

що основні лінгвістичні залежності, які виконуються для текстів, написаних на природних мова, виконуються також для комп'терного коду.
3

Abstract
4

Розділ 1
Теорія
1.1 Закони Ципфа
Закони Ципфа описують статистичні закономірності в розподілі величини частоти виникнення певних подій. Ці закони були відкриті в 20-х роках XX століття економістом Вільямом Стенлі Ципфом, 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 Ципфа, 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) =
A1
rα
, (1.1)
де α — степеневий коефіцієнт, зазвичай α ≈ 1.
Другий закон Ципфа 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) =
B
F β
, (1.2)
де β — степеневий коефіцієнт. Зазвичай β ≈ 2.
Закон Парето — інтегральна форма другого закону Ципфа, де P (F) — кумулятивна густина ймовірності (тобто ймовірність натрапити на слово з частотою меншою або рівною F :
P (f) =
C
F k
, (1.3)
5

РОЗДІЛ 1. ТЕОРІЯ 6
тут k — степеневий коефіцієнт.
Наведені степеневі коефіцієнти, як відомо з теорії, пов'язані такими залежностями: β = 1 + 1/α, а k = 1/α.
1.2 Поправка Мандельброта
Відомо, що зазвичай закон Ципфа 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 Ципфа-Мандельброта має вигляд:
f(r) =
A2
(r0 + r)α
, (1.4)
де A2, r0 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) = **DLγ** (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.

РОЗДІЛ 1. ТЕОРІЯ 7
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 звертатиметься саме до визначення цих законів через словник, оскільки таким чином ми їх використовуємо в роботі.
Так залежність середньоквадратичного відхилення розмірів словника dV відрозмірів тексту L має такий вигляд:
dV ∝ Lα1, (1.6)
де α1 — степеневий коефіцієнт. Зазвичай α1 лежить між значенням 0.5 і 1.0, коли α1 = 0.5, то це означає, що в тексті нема далекосяжних кореляцій, якщо ж α1 > 0.5—це вказує на наявність цих далекосяжних кореляцій. Рівняння 1.6 ще називають флуктуаційним скейлінгом.
Залежність середньоквадратичного відхилення розмірів словника dV відрозмірів словника V описується через закон Тейлора:
dV ∝ V α2, (1.7)
де α2 —степеневий коефіцієнт. Подібно до флуктуаційного скейлінгу 1.6, якщо α2 лежить між значенням 0.5 і 1.0, коли α2 = 0.5, то це означає, що в тексті нема далекосяжних кореляцій, якщо ж α2 > 0.5 — це вказує на наявність цих далекосяжних кореляцій.

РОЗДІЛ 1. ТЕОРІЯ 8
1.5 Лінгвістичні дослідженнятекстівкомп'ютерних програм
На сьогоднііснує 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 підтримуєзноманітні платформи, такі як ПК, консолі, мобільні пристрої і віртуальна реальність (VR), що робить його популярним в галузіігрової 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):Компонентипредставляютьфункціональні одиниці, які можуть бути додані до ігрових об’єктів. Вони визначають поведінку, властивості та взаємодіюоб’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 розробціігор 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 іншіінтерактивні 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 використанішні 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.

РОЗДІЛ 2. МЕТОДИКА 12
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 використанішні 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 тип ізацію, лексичну область видимості, імператив, деклара-**
тивний, функціональний, універсальний об'єкт- орієнтован (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 2019версії 16.3.Під час розробки .NET Framework бібліотекикласів 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 «С-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

РОЗДІЛ 2. МЕТОДИКА 13
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**TurboPascal**, EmbarcaderoDelphi (**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 **1994році, 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 Анджелика 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** створювати, що **Java або C# є революційними мовами програмування,** які змінили **спосіб написання програм »**, а« C# **багато запозичив у** Java - і навпакиТепер, коли **C# п** ідтримує упаковки і розпаковування, у

нас буде дуже схожа функц я в Java.
"У липн 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 листопад 2005 року мови C# і Java розвивалися за все б льш і б льш розб жним тра єкторіях, ставши двома зовс м р зними мовами. Одним з перших основних в дхилень стало додавання узагальнень до обох мов з абсолютно р зними реал зац ями. C# використовує reification для надання «першокласних» ун версальних об'єкт в, як можна використовувати як будь-який нший клас, з генерац єю коду, що виконується п д час завантаження класу. Кр м того, в C#, додано к лька основних функц й для програмування функц онального стилю, кульм нац єю яких є розширення LINQ, випущен в C# 3.0, і п дтримуюча його структура лямбда-вираз в, метод в розширення і анон мних тип в. Ц функц дозволяють програм стам C# використовувати методи функц онального програмування, так як замикання, коли це виг дно для їх застосування. Розширення LINQ і функц ональний імпорт допомагають розробникам скоротити обсяг стандартного коду, який включається в загальн завдання, так як запити до бази даних, анал з файлу XML або пошук в структур даних, зм щуючи акцент на реальну лог ку програми, щоб пол пшити читан сть і ремонтопридатн сть. По своєму дизайну C# є мовою програмування, який безпосередньо в дображає основну нфраструктуру сп льно мови (CLI). Б льш сть його внутр -

РОЗД Л 2. МЕТОДИКА 14
шн х типів в дпов дають типам значень, реал зованих середовищем CLI. Однак специф кац я мови не м стить вимог до генерац ї коду комп лятора, тобто не вказує, що комп лятор C# має ор єнтуватися на середовище виконання сп льно мови, або генерувати загальний пром жний мова (CIL), або генерувати будь- який нший конкретний формат. Теоретично, комп лятор C# може генерувати машинний код, як традиц йн комп лятори C++ та Fortran.
C# п дтримує строго типізован оголошення неявних зм нних з ключовим словом var, а також неявно типізован масиви з ключовим словом new [], за яким сл д н ц ал затор колекц ї.
C# п дтримує суворий лог чний тип даних, bool. Оператори, як приймають умови, так як while і if, вимагають вирази типу, що реал зує оператор true, такого як лог чний тип. Хоча C ++ також має лог чний тип, в н може в льно перетворюватися в ц л і з ц лих чисел, а вирази, наприклад, якщо (а) вимагають т льки, щоб а був перетворений в bool, дозволяючи а бути int або покажчиком. C# забороняє цей п дх д «ц лочисельне значення стина або брехня» на т й п дстав, що примус програм ст в використовувати вирази, як повертають саме bool, може запоб гти певн типи помилок програмування, таких як if (a = b) (використання присвоювання = зам сть р вност ==). C# б льш безпечний для тип в, н ж C++. Єдиними неявними перетворення-
ми за замовчуванням є т , як вважаються безпечними, наприклад, розширення ц лих чисел. Це застосовується п д час компл ц , п д час JIT і, в деяких випадках, п д час виконання. Не в дбуваються неявного перетворення м ж логічними значеннями і ц лими числами, а також м ж членами перерахування і ц лими числами (за винятком л терала 0, який може бути неявно перетворений в будь- який перераховується тип). Будь кероване перетворення повинне бути ч тко позначено як явне або неявне, на в дм ну в д конструктор в коп ювання C ++ та операторів перетворення, як за замовчуванням неявн .
C# має явну п дтримку ковар ац і контравар антності в універсальних типах, на в дм ну в д C++, яким має деяку ступ нь п дтримки контравар антності просто завдяки семантиц повертаються тип в в ртуальних методи.
Учасники перерахування розм щуються у своїй област видимості.
Мова C# не допускає глобальних зм нних або функц й. Вс методи і чл ни повинн бути оголошен всередин клас в. Статичн члени в дкритих клас в можуть зам нювати глобальн зм нні і функц ї.
Локальн зм нні не можуть приховувати зм нні вм щує блоку, на в дм ну в д C та C++.

РОЗД Л 2. МЕТОДИКА 15
C# має п дтримку строго типізованих покажчик в на функц ї через ключове слово делегат. Як і псевдо-C ++ фреймворк Qt, в C# є семантика, зокрема навколишнє под ї стилю публ кац й-п дписки, хоча C# використовує для цього делегати.
Керована пам'ять не може бути зв льнена; зам сть цього в н автоматично збирається. Зб рка см тт я вир шує проблему витоків пам'яті, позбавляючи програм ста в д в дпов дальності за зв льнення пам'яті, яка б льше не потр бна.
На в дм ну в д C ++, C# не п дтримує множинне успадкування, хоча клас може реал зовувати будь-яку к льк сть

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 частков класи, анонімні методи,

нульові типи і універсальні шаблони (щось схоже на шаблони C ++). У липні 2005 року ECMA представила в ICO / MEK JTC 1 за допомогою прискореного процесу останнього стандарти та відповідні ТЗ. Цей процес зазвичай займає 6-9 м сяців. Визначення мови C# і CLIстандартизовані 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 в дкритим 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 повн стю написаний на керованому коді (C#), був відкритий, а функціональність відображена як API. Це дозволяє розробникам створювати інструменти рефакторингу та діагностики.

Інш компілятори C# (деякі з яких включають реал зацію 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 необх дні 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, план в по впровадженню WPF не існує; WF планується до більш пізнього випуску; і є лише частков 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 необх дні 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 Microsoftпід 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 Ципфа, Парето та Гіпса, звісно, до кінця автоматизувати цю перевірку майже неможливо (адже ширина ципфового 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 лишефлуктуації 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.csv, частини програмного пакету Unity Reference Code. Червоною пунктирною лінією зображена апроксимація залежності законом Ципфа 1.1, синьою—законом Ципфа–Мандельброта 1.4. Для закону Ципфа отримано α = 1.32 і A1 = 0.524 з коефіцієнтом кореляції Пірсона r = 0.992, для закону Ципфа–Мандельброта — α = 1.39, A2 = 0.634 і r0 = 2.9. Як і очікувалося, закон Ципфа–Мандельброта доволі непогано описує залежність частоти від рангу для слів низьких рангів (високої частоти), проте недооцінює частоти для середніх та високих рангів. Натомість закон Ципфа добре описує якраз середні ранги, низькі— дещо гірше, та все ж краще, ніж закон Ципфа–Мандельброта. Співвідношення N2/N1 для EditorGUI.csv складає 63 %. Частка hapax legomena в усьому тексті становить ~ 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 бінновано за допомогою естиматора 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 Ципфа (другий закон Ципфа) 1.2:β = 1.644,B = 7.691×10−9, 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 × 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 γ ∼ 0.8, а синім — режим сповільненого росту з γ ∼ 0.57.
На рис. 3.4 зображений ріст словника відносно росту довжини тексту для EditorGUI.csv. Ранні 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, γ ∼ 0.8, пізніше ріст доволі різко сповільнюється до того, що γ ∼ 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 α = 1.09, A1 = 0.1183 і коефіцієнт кореляції Пірсона r = 0.987. Апроксимація законом Ціпфа–Мандельброта 1.4 повертала не дуже реалістичні значення (наприклад, α ∼ 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, томуформула 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 найбільшоючастотоюоб’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 більнуваннявідбувалося за допомогою естиматора Фрідмана–Діаконіса [6], після чого біни з нульовою кількістю частот відкидалися. У результаті отримано 570 точок. Апроксимація формулою розподілу Ціпфа 1.2 дала β = 1.99, B = 5.1083× 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× 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 γ ∼ 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 достатньо великий текст, проте, мабуть, недостатньо, щоб повертати статистично надійні дані.
3.2 Аналіз корпусу
Уцій частині роботими 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 бони 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 тією ж формулою 1.5. Цікаво, що отримали результат, відмінний від того,що спостерігали на прикладі об'єднаного тексту Unity, якщо там був γ ∼ 0.8, то тут γ = 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 дрозмірів тексту L. Ми бачимо, що α1 = 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 дрозмірів середнього словника $\bar{V}$. Як відомо, ця залежність описується законом Тейлора 1.7. Також показник α2 пов'язує між собою показники γ та α1 таким співвідношенням: α2 = γ/α1. Як бачимо, α2 = 1.18 отримане як наслідок процедури апроксимації та таке «теоретично» оцінене α2 = 1.16 лежать доволі близько. Таке значення α2 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
написаногомовоюпрограмування. Зазвичай у цихмов 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 V

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

степінь значення коефіцієнт кореляції
γ 0.627 0.999
α1 0.739 0.983
α2 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. ZhangH.Discovering power laws in computer programs // InformationProcessing and Management. — 2009. — Т. 45. — С. 477—483. — DOI: 10.1016/j.ipm.2009.02.001.

3. Pierret D.,D.P.Anempirical exploration of regularities in open-source software lexicons // 2009 IEEE 17th International Conference on ProgramComprehension. — 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. FreedmanD.,Diaconis P.On the Histogram as a Density Estimator:L2 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.

Додаток А
Програмний код
preprocessor.py

```
1 import pandas as pd
3 import re
5  import matplotlib.pyplot as plt
7  import  os
9 def dotsub(lines):
lines = re.sub('\.\.', '␣..␣', lines)
11 lines = re.sub('(?&lt;=[a-zA-Z_\?]) \.(?=[a-zA -Z_\.])', '␣.␣',
lines)
return lines
13
def bracketsub(lines):
15 lines = re.sub('{', '{␣', lines)
lines = re.sub('\(', '␣(␣', lines)
17 lines = re.sub('\[', '␣[␣', lines)
return lines
19
def ltsub(lines):
21 return re.sub(' &lt;[^=-]', '␣&lt;␣', lines)
23 def gtsub(lines):
return lines = re.sub('(? &lt;=[^=])&gt;', '␣&gt;␣', lines)
25
def equalsub(lines):
33
```



```
27 return re.sub('=[^=&gt;]', '=␣', lines)
29 def pointersub(lines):
return re.sub(' -[^&gt;]', '-␣', lines)
31
def plussub(lines):
33 lines = re.sub('\+\+', '␣++␣', lines)
lines = re.sub('\+[^\+] ', '␣+␣', lines)
35 return lines
37 def minussub(lines);
lines = re.sub('--', '␣--␣', lines)
39 lines = re.sub(' -[^-]', '␣-␣', lines)
return lines
41
def slashsub(lines):
43 lines = re.sub('\*', '␣*␣', lines)
lines = re.sub(r'\\', '␣\␣', lines)
45 return lines
47 def questsub(lines):
return re.sub('\?[^\.] ', '␣?␣', lines)
49
def spacesub(lines):
51 return re.sub('\s+', '␣', lines)
53 def punctsub(lines):
lines = re.sub(':', '␣:␣', lines)
55 lines = re.sub(';', '␣;␣', lines)
lines = re.sub('"', '␣"␣', lines)
57 return lines
59 def main():
root_dir = os.getcwd ()
61
directories = ['profilereader -main', 'unitychanspringbone ', '
unityrenderstreaming ', 'anotherthread -game', 'fpssample -
```

game', 'unitycsreference ', 'waveshooter -demo']
63
#directory_path=directory[7]
65
directory_path = 'unitycsreference '
67 for file in os.listdir(f'{directory }/ source /{ directory_path}'

):
filename , _ = os.path.splitext(file)
69
with open(f'{directory }/ source /{ directory_path }/{ file}')
as f:
71 lines = []
for line in f.readlines ():
73 if not line.strip ().startswith('/'):
line = re.sub(r'.*?\/(.*)$', '', line)
75 lines.append(line)
77 lines = '␣'.join(lines).replace('\n', '␣\n␣').replace(','
, '␣').replace('@', '').replace(')', '').replace('}',
'').replace(']', '')
79 lines = dotsub(lines)
lines = bracketsub(lines)
81
lines = ltsub(lines)
83
lines = gtsub(lines)
85
lines = equalsub(lines)
87
lines = pointersub(lines)
89
lines = plussub(lines)
91
lines = minussub(lines)
93
lines = slashsub(lines)
95
lines = questsub(lines)
97
lines = spacesub(lines)
99
with open(f'{directory }/ corpus/unity_code /{ filename }.txt'
, 'w') as f:
101 f.write(lines)
103 if __name__ =='__main__ ':
main()

Додаток Б
Програмний код
corpus_analyser.py
 import pandas as _pd_
2
from scipy_ import stats
4 import numpy as np
6 import os
8 import matplotlib.pyplot as plt
10
directory = os.getcwd ()
12 folder = 'processed '
14 data_inside = {}
16 for root , dirs , files in os.walk(f'{directory }/{ folder}'):
for file in files:
18 filename , extension = os.path.splitext(file)
if extension == '.txt':
20 print(file)
vocab_counter = 0

```
22 length_counter = 0
vocab = []
24 with open(f'{root }/{ file}', 'r') as f:
text = f.read().split('␣')
26 for word in text:
if not word in vocab:
36
```



```
28 vocab.append(word)
30
data_inside[file] = (len(text), len(vocab))
32
34 data = pd.DataFrame.from_dict(data_inside , orient='index ').
reset_index ()
data = data.rename(columns ={'index': 'name', 0: 'length ', 1: '
vocab_size '})
36
data = data.sort_values(by=['length '], ignore_index=True)
38
fig , ax = plt.subplots(dpi =300)
40
ax.set(xscale='log', yscale='log', xlabel=r'$L$', ylabel=r'$V$')
42
ax.scatter(data['length '], data['vocab_size '], edgecolor='black',
facecolor='white ')
44
plt.savefig(f'{directory }/ figures/unity -corpuse.png')
46
alpha_1 = []
48 alpha_2 = []
gamma = []
50
windows = [i for i in range(5, 300)]
52
for window in windows:
54 step = window
ls = data['length ']. rolling(window=window , step=step).mean()
56 vs = data['vocab_size ']. rolling(window=window , step=step).
mean()
dv = data['vocab_size ']. rolling(window=window , step=step).std
()
58
slope , intercept , r, p, se = stats.linregress(np.log(ls.
dropna ()), np.log(vs.dropna ()))
60
alpha_1.append(slope)
62
slope , intercept , r, p, se = stats.linregress(np.log(ls.
dropna ()), np.log(dv.dropna ()))
```



```
64
alpha_2.append(slope)
66
slope , intercept , r, p, se = stats.linregress(np.log(vs.
dropna ()), np.log(dv.dropna ()))
68
gamma.append(slope)
70
window = 50
72 step = window
74 ls = data['length ']. rolling(window=window , step=step).mean()
vs = data['vocab_size ']. rolling(window=window , step=step).mean()
76 dv = data['vocab_size ']. rolling(window=window , step=step).std()
78 r_symbol = r'$r␣=␣$'
alpha1_symbol = r'$\alpha_1␣=$'
```

```
80 alpha2_symbol = r'$\alpha_2␣=$'
gamma_symbol = r'$\gamma␣=␣$'
82
slope , intercept , r, p, se = stats.linregress(np.log(ls.dropna ())
, np.log(vs.dropna ()))
84
print(slope , intercept , r)
86
fig , ax = plt.subplots(dpi =300)
88
ax.set(xscale='log', yscale='log', xlabel=r'$L$', ylabel=r'$V$')
90
ax.scatter(ls, vs, color='black', label='data')
92 ax.plot(ls.dropna ().tolist (), np.e** intercept * ls.dropna ().
to_numpy ()**slope , '--', color='red', label='fit')
94 ax.text(x=70, y=120, s=f'{gamma_symbol }{slope :.3} ,\n␣{r_symbol }{r
:.3}')
96 ax.legend ()
98 plt.savefig(f'{directory }/ figures/songs_window50_step50_VL.png')
100 slope , intercept , r, p, se = stats.linregress(np.log(ls.dropna ())
, np.log(dv.dropna ()))
```



```
102 print(slope , intercept , r)
104 fig , ax = plt.subplots(dpi =300)
106 ax.set(xscale='log', yscale='log', xlabel=r'$L$', ylabel=r'$\
Delta␣V$')
108 ax.scatter(ls, dv, color='black ', label='data')
ax.plot(ls.dropna ().tolist (), np.e** intercept * ls.dropna ().
to_numpy ()**slope , '--', color='red', label='fit')
110
ax.text(x=100, y=105, s=f'{alpha1_symbol }{slope :.3},\n␣{r_symbol
}{r:.3}')
112
ax.legend ()
114
plt.savefig(f'{directory }/ figures/songs_window50_step50_dVL.png')
116
slope , intercept , r, p, se = stats.linregress(np.log(vs.dropna ())
, np.log(dv.dropna ()))
118
print(slope , intercept , r)
120
fig , ax = plt.subplots(dpi =300)
122
ax.set(xscale='log', yscale='log', xlabel=r'$V$', ylabel=r'$\
Delta␣V$')
124
ax.scatter(vs, dv, color='black ', label='data')
126 ax.plot(vs.dropna ().tolist (), np.e** intercept * vs.dropna ().
to_numpy ()**slope , '--', color='red', label='fit')
128 ax.text(x=100, y=105, s=f'{alpha2_symbol }{slope :.3},\n␣{r_symbol
}{r:.3}')
130 ax.legend ()
132 plt.savefig(f'{directory }/ figures/songs_window50_step50_dVV.png')
```

Додаток В

Програмний код

static_characteristics.py

```
1 import os
3  import pandas as pd
import numpy as np
5 from scipy  import  stats
7  import matplotlib.pyplot as plt
9 from collections import Counter
11 directory = os.getcwd ()
13 folder = 'processed '
```

```
filename = 'EditorGUI '
15
with open(f'{directory }/{ folder }/{ filename }.txt', 'r') as file:
17 lines = file.read()
19
text = lines.split('␣')
21
unique = Counter(text)
23
vocab = pd.DataFrame.from_dict(unique , orient='index ').
reset_index ()
25 vocab = vocab.rename(columns ={"index": "word", 0: "freq"})
vocab = vocab.sort_values(by='freq', ascending=False ,
40
```



```
ignore_index=True)
27 vocab.index = vocab.index + 1
vocab['norm_freq '] = vocab['freq'] / vocab['freq'].sum()
29
start = 50
31 end = 5000
33 slope , intercept , r, p, se = stats.linregress(x=np.log(vocab.
index[start:end]), y=np.log(vocab['freq'][start:end]))
35 print(slope , np.e**intercept , r)
37 fig , ax = plt.subplots(dpi =300)
39 ax.set(xscale='log', yscale='log', xlabel=r'$r$', ylabel=r'$f(r)$
')
41 ax.scatter(vocab.index , vocab['norm_freq '], edgecolor='black',
facecolor='white ', label='data')
ax.plot(vocab.index.to_numpy (), np.e** intercept * vocab.index.
to_numpy ()**slope , '--', color='red', label='Zipf')
43
ax.legend ()
45
plt.savefig(f'{directory }/ figures /{ filename}_zipfs -law.png')
47
probs , bins = np.histogram(vocab['norm_freq '], bins='fd')
49
pdf = pd.DataFrame ()
51 pdf['bins'] = bins [:-1]
pdf['probs'] = probs / len(vocab['norm_freq '])
53
pdf = pdf[pdf['probs'] != 0]. reset_index(drop=True)
55 pdf.index = pdf.index + 1
57 start_rank = 5
end_rank = 250
59
slope , intercept , r, p, se = stats.linregress(x=np.log(pdf['bins'
][ start_rank:end_rank ]), y=np.log(pdf['probs'][ start_rank:
end_rank ]))
61
print(slope , np.e**intercept , r)
```



```
63
fig , ax = plt.subplots(dpi =300)
65
ax.set(xscale='log', yscale='log', xlabel=r'$f$', ylabel=r'$p(f)$
')
67
ax.scatter(pdf['bins'], pdf['probs '], edgecolor='black',
facecolor='white ')
69 ax.plot(pdf['bins'][: end_rank ]. to_numpy (), np.e** intercept * pdf[
'bins'][: end_rank ]. to_numpy ()**slope , '--', color='red', label
='Zipf')
71 ax.legend ()
```

```
73 plt.savefig(f'{directory }/ figures /{ filename}_pdf.png')
75 pdf['cdf'] = np.cumsum(pdf['probs '])
pdf['ccdf'] = 1 - pdf['cdf']
77
slope , intercept , r, p, se = stats.linregress(x=np.log(pdf['bins'
][5: end_rank ]), y=np.log(pdf['ccdf'][5: end_rank ]))
79
print(slope , np.e**intercept , r)
81
fig , ax = plt.subplots(dpi =300)
83
ax.set(xscale='log', yscale='log', xlabel=r'$f$', ylabel=r'$P(f)$
', ylim ={1e-5, 2})
85
ax.scatter(pdf['bins'], pdf['ccdf'], edgecolor='black', facecolor
='white')
87 ax.plot(pdf['bins']. to_numpy (), np.e** intercept * pdf['bins'].
to_numpy ()**slope , '--', color='red', label='Zipf')
89 ax.legend ()
91 plt.savefig(f'{directory }/ figures /{ filename}_ccdf.png')
```

Додаток Г
Програмний код heaps.py

```
2 import os
4 import pandas as pd import numpy as np
6 import matplotlib.pyplot as plt
8 from scipy import stats
10 root = os.getcwd ()
corpus_path = 'corpus/mergers '
12 figure_path = 'figures '
textname = 'songs '
14
with open(f'{root }/{ corpus_path }/{ textname }.txt', 'r') as file:
16 lines = file.read()
18 text = lines.split('␣')
20 lengths = []
vocabs = []
22
step = 100
24
for i in range (100, len(text), step):
26 lengths.append(i)
vocabs.append(len(set(text[:i])))
28
30 print(f'Length␣of␣text␣in␣words␣is:␣{len(text)}')
43
```



```
print(f'Vocabulary␣size␣is:␣{len(set(text))}')
32
slope , intercept , r, p, se = stats.linregress(x=np.log(lengths[
start :]), y=np.log(vocabs[start :]))
34
print(f'slope␣is␣{slope :.4}')
36 print(f"Pearson 's␣correlation␣is␣{r:.4}")
38 fig , ax = plt.subplots(dpi =300)
40 ax.set(xscale='log', yscale='log', xlabel=r'$L$', ylabel=r'$V$')
42 ax.scatter(lengths [::], vocabs [::], facecolor='white', edgecolor=
'black', label='data')
ax.plot(lengths , np.e** intercept * lengths **slope , '--', color='
red', label=f'Heaps')
44
ax.legend ()
46
plt.show()
48
plt.savefig(f'{root }/{ figure_path }/{ textname}-heaps.png')
```

Додаток Д

Програмний код merger.py

```
import os
2
import shutil
4
home = os.getcwd ()
6 corpus_folder = 'corpus '
texts_folder = 'unity_code '
8
with open(f'{home }/{ corpus_folder }/ mergers /{ texts_folder }.txt','
wb') as destination_file:
10 for filename in os.listdir(f'{home }/{ corpus_folder }/{
texts_folder}'):
with open(f'{home }/{ corpus_folder }/{ texts_folder }/{
filename}','rb') as current_file:
12 shutil.copyfileobj(current_file , destination_file)
45
```