МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ОРЛОВ СЕРГІЙ СЕРГІЙОВИЧ

Допускається до захисту:
в. о. завідувача кафедри
Прикладної математики,
_____Трофименко О. Д.
«____» _____ 20__ р.

СЕРВІС ПІДТРИМКИ ЛІНГВІСТИЧНОГО ДОСЛІДЖЕННЯ

Спеціальність 113 Прикладна математика

Кваліфікаційна (бакалаврська) робота

Керівник:
Вєтров О. С., старший викладач
кафедри Прикладної математики

Оцінка: _____/_____/_____
(бали за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2021

АННОТАЦІЯ

Орлов С. С. Сервіс підтримки лінгвістичного дослідження. Спеціальність 113 «Прикладна математика», спеціалізація «». Донецький національний університет імені Василя Стуса, Вінниця, 2021.

У кваліфікаційній (бакалаврській) роботі досліджена можливість створення власного програмного засобу для розпізнавання мовлення. Показано процес створення експертної системи. З'ясовано проблематику побудови систем розпізнавання мовлення.

Ключові слова: лінгвістика, аналіз, сервіс, експертна система

36 с., 3 рис., 24 посилання.

ABSTRACT

Orlov S. S. Linguistic research support service. Specialty 113 «Applied mathematics», Vasyl' Stus Donetsk National University, Vinnitsia, 2021

In the qualification (bachelor's) work investigated the possibility of creating your own software for speech recognition. Shown the process of creating an expert system. Clarified the problems of building speech recognition systems.

Keywords: linguistics, analysis, service, expert system

ЗМІСТ

ВСТУП

Актуальність теми. Проектування експертних систем – це один з напрямків машинного навчання (засобів штучного інтелекту), що є дуже потужним рішенням для вирішення низки практичних задач. Лінгвістика, як і будь-яка інша наукова галузь, також має різні практичні задачі, для вирішення яких було б зручно застосувати експертну систему. Зокрема з цих задач можна виділити задачі на аналіз діалектного мовлення, а саме у цій роботі мовлення мешканців міста Гнівань. Саме тому створення експертної системи аналізу діалектного мовлення – важливий крок для вирішення лінгвістичних питань. Також система може бути легко масштабована для аналізу будь-яких діалектів або, наприклад, сленгу.

Мета роботи. Дослідження різних готових систем розпізнавання мовлення. Аналіз можливості створення власної системи. Створення експертної системи для аналізу мовлення мешканців міста Гнівань.

Завдання дослідження. Дослідити процес створення систем розпізнавання мовлення. Виявити можливі проблеми, що заважають власноручній реалізації. Створити експертну систему для аналізу гніваньської говірки.

Об’єкт дослідження. Програма (експертна система) для аналізу діалекту мешканців міста Гнівань.

Предмет дослідження. Програмні засоби, за допомогою яких буде створена експертна система.

Теоретичне та/або практичне значення одержаних результатів. Отримана в результаті роботи експертна система може бути успішно застосована для вирішення задач прикладної лінгвістики. Також при незначних модифікаціях може отримати широкий спектр практичних застосувань.

Структура кваліфікаційної (бакалаврської) роботи. Основна частина роботи складається з трьох частин. В першій описується теоретична сторона експертної системи, та досліджуються системи розпізнавання мовлення. Друга частина описує практичну частину розробки експертної системи говірки міста

Гнівань. Третя частина присвячена можливостям для подальшого розвитку та масштабування експертної системи. В роботі присутні 3 рисунки (ще 5 у додатку А): 24 посилань; 2 додатки: перший містить скріншоти розробленої експертної системи, другий лістинги коду.

РОЗДІЛ 1

ЕКСПЕРТНІ СИСТЕМИ ЛІНГВІСТИЧНОГО НАПРАВЛЕННЯ

1.1. Поняття експертної системи

Експертною системою називають комп’ютерну програму, побудовану на основі певної бази знань, з метою вирішення певних практичних питань, притаманних конкретній галузі, або, наприклад, для надання рекомендацій по цим питанням. Ці системи створюються для заміщення людей-експертів, що дасть змогу вирішувати нові питання та задачі, а не витрачати час на вже

розв’язані. До того ж комп’ютер не помиляється та не спить. Проектування експертних систем – це одна з галузей методів застосування штучного інтелекту або машинного навчання. Ці технології спрямованні на заміщення людей в тих предметних галузях, що потребують мислення. Як парові машини раніше замістили звичайних робітників, експертні системи через деякий час почнуть заміщувати вчених в деяких галузях [1].

В основі будь-якої експертної системи має лежати база знань, котру формують фахівці та експерти в даній конкретній галузі на основі своїх знань та досвіду. Через це можна легко та швидко отримувати відповіді на питання з цієї галузі, не витрачаючи багато часу на пошук експертів, обговорення питання з ними, аналіз їхніх відповідей.

На сьогоднішній день створено вже багато експертних систем. Ці системи вирішують широкий спектр питань з багатьох різних предметних галузей. Предметні галузі, у яких створені ці експертні системи, як правило, достатньо добре вивчені та систематизовані, що дає змогу з великою точністю вирішувати будь-які питання, що відносяться до цих галузей знань.

Перші експертні системи почали створювати у 70-х роках минулого століття та активно розвиваються у наші дні

Ідея експертних систем була винайдена у Стенфордському університеті в рамках проекту по дослідженню методів евристичного програмування, котрий очолював Едвард Фейгенбаум. Через це Фейгенбаума деякі дослідники вважають «засновником експертних систем». Це також зазначено на обладинці його книги про ці дослідження, яка називається «Становлення експертної кампанії» [2]. Мета дослідження науковців зі Стенфордського університету – встановити, у яких сферах науки знання та практичний досвід людей міг би бути класифікованим та структурованим, так, щоб його можна було б використовувати для проектування експертних систем, за допомогою можна буде полегшити рішення практичних питань. Зокрема можна виділити такі галузі як діагностика різноманітних захворювань, або ідентифікація невідомих

органічних поєднань, в цих сферах існує вірогідність помилки, через так званий «людський фактор», тому можливість довірити ці питання машинам значно полегшить життя людству. Фейгенбаум сформулював визначення операційної системи наступним чином – це інтелектуальна програма, що заснована на знаннях експертів, які використовуються для вирішення питань, котрі занадто трудозатратні, для людей.

Експертні системи – це найперше вдале варіація програмного засобу, в основі якого закладені методи штучного інтелекту.

Також під експертною системою можна вважати програмний засіб, за допомогою якого знання та досвід, отримані багатьма експертами в певній сфері, можуть бути отримані в будь-який час будь-ким, без необхідності шукати фахівця та відволікати його від наукової роботи [3].

Експертні системи достатньо широко використовуються для вирішення низки питань у галузі прикладної лінгвістики. Найвідоміший приклад – це програми-перекладачі, також можна відмітити голосових помічників, програми для вивчення мови, тощо.

1.2. Особливості мовлення жителів міста Гнівань

Діалект міста Гнівань належить до групи північно-подільських говірок, подільського говору, волинсько-подільської групи, південно-західного наріччя. Вона межує по класифікації з говірками волинської групи, південно-західного наріччя та говірками середньонаддніпрянської групи, південно-східного наріччя.

Ця говірка має певні спільні морфологічні особливості з волинськими та середньонаддніпрянськими говірками, південно-західного та південно-східного наріччя відповідно. Також є деякі спільні риси з говірками північного наріччя [4].

Для Гніваньської говірки характерне активне використання іменників, дієслів та займенників. Рідше носії використовують прикметники та

числівники. До пасивної лексики також можна віднести дієприкметники. Використання дієприслівників мінімальне [5].

1.3. Використання машинного навчання для аналізу мовлення

Розпізнавання мовлення – це процес трансформації звукових сигналів у вигляді послідовності звукових хвиль у цифрову інформацію, найчастіше – набір текстових даних.

Перший крок у вирішенні задачі розпізнавання мовлення був зроблений у 1952 році, коли була розроблена перша машина. Вона могла розпізнавати сказані людиною цифри англійською мовою.

Перші комерційні рішення для розпізнавання мовлення стали достпуними широкій публіці на початку 90-х років минулого століття.

На сьогоднішній день ми вже маємо набагато розвиненіші системи, що здатні «на льоту» транскрибувати майже будь-яку мову Світу у звичайний текст.

Наступним кроком у розвитку рішень розпізнавання мовлення можна вважати впровадження інтерфейсів безмовного доступу. Системи такого рівня базуються на отриманні та обробці даних на ранній стадії артикуляції. Необхідність цих систем зумовлена двома головними недоліками існуючих рішень розпізнавання мовлення, а саме: висока чутливість до зайвих шумів та необхідність чіткого та зрозумілого вимову. Це створює перешкоди для використання програм поточного рівня у шумних місцях, або для людей з розладами мовлення. Інтерфейси безмовного доступу передбачають впровадження нових сенсорів, які не будуть схильні до впливу небажаних шумів.

Ця технологія використовується у численних сферах життя, мабуть найпростішим прикладом буде набір пошукового запиту голосом, крім того можна наговорювати в мікрофон повідомлення у багатьох месседжерах, деякі

текстові редактори пропонують навіть опції набору цілих документів лише голосом.

Розпізнавання окремих слів з постійного потоку звуків через мікрофон використовується у реалізації багатьох систем «розумних домів». Таким чином можна прив’язати до конкретних слів або словосполучень конкретні дії, наприклад ввімкнути або вимкнути світло, запустити робот-пилосос або пральну машину.

Але використання засобів розпізнавання мовлення не обмежується лише побутовими потребами, наприклад лікар може не записувати довгий діагноз або рецепт не завжди зрозумілим почерком, а просто надиктувати його у мікрофон, а спеціальна програма сама заповнить необхідний бланк, а потім ще й роздрукує його. Також на якомусь виробництві можна підвищити швидкість роботи, прив’язавши певні процеси до спеціальних голосових команд. А можна і підвищити безпеку, якщо для запинки виробничої лінії можна буде лише вимовити команду, а не бігти до рубильника через увесь цех.

1.4. Класифікація систем розпізнавання мовлення

Існуючі **системи розпізнавання мовлення** можна **класифікувати** за наступними критеріями:

- **За розміром словника** (невелика вибірка конкретних команд чи цілий словник для систем загального значення);
- **За залежністю від диктора** (дикторо- залежні або незалежні);
- **За типом мовлення** (роздільне або суцільне мовлення);
- **За призначенням** (командні системи, або системи потокової диктовки);
- **За використовуваним алгоритмом** (скриті Марковські моделі, динамічне програмування, нейронні мережі тощо);
- **За типом структурної одиниці** (фонема, слово, сполучення слів, тощо);
- **За принципом виокремлення структурних одиниць** (виділення окремих лексичних одиниць, розпізнавання по шаблону);

[6]

Серед методів розпізнавання мовлення можна виокремити метод динамічного програмування (Dynamic Type Warping), що передбачає порівняння зразків з еталоном, та низку методів на основі контекстно-залежної класифікації, яка передбачає, при перетворенні мовлення до тексту, виокремлення окремих елементів – фонем та алофонів, які потім групуються у склади та морфеми. До таких систем належать, наприклад: нейронні мережі, скриті Марківські моделі (одна з найбільш активно використовуваних систем для рішення питань розпізнавання мовлення) та методи дискримінантного аналізу, що засновані на Байесовській дискримінації.

1.5. Архітектура систем розпізнавання мовлення

Архітектура більшості використовуваних зараз програмних засобів для розпізнавання мовлення складається з деяких шарів, які спочатку

перетворюють аудіозапис, прибираючи з нього зайві перешкоди, потім розбивають на найменші складові, які далі, за допомогою певного алгоритму розпізнавання мовлення, будуть складені у слова, а потім у речення.

Деякі системи також здатні додавати найпростіші знаки пунктуації, такі як крапка або кома, але на даний момент часу не існує достатньо точного способу розпізнати з аудіозапису знаки пунктуації. Це можна зробити потім з готовим набором текстових даних, за допомогою, наприклад нейронних мереж, навчених розставляти знаки пунктуації у тексті, але це може спотворити початкову думку тексту і взагалі виходить за рамки даного дослідження [7].

Отже архітектура сучасної програми розпізнавання мовлення складається з наступних шарів:

1) **Модуль шумозаглушення** видаляє з початкової звукової доріжки зайві шуми, такі як дзвін посуду, гавкіт собаки, звуки працюючого автомобільного мотору, тощо. Також можуть видалятись звуки кашлю або чихання, або їх також можна розпізнавати і вставляти в текст спеціальними позначеннями.
2) **Акустичний модуль** дозволяє оцінити розпізнавання сегменту мовлення з точки зору схожості на рівні звукових хвиль. Спочатку для кожного звука будується складна статистична модель, що описує вимову цього звуку при мовленні
3) **Модель мови** визначає найбільш вірогідні послідовності морфем та слів. Складність побудови моделі залежить від конкретної мови, для якої ця модель будується. Для більшості мов, як-от наприклад англійська мова, достатньо побудови звичайних N-грам (статистичні моделі). Але для більш точного розпізнавання високофлективних мов, до яких, зокрема, відноситься й українська мова, звичайних статистичних моделей буде вже недостатньо – потрібно дуже багато даних, щоб точно визначити статистичний зв’язок між словами. Для таких мов використовують гібридні моделі, що вміщують в собі також

і правила мови, дані про форму слова та частину мови для побудови найбільш точних статистичних моделей.

4) **Декодер** сполучає дані, що були отримані в ході аналізу звуковою доріжки на акустичні моделі та моделі мови. Після об'єднання цих даних декодер визначає найбільш вірогідну послідовність слів, яка і буде кінцевим результатом розпізнавання мовлення для повного аудіозапису.

1.6. Перешкоди для реалізації систем розпізнавання мовлення

На відміну від графічної інформації (рисунки або відеозаписи) **мовлення** поділено на країни та народи. Тобто якщо фотографії тварин зроблені в Австралії та Україні, за умови, що фотографується один й той самий вид тварини, в схожих умовах, наприклад на фоні стіні – різниця між ними, з точки зору визначення виду тварини, буде відсутня. Натомість якщо ми будемо описувати якусь тварину, певного виду, одними й тими ж самими прикметниками, таким чином що суть опису буде цілком ідентична, то аудіозапис буде відрізнятись абсолютно. Через це систему розпізнавання мовлення потрібно будувати для кожної мови окремо, аналізуючи правила мови, особливості вимови, а також інші фактори.

Більше того, кожна мова має в собі різні діалекти, притаманні для певних місцевостей. Через це система, навчена розпізнавати певну мову, але з домішками якогось з її діалектів, може взагалі не правильно розпізнавати інший діалект.

Окрім різниці у вимові між різними діалектами, різниця може існувати на рівні звичайних людей. Деякі люди говорять швидше, деякі повільніше, хтось «ковтає» закінчення слова.

Також існують і різні розлади мовлення. Найбільш поширені з них ротацизм або шепелявість.

Зазвичай люди без зайвих зусиль розуміють слова та речення, промовлені на іншому діалекті (хоча деякі мешканці однієї і тієї ж країни, що живуть у

віддалених один від одного регіонах можуть не одразу зрозуміти один одного), з незвичною швидкістю або чіткістю вимови, з помилками у вимові окремих звуків, але для програмного засобу ці зміни можуть повністю порушити моделі, на яких він навчений.

Уникнути цих проблем взагалі навряд чи можливо, принаймні на сучасному рівні розвитку систем розпізнання мовлення. Але можна істотно покращити рівень якості розпізнавання, що визначається за наступною формулою:

$$WRR = 100 - (\frac{S+D+I}{T} * 100) \quad (1.1)$$

де: WRR (Word Recognition Rate) – відсоток правильно розпізнаних слів,
S (Swap) – кількість замінених слів,
D (Delete) – кількість видалених слів,
I (Insert) – кількість вставлених слів,
T (Total) – загальна кількість слів.
Також існує такий варіант:

$$WER = \frac{S + D + I}{T} * 100 \quad (1.2)$$

де: WER (Word error rate) – відсоток неправильно розпізнаних слів. Значення WER теоретично може перевищувати 100%, але на практиці це можливо лише у разі істотних помилок при проектуванні.

Іншою важливою формулою для аналізу якості розпізнавання мовлення є наступна формула:

$$RTF = \frac{T1}{T2} \quad (1.3)$$

де: RTF (Real-time factor, speed factor) – коефіцієнт швидкості, що показує здатність системи розпізнавати мовлення «на льоту», у реальному часі. Якщо

цей коефіцієнт менше одиниці, то програма спроможна аналізувати мовлення у реальному часі.

T1 (Processing Time) – час, необхідний для аналізу певної звукової доріжки,

T2 (Signal Time) – загальна тривалість аналізованої звукової доріжки.

Для того, щоб отримати високий показник WRR, та, відповідно, низький WER, потрібна велика кількість годин записів певною мовою, з високою диверсифікацією дикторів, які будуть вимовляти ті ж самі слова, але різним чином, притаманним саме для себе. Таким чином можна отримати широку вибірку даних, яка дозволить створити дикторонезалежну систему розпізнавання мовлення, таку, що зможе видавати досить високий показник правильно розпізнаних слів [8].

Проте не можна просто записати тисячі годин аудіозаписів, та віддати їх системі для навчання. По-перше це займе дуже багато часу при навчанні, не враховуючи навіть час, витрачений на запис та транскрибування записів. По-друге існує неілюзорна можливість «перенавчання», коли статистичні моделі будуть неймовірно складними, що призведе до стрімкого збільшення коефіцієнту швидкості, а отже унеможливить розпізнавання мовлення у реальному часі [9].

РОЗДІЛ 2

СТВОРЕННЯ ЕКСПЕРТНОЇ СИСТЕМИ ДЛЯ ГНІВАНЬСЬКОЇ ГОВІРКИ

2.1. Вибір мови та фреймворку для побудови експертної системи.

На сьогоднішній день, коли кажуть «штучний інтелект» або «машинне навчання», в контексті **мови програмування**, як правило, мають на увазі мову Python3 [10]. Ця мова програмування дійсно має низку переваг над іншими, зокрема вона дуже проста та зручна для написання програм. Саме тому програму на мові Python3 написати та налагодити можна набагато швидше, ніж програму з аналогічних функціоналом на якій-небуть іншій мові програмування.

Також, слід зазначити, що Python3 – це одна з мов, які належать до групи «інтерпретованих», це означає, що початковий код (source code) не компілюється в виконуваний файл (executable file), а у реальному часі, за допомогою спеціальної програми-інтерпретатора, рядок за рядком виконує всі програмні інструкції з вихідного текстового файлу [11]. Це дає змогу написати вихідний код лише один раз, а потім запускати його на будь-якій операційній системі та на будь-якому пристрої, потрібна лише наявність цієї самої програми-інтерпретатора.

Але за ці, безперечно, зручні функції доводиться розплачуватись продуктивністю написаної програми, або іншими словами часом виконання. Через те, що Python3 мова дуже високого рівня, програми, написані цією мовою програмування, легкі у читанні. Це відбувається через те, що чим вище рівень мови програмування – тим ближче вона до «натурального мовлення», тобто звичайних мов різних народів світу. Але на поточному рівні розвитку комп’ютерної техніки дуже важко аналізувати натуральні мови – одні й ті ж слова можуть означати різні речі, знаки пунктуації, поставлені не в тому місці, можуть змінити суть речення на діаметрально протилежну, як у всім відомому прикладі «стратити не можна помилувати». До того ж існують такі речі як

гумор та сарказм – ці області мовлення не дуже вивчені навіть людьми, а про те, щоб ці речі міг аналізувати комп'ютер мова не йде взагалі.

На зворотній стороні медалі мови низького рівня, наприклад мова assembly, або ж мова Ассемблеру (інколи люди називають її просто «ассемблер», проте це не вірно, тому що Ассемблер – це програма інтерпитатор, що виконує послідовність команд, написаних мовою Ассемблера [12]). На відміну від мов програмування високого рівня, ця мова не має ніяких зручних для розуміння програмістом надбудов, у цій мові не існує, в звичному для нас вигляді, навіть змінних та функцій, що й казати про абстрактне програмування та концепції ООП. Але такий код, без жодних зайвих речей, легше за все аналізувати та виконувати комп'ютеру.

Звісно жоден адекватний програміст в наш час не буде писати повноцінний програмний засіб на мові Ассемблера, через неймовірно великий об'єм часу, потрібний для написання цієї програми, це ще не кажучи про пошук та виправлення помилок в такому коді, або про подальшу модифікацію та масштабування. Але й жертвувати швидкодією програми теж не варто, якщо задача потребує великої кількості обчислень, або дуже частого оновлення стану.

Отже потрібно знайти «золоту середину» - мову, котра дасть достатньо високий рівень швидкодії, та разом з тим не буде потребувати від програміста дуже важких зусиль для написання та відладки програми.

Більшість програмістів вважає цією самої «золотою серединою» мову програмування C++, яку у 1983 створив данець Б'ярн Страуструп [13] (дан. Bjarne Stroustrup). Мова C++ представляє собою модифікацію для мови програмування C (сі), що й відображено в назві, де ++ - зображає операцію інкременту для мови C. Одним з найбільших та важливіших нововведень стала підтримка концепцій Об'єктно-Орієнтованого Програмування (ООП), що істотно розширило й без того великі можливості мови C. В результаті отримали дуже потужний інструмент для написання програних засобів, який існує вже

майже 40 років, постійно розвивається та отримує новий функціонал там можливості, наприклад це одна з перший мов програмування, яка отримала вбудовану в стандартну бібліотеку підтримку багатопотоковості (multithreading) [14].

Додаванням нового функціоналу та підтримкою старого у мові C++ керує міжнародний Комітет стандартизації C++. До нього входять найкращі програмісти з усього світу, які на спеціальних засіданнях вирішують, які нові можливості слід додати до стандартної бібліотеки мови C++ у наступному стандарті. На даний час, починаючи з C++11, нові стандарти узгоджують один раз на три роки. Останній на сьогодні стандарт – C++20, який приніс підтримку модулів та корутин [15].

Мова програмування C++ має чи не найширшу область використання з усіх мов, на C++ написані операційні система, драйвери для комп'ютерної периферії, навігаційні програми для величезних океанських танкерів, програми керування для промислової робототехніки, програми для медецинських приладів, а окрім цього і безліч звичайних програм для комп'ютерів – браузери для доступу в Інтернет, відеоігри, програмні засоби для відтворення та редагування медіафайлів, тощо [16].

Виходячи з наведеної вище інформації, найкращою мовою для написання експертної системи буде C++. На момент початку написання системи стандарт C++20 ще був у розробці, через це вона написана з використанням стандарту C++17.

Експертна система повинна мати достатньо дружній до користувача **інтерфейс** (User Interface – UI), тому варіант з інтерфейсом у вигляді командного рядка (Command-Line Interface – CLI) не задовольняє вимогам. Отже потрібно знайти бібліотеку або фреймворк для розробки зручного графічного інтерфейсу користувача (Graphical User Interface – GUI).

Для пошуку необхідної бібліотеки можна скористуватись інтернет-сайтом cppreference.com [17], який взагалі містить документацію для усіх існуючих стандартів C++ та стандартної бібліотеки.

У спеціальному розділі на цьому сайті міститься список різних бібліотек з відкритим джерелом (open-source) від сторонніх розробників. Бібліотек у списку велика кількість, є для роботи з аудіо, графікою, ГКІ, тощо.

У розділі списку з бібліотеками призначеними для створення графічних користувацьких інтерфейсів більше 20 найменувань. Серед них можна виділити наступні:

- FLTK (Fast Light ToolKit) – кросплатформенний інструментарій для створення графічних інтерфейсів. Пропонує сучасний функціонал для графічних рішень. Підтримує тривимірну графіку. Збудований на технології OpenGL. Має редактор для створення графічних форм [18].
- nanogui – інструмент для створення мінімалістичних користувацьких інтерфейсів. Збудований на OpenGL
- tiny file dialogs – один-єдиний заголовний файл ".h", який містить всього 8 функцій. Зручний та простий інструмент для створення простого інтерфейсу, що передбачає якусь навігацію у файловій системі. Може використовуватись на ОС Windows, GNU/Linux, macOS [19].
- Ultimate++ – потужний інструментарій для створення великонавантажених користувацьких інтерфейсів. Містить в собі широкий список готових віджетів, які за допомогою зручного конструктора можна дуже легко об'єднувати для створення необхідних вікон. Також кросплатформенний [20].
- Qt – один з найбільш поширених фреймворків для створення графічних інтерфейсів користувача. Побудований на основі технології OpenGL. Разом з фреймворком йде також і власний редактор коду, що робить цей фреймворк чи не найзручнішим з усіх. Можна просто завантажити з сайту редактор QtCreator і одразу почати створювати свої програми, з сучасним графічним інтерфейсом, у створенні якого допомагає зручний і потужний редактор графічних форм. А якщо щось не зрозуміло – існує спеціальний розділ з великою кількістю готових

прикладів, які можна завантажити у вигляді проектів та продивитись вихідний код програми [21].

З наведеної вище інформації видно, що найкращим інструментом для створення графічного інтерфейсу для експертної системи буде Qt та редактор QtCreator. Про якість даного фреймворку кращим чином говорять сфери його застосування – він використовується для створення інтерфейсів телевізорів та іншої побутової техніки в компанії LG, для створення інтерфейсів сучасних мультимедійних систем в автомобілях компаній Peugeot та Mercedes-Benz, та безліч інших побутових, корпоративних та промислових пристроїв у різних компаніях світу, від маленьких стартапів до корпорацій зі світовим ім’ям.

Також варто відмітити що сфера застосування фреймворку Qt не обмежується лише графічними інтерфейсами користувача, можна його використовувати і для програм з інтерфейсом командного рядка, а також писати програми для вбудованих систем, які взагалі не мають інтерфейсу в звичному користувацькому понятті.

2.2. Проектування експертної системи

Робота експертної системи заключюється в отриманні на вхід речення, яке в спеціальну форму вводить користувач. Потім спеціальний алгоритм шукає в реченні слова, що належать до говірки жителів міста Гнівань та не зустрічаються у літературній український мові. Ці слова заносяться у спеціальний список, а у поле виводу записується речення, в якому говіркові слова заміщуються словами з літературної мови.

Список має додаткову функцію – при подвійному натисканні на слово зі списку, в спеціальне поле записується слово, що є літературним відповідником для говіркового слова.

Алгоритм може працювати в зворотньому напрямку. Тобто на вхід вже буде очікуватись речення, що складається з літературних слів, а на вихід буде

подаватись речення, в якому усі літературні слова заміщені аналогічними словами з Гніваньської говірки, якщо такі відповідники існують.

Словник представляє з себе таблицю з двух стовпців: у першому записується говіркове слово, у другому його літературний відповідник.

Звісно говірка має обмежений запас унікальних слів, таких що не зустрічаються в літературній мові. Таким чином, якщо алгоритм зустрічає слово, якого не існує в словнику, то просто ігнорує його, припускаючи, що це літературне слово, яке і не потребує заміщення. Завдяки цьому підходу словник включає в себе тільки унікальні говіркові слова, та літературні відповідники, тому і його розмір, принаймні для Гніваньської говірки, достатньо невеликий.

Основне вікно експертної системи побудовано за принципом вкладок, як сучасні веб-браузери. Перша вкладка містить інтерфейс саме для системи аналізу речень на предмет діалектних слів. Додатково відкриваються три вкладки інформаційно-довідкового характеру: «Морфологічні риси» та «Фонетичні риси» - про морфологічні та фонетичні відмінності говірки міста Гнівань від літературної мови, а вкладка «Історія міста» розповідає коротку історію міста Гнівань Вінницької області.

Також потрібна окрема вкладка, котра буде відображати таблицю з усіма наявними у словнику говірковими словами та їх літературними відповідниками. Додатково під таблицею два текстових поля: перше для говіркового слова, друге для говіркового відповідника. Якщо ввести слово в одну з них (або в обидва одночасно) та натиснути спеціальну кнопку, то стрічка словника, що містить ці слова видалиться. Якщо заповнити обидва поля та натиснути кнопку «Додати» у словнику з’явиться відповідний запис. Якщо запис включаючий одне з цих слів вже існує то такий запит ігнорується. Аналогічно якщо при видаленні запитуваного слова в словнику не знайдено запит так само ігнорується. Користувач зможе про це дізнатись зі спеціального рядка статусу, який відображає поточний стан програми.

При роботі з експертною системою також можливо завантажити новий словник, вказавши до нього шлях у спеціальному діалоговому вікні. При цьому,

як і при виході з системи, поточний словник зберігається у директорію програми.

Для тексту, який користувач вводить в поле запиту, а також для тексту з поля результату передбачається можливість озвучування програмою, тобто при натисканні на відповідну кнопку поруч з текстовим полем експертна система, за допомогою спеціальної функції, яку надає фреймворк Qt, зачитує так званим «комп'ютерним голосом» текст з відповідного текстового поля.

Також текст можна ввести за допомогою голосового вводу, але через неможливість навчання власної системи, яка б включала в себе і говіркові слова, текст введений таким чином буде включати в себе лише літературні слова. Але через те що експерта система передбачає роботу у зворотньому напрямку, такі речення можна аналізувати на предмет кількості слів, які можна було б замінити на говіркові.

2.3. Реалізація алгоритму заміщення говіркових слів

Починаючи роботу над алгоритмом з метою закласти гарну базу для подальшого масштабування та модифікації алгоритм був написаний у окремому заголовному файлі. Завдяки цьому даний алгоритм може бути використаний взагалі без графічного користувацього інтерфейсу.

При використанні з інтерфейсу командного рядка програма очікує на вхід певне речення, яке вводить користувач. При натисканні на клавішу «Enter» рядок зчитується і алгоритм починає роботу. По мірі обробки введені слова по черзі записуються у командний рядок. Якщо слово не знайдено у словнику – воно записується без жодних змін. У випадку коли слово міститься у говірковому словнику – воно виводиться не зміненим, але в консолі друкується червоними літерами.

Паралельно з цим слова заносяться у спеціальну внутрішню змінну. Таким чином літературні слова записуються так само без змін, а замість слів з говірки записуються їх літературні відповідники.

Після завершення аналізу виводиться і відредаговане речення. Таким чином можна бачити які самі слова були заміщені, та їхні відповідники.

При використанні алгоритму у експертній системі, котра має графічний користувацький інтерфейс, не відбувається записування речення з підфарбованими словами до консолі. Натомість слова які знаходяться у словнику Гніваньської говірки заносяться в окремий контейнер типу std::vector<std::string> з стандартної бібліотеки мови C++. Після завершення аналізу відредаговане речення виводиться в спеціальне тестове поле відповіді, а усі знайдені говіркові слова записуються у спеціальний список, який дає змогу окремо для кожного слова отримувати його літературний відповідник.

Робота власне алгоритму заключається в покроковому розбитті введеного речення на окремі слова, які заносяться у контейнер std::vector<std::string>, при цьому знаки пунктуації ігноруються. Далі перевіряється умова «прямого заміщення», іншими словами визначається які слова потрібно знайти та замістити – говіркові чи літературні відповідно. Після цього у циклі перевіряється кожне слово з раніше занесених до контейнера. Якщо перевірене слово не знаходиться у словнику, то воно записується у спеціальний рядок без змін, у протилежному випадку в рядок записується літературний відповідник знайденого говіркового слова (або говіркове слово, у випадку оберненого аналізу). Разом з цим слова, що відповідають критерію пошуку заносяться одночасно і в список заміщених слів.

2.4. Розробка основного інтерфейсу експертної системи

Розробка графічного інтерфейсу користувача почалася з узгодження макета цього інтерфейсу, який зображав розміщення основних елементів користувацького інтерфейсу у вікні експертної системи.

Перед початком безпосередньої розробки експертної системи в редакторі QtCreator був створений не публічний репозиторій у мережі GitHub. Ця система дозволяє контролювати версії розроблюваної програми, що допоможе

виправити якісь серйозні зміни, що призвели до фатальних проблем у коді програми. Також система контроля версій може врятувати у випадку яких-небуть форс-мажорних ситуацій, наприклад помилка жорстокого диска на комп’ютері, що призвела до втрати усіх файлів.

Завантаження змін у репозиторій проводилось за допомогою інтерфейсу командної стрічки Git Bash. Також існує стандартна версія з графічним інтерфейсом та низка окремих програм для контролю версій через git від сторонніх розробників.

Спочатку була створена система вкладок. Зі стандартного шаблону було видалено усе, окрім стрічки статусу.

Далі на основне вікно програми був перенесений об’єкт QTabWidget, який представляє собою батьківський віджет для створення інтерфейсу на основі вкладок. Цей віджет був розтягнутий на все вікно програми, окрім рядка статусу, через те, що програма не передбачає ніяких інших елементів інтерфейсу поза зоною віджета вкладок.

Для розтягування та масштабування доречно змінювати значення параметрів «geometry» та «sizePolicy». Також параметр «currentIndex» варто зробити рівним нулю, для того щоб при кожному відкритті вікна відкривалася перша вкладка (нумерація починається з нуля)

Після створення базової системи були додані 4 вкладки: одна вкладка для аналізатора та ще три вкладки інформаційно-довідкового характеру.

На вкладку аналізатора в першу чергу були додані два віджета класу QTextEdit, які розмістилися в лівій стороні вікна один над одним. Були підібрані оптимальні параметри шрифту для зручного читання та введення тексту. Для цього доцільно змінювати параметри з розділу «Font». Також для параметру «locale» необхідно встановити значення відповідно української мови. Усі параметри для цих віджетів варто робити ідентичними, для зручності кінцевого користувача, окрім параметру «readOnly». Для текстового поля, яке призначене для виведення тексту із заміщеними говірковими словами значення

даного параметру варто встановити як істинне, щоб кінцевий користувач не міг редагувати отриманий текст, але міг його зручно читати та копіювати.

Після цього була додана, за допомогою віджета QPushButton, кнопка, яка запускає виконання алгоритму. Вона розміщена одразу під текстовим полем виводу, що дуже зручно і одразу зрозуміло користувачеві, а значить гарантує позитивний досвід використання програми.

Для розширення функціоналу експертної системи був створений механізм запуску алгоритму в зворотній бік. Для того, щоб користувач міг сам обирати, в яку сторону він хоче виконувати аналіз був доданий віджет класу QCheckBox, який являє собою перемикач, котрий існує в двох положеннях. Для зручності даний віджет був розміщений праворуч від кнопки. За замовченням він вимкнутий.

Далі був доданий віджет QlistWidget, який буде містити в собі усі знайдені в початковому тексті діалектні слова (або літературні слова, для оберненого запуску). За замовченням список ініціалізується пустим, очікуючи на введення тексту та запуст аналізу.

Під списком розмістився об’єкт класу QLabel, який представляє з себе звичайну текстову стрічку, без будь-якого функціоналу зі сторони користувача. За замовченням стрічка пуста, а при натисканні на будь-яке слово і списку стрічка приймає значення літературного чи говіркового відповідника зі словнику, в залежності від напрямку роботи алгоритму.

Після цього були додані ще три такі ж об’єкти для підпису обидвох текстових полів та списку. При переключенні режиму роботи програми підписи текстових полів міняються місцями (з точки зору програми просто обмінюються текстом за замовченням), а підпис списку просто змінює своє значення на те, що вказано в коді програми.

Отже на цьому етапі роботу над основною вкладкою експертної системи можна вважити завершеною. Ця вкладка прийняла такий вигляд:

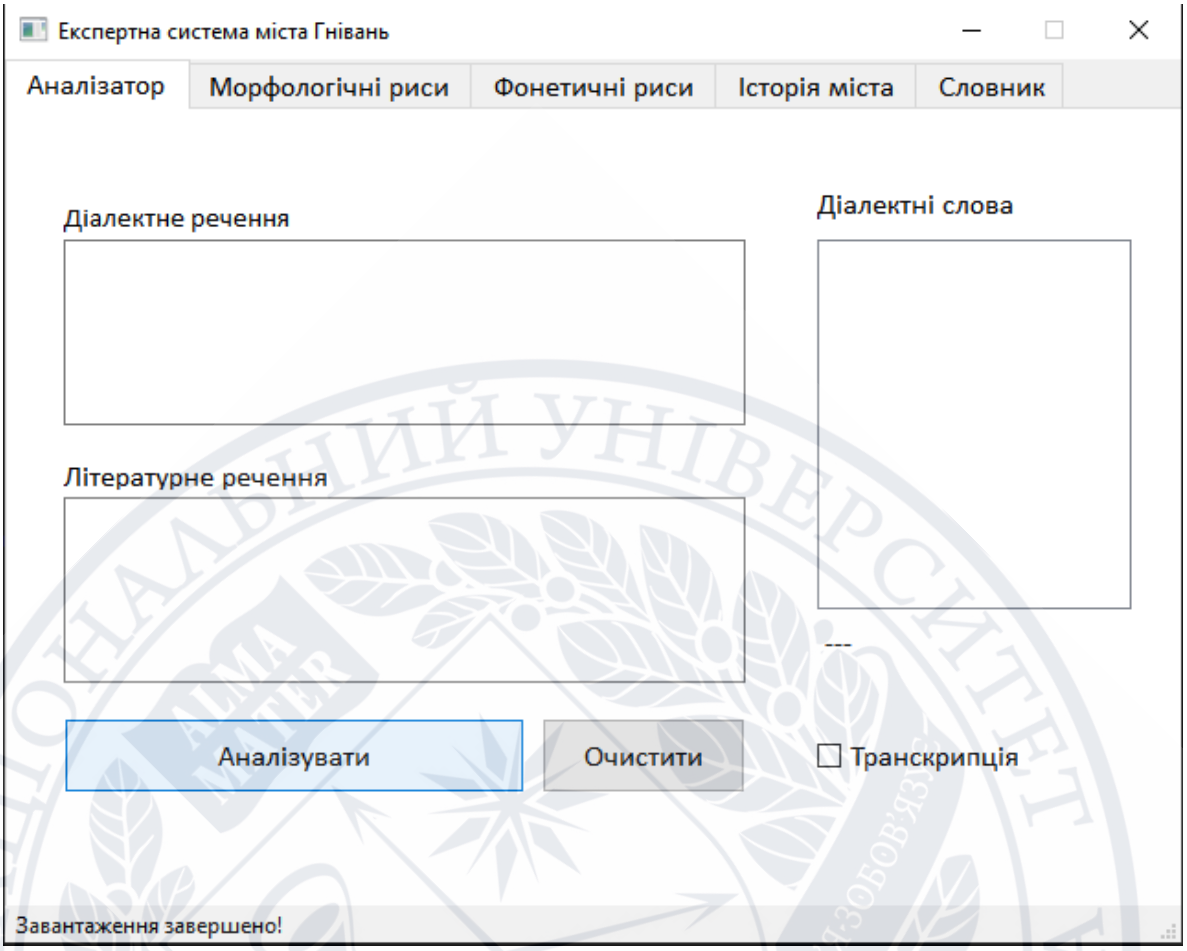

Рис. 2.1. Основне вікно та вкладка експертної системи.

Далі почалося оформлення інформаційних вкладок. Всі три вкладки були оформлені однаково, тому буде описаних лише процес для вкладки «Морфологічні риси», наступні дві оформлені аналогічно, змінюється лише текст наповнення.

Був доданий віджет з класу QPlainTextEdit. Він був розтягнутий майже на всю площину вкладки, адже там все одно нічого іншого не має бути. Для зручності користувача виставлені найбільш комфортні параметри групи «Font». Параметр «readOnly» за замовченням включений, адже це довідковий текст, що виключає потребу редагування зі сторони користувача. Далі у віджет був записаний текст відповідної наукової статті, щоб користувач зміг отримати необхідну інформацію стосовно Гніваньської говірки.

Інформаційно-довідкові вкладки оформлені однаково, тому одного скріншоту буде достатньо. Для прикладу приводиться скріншот вкладки «Морфологічні риси»:

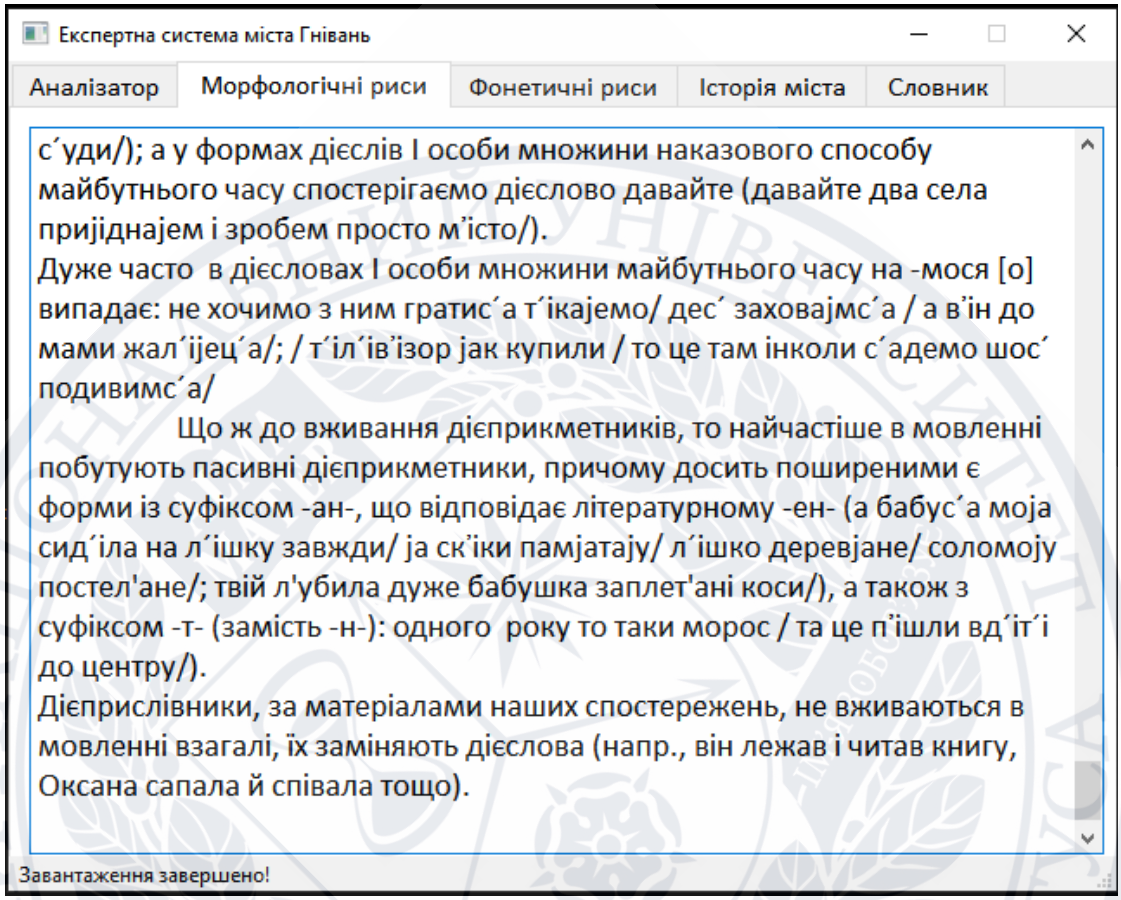

Рис. 2.1 Вкладка "Морфологічні риси".

Використавши віджет QListWidget було створено вкладку «Словник». Ця вкладка відображає повний словник, за яким у даний час виконується аналіз тексту на предмет говіркових слів. Виглядає вона наступним чином:

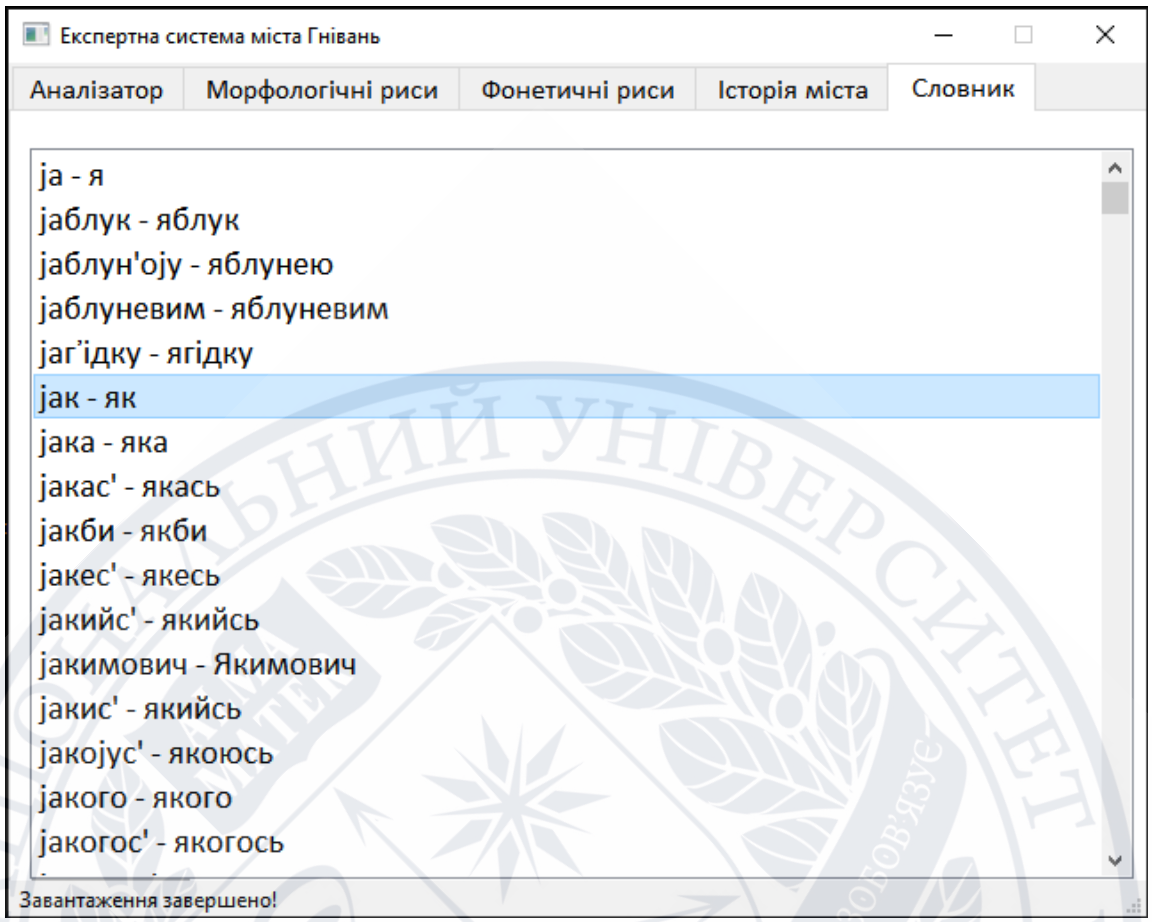

Рис. 2.2 Вкладка "Словник".

На майбутнє можна зробити це поле за замовченням пустим, а сам текст завантажувати при кожному запуску програми. Це нічого не змінить з точки зору користувача, але значно спростить роботу для програміста при подальшому розвитку експертної системи. Наприклад для різних говірок можна підготувати декілька відповідних статей та словників, та кожен раз на старті прогами запитувати користувача яку говірку саме зараз потрібно завантажити. Таким чином можна не вносити жодних змін у код програми, а просто додавати нові набори файлів для нових говірок та легко масштабувати експертну систему.

РОЗДІЛ 3

МОЖЛИВОСТІ ДЛЯ ПОДАЛЬШОГО РОЗВИТКУ ТА МАСШТАБУВАННЯ ЕКСПЕРТНОЇ СИСТЕМИ

3.1. Створення ботів-коректорів для онлайн месенджерів

Алгоритм, написаний для цієї експертної системи, можна використати й поза її межами. Експертна система сама по собі має лише одне практичне використання – вирішення задач прикладної лінгвістики. Конкретно в поточній реалізації система ще й обмежена рамками лише української мови на гніваньського діалекту. Але гнучка архітектура, на якій побудована експертна система, дозволяє швидко змінювати діалект. Також можна змінювати і мови, але в такому випадку, задля зручності кінцевого користувача, варто буде додати локалізацію іншими мовами, наприклад англійською або німецькою.

Проте, як було неодноразово вказано вище, алгоритм сам по собі здатен працювати з будь-якими мовами, без жодних змін. Тобто якщо прибрати середу використання (експертну систему) – то алгоритм можна в існуючому вигляді використовувати в будь-якій іншій середі, де він може бути потрібен.

Одним з варіантів, або правильніше сказати списком варіантів, є онлайн месенджери, або ж чати. Наприклад Telegram, Viber, WhatsUp, тощо. Як правило, у всіх популярних месенджерах з великою аудиторією є можливість (відкриті API) для створення користувацьких ботів. Спектр використання цих ботів неймовірно широких. Їх використовують для рекламних компаній, розваг, пошуку будь-якої інформації в певних базах даних, як службу підтримки для заміни застарілих колл-центрів, та ще безліч ймовірних застосувань. Ботів можна використовувати як в особистому листуванні, так й у великих групових чатах з необмеженою кількістю учасників.

В таких чатах з великою кількістю учасників швидкість набору тексту частіше переважає бажання писати правильно, люди скорочують слова, замінюють їх на запозичені з інших мов, як правило, з англійської. В якихось конкретних чатах, наприклад корпоративних, такий стиль листування може бути просто неприпустимим, але відмовитися від звички використовувати скорочення та запозичені слова, що замінюють літературні аналоги.

Вирішенням цієї проблеми може стати спеціальний бот. При додаванні його в бесіду потрібно передавати йому користувацький словник, в який будуть входити небажані слова. Це можуть бути як певні сленгові слова, так і якісь діалектні слова, або ненормативна лексика. Іншими словами будь який набір слів або фраз, які адміністратори чату вважають небажаними у своїй бесіді. Варто також передбачити функції додавання та видалення слів з існуючого словника, щоб додати більше гнучкості у використанні.

Сценарій застосування такого бота досить простий: при додаванні в бесіду він отримує доступ до усіх текстових повідомлень, які надсилають користувачі. Кожне нове повідомлення буде одразу ж зчитуватись на оброблюватись ботом на предмет наявності в ньому слів зі словника. Далі є два варіанти: можна окремим повідомленням писати знайдені неправильні слова, та їх бажані літературні аналоги для заміни (це повідомлення можна надсилати в бесіду одразу після повідомлення користувача, або одразу користувачу в приватні повідомлення), або можна видаляти повідомлення користувача, та надсилати замість нього копію зі вказанням автора повідомлення, та літературними словами, замість тих, що адміністратори бесіди вважають небажаними.

Також можна вести статистику використання небажаних слів, наприклад які слова вживаються найчастіше загалом, або який користувач найчастіше використовує ці слова.

На даний час не подібних ботів у загальному доступі не існує. Найближчий аналог – боти-контролери, які також аналізують надіслані речення за певними словниками (як правило, пропріетарними), але вони

використовуються як захист від спаму, тобто реагують на ненормативну лексику, блокуючи користувачів, що її використовують. Тому цю нішу можна вважати вільною, та розвити ідею експертної системи у цьому напрямку.

3.2. Додаток для розумних годинників або смартфонів

Як було зазначено у першому розділі, системи розпізнавання мовлення поділяються на два типи: загальної направленості, які спочатку навчаються на великій кількості дикторів, з ціллю розпізнавання мовлення будь-якої людини; або ж системи по типу віртуальних помічників або розумного будинку (так званий «інтернет речей»). Останні навчаються розпізнавати мовлення конкретного кінцевого користувача, для того щоб виконувати його команди.

Систему такого типу можна використати і для корекції мовлення. Наприклад користувач хоче зменшити кількість ненормативних слів або слів-паразитів у своїй лексиці. Для полегшення цієї задачі можна створити спеціальний додаток, який можна буде встановити на смартфон або розумний годинник. Цей додаток має давати користувачеві інструменти для створення власного словника небажаних слів. Ці слова потрібно буде декілька разів промовити у мікрофон, щоб програма запам’ятала промову цих слів саме від користувача.

Далі програма має постійно працювати у фоновому режимі, та сканувати аудіо сигнали з мікрофону пристрою. Коли додаток буде розпізнавати слова зі словнику можна подавати звуковий або вібро-сигнал. Також можна вести статистику використання слів, щоб користувач міг бачити свою динаміку.

Ці варіанти значно розширюють початкову сферу використання експертної системи від суто наукової, до достатньо широкого спектру прикладних варіантів застосування. При достатньому рівні підготовки такі програмні рішення мають навіть комерційний потенціал: саме використання додатків може бути безкоштовним, а готові словники можна продавати.

ВИСНОВКИ

У даній кваліфікаційній (бакалаврській) роботі був досліджений та описаний процес розробки власної системи розпізнавання мовлення. Визначена проблематика і складнощі з якими можуть зіштовхнутися ентузіасти або наукові дослідники, яких з певний причин не влаштовують пропріетарні системи розпізнавання мовлення, та які бажають створити свою власну, котра буде відповідати саме їх вимогам.

В практичній частині було створено експертну систему для аналізу діалектів української мови на прикладі говірки жителів міста Гнівань. Описано покроково процес створення системи від узгодження ескізу графічного інтерфейсу користувача, до компіляції готової експертної системи.

Також розглянуто можливості подальшого розвитку та масштабування експертної системи, для розширення науково-дослідницьких можливостей системи, або навіть створення систем для загального використання широким колом користувачів.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Баклан І. В. Експертні системи: курс лекцій/ навчальний посібник. м. Київ: НАУ, 2012, 132 с.
2. Вавіленкова А. І. Розробка експертної системи на базі методу автоматизованого формування логіко-лінгвістичної моделі текстової інформації. Проблеми автоматизації та управління: збірник наукових праць. м. Київ: НАУ, 2009, с. 14-19.
3. Вдовиченко А. В. Морфологічні особливості говірки міста Гнівань. Лінгвоукраїністика XXI століття: традиції та новаторство: зб. наук. праць/ відп. ред. Л. М. Коваль. Вінниця: ДонНУ ім. В. Стуса, 2018. Вип. 1. с. 120-133.
4. Вдовиченко А. В. Фонетичні особливості говірки міста Гнівані Вінницької області: підсистема приголосних. Вісник СНТ. Вінниця, 2018. Вип. 10. с. 93-96.
5. Вдовиченко А.В. До проблеми використання експертних систем у лінгвістиці. Розвиток суспільства та науки в умовах цифрової трансформації 61: матеріали міжнародної студентської наукової конференції, м. Одеса, 8 травня 2020 рік: Молодіжна наукова ліга, 2020. Т.4. с. 27-29.
6. Коваль О. І. Гнівань: Історія, сьогодення, майбутнє. Вінниця: Книга-Вега, 2004. 60 с.
7. Ю. Бабуров Deep learning на пальцах 11 – распознавание речи и аудио, презентація, 2019, URL: https://www.dropbox.com/s/tv3cv0ihq2l0u9f/Lecture%2011%20-%20Audio%20and%20Speech.pdf?dl=0
8. Н. Зуєва Распознавание речи (speech2text), Deep Learning School при ФІВТ МФТІ, лекція, 2019
9. І. Бондаренко Распознавание речи: как сделать Speech-To-Text своими руками, конференція HighLoad++ Siberea, 2018

10. INTRODUCTION TO EXPERT SYSTEMS, 2nd edn, P. Jackson, Addison-Wesley, Wokingham, 1990, ISBN 0-201-17578-9, 526 p.
11. E. A. Feigenbaum, P. McCorduck, P. Nii The Rise of the Expert Company: How Visionary Companies are Using Artificial Intelligence to Achieve Higher Productivity and Profits, USA, Times Books, 1988, 322 p.
12. Stroustrup, Bjarne, 2008. Programming: Principles and Practice Using C++ (1st ed.). Addison-Wesley Professional. ISBN 978-0-321-54372-1
13. B. Stroustrup: A History of C++: 1979–1991. Proc ACM History of Programming Languages conference (HOPL-2). ACM Sigplan Notices. Vol 28 No 3, pp 271–298. March 1993
14. B. Stroustrup: Evolving a language in and for the real world: C++ 1991–2006. ACM HOPL-III. June 2007.
15. C++ documentation and help, USA, 2021, URL: https://en.cppreference.com/w/
16. ISO/IEC 14882:2020. Programming language – C++, vol. 6, 2020-12, 1853 p.
17. Qt Creator framework documentation and support, USA, 2021, URL: https://doc.qt.io/
18. Fast Light Toolkit – crossplatform GUI library, 2019, URL: https://www.fltk.org/links.php?LC+P12+Q
19. Ultimate++ framework – C++ GUI library, 2016, URL: https://www.ultimatepp.org/www$uppweb$overview$en-us.html
20. tiny file dialogs (cross-platform C-C++ library), 2021, URL: https://sourceforge.net/projects/tinyfiledialogs/
21. x86 Assembly language Reference Manual, Oracle, 2010, URL: https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf
22. Python3 programming language documentation, Python Software Foundation, 2021, URL: https://docs.python.org/3/
23. The main Python3 internet-site, Python Software Foundation, 2021, URL: https://www.python.org/

24. D. Amos The Ultimate Guide To Speech Recognition With Python, RealPython, 2019, URL: https://realpython.com/python-speech-recognition/#how-speech-recognition-works-an-overview

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Б. І. В., Експертні системи: курс лекцій/ навчальний посібник, Київ: НАУ, 2012, p. 132.

[2] P. M. P. N. E. A. Feigenbaum, The Rise of the Expert Company: How Visionary Companies are Using Artificial Intelligence to Achieve Higher Productivity and Profits, USA: Times Books, 1988, p. 322.

[3] В. А. І., Розробка експертної системи на базі методу автоматизованого формування логіко-лінгвістичної моделі текстової інформації. Проблеми автоматизації та управління: збірник наукових праць, Київ: НАУ, 2009, pp. 14-19.

[4] В. А. В., Морфологічні особливості говірки міста Гнівань. Лінгвоукраїністика XXI століття: традиції та новаторство: зб. наук. праць, 1 ред., К. Л. М., Ред., Вінниця: ДонНУ ім. В. Стуса, 2018, pp. 120-133.

[5] В. А. В., Фонетичні особливості говірки міста Гнівані Вінницької області: підсистема приголосних, 10 ред., Вінниця: Вісник СНТ, 2018, pp. 93-96.

[6] З. Н., «Распознавание речи (speech2text),» Deep Learning School при ФІВТ МФТІ, 2019.

[7] Б. Ю., «Deep Learning на пальцах 11 - распознавание речи и аудио,» 2019.

[8] Б. І., «Распознавание речи: как сделать Speech-To-Text своими руками,» HighLoad++ Siberea, 2018.

[9] A. D., «The Ultimate Guide To Speech Recognition With Python,» RealPython, 2019. [Онлайновий]. Available: https://realpython.com/python-speech-recognition/#how-speech-recognition-works-an-overview.

[10] «The main Python3 internet-site,» Python Software Foundation, 2021. [Онлайновий]. Available: https://www.python.org/.

[11] «Python3 programming language documentation,» Python Software Foundation, 2021. [Онлайновий]. Available: https://docs.python.org/3/.

[12] «x86 Assembly Language Reference Manual,» Oracle, 2010. [Онлайновий]. Available: https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf.

[13] S. B., A History of C++:1979 - 1991., Vol. 28, No. 3 ред., Proc ACM History of Programming Languages conference (HOPL-2) ACM Sigplan Notices, 1993

March, pp. 271-298.

[14] S. B., Evolving a language in and for the real world: C++ 1991–2006, ACM HOPL-III, 2007 June.

[15] ISO/IEC 14882:2020. Programming language – C++, 6 ред., ISO/IEC, 2020-12, p. 1853.

[16] S. B., Programming: Principles and Practice Using C++, 1st ред., Addison-Welsey Proffesional, 2008.

[17] C++ documentation and support, 2021.

[18] «Fast Light Toolkit - cross-platform GUI library,» 2019. [Онлайновий]. Available: https://www.fltk.org/links.php?LC+P12+Q.

- «tiny file dialogs (cross-platform C-C++ librery),» 2021. [Онлайновий].
[19] Available: https://sourceforge.net/projects/tinyfiledialogs/.

[20] «Ultimate++ framework - C++ GUI library,» 2016. [Онлайновий]. Available: https://www.ultimatepp.org/www$uppweb$overview$en-us.html.

[21] «Qt Creator framework documentation and support,» The Qt Company, 2021. [Онлайновий]. Available: https://doc.qt.io/.

[22] P. Jackson, INTRODUCTION TO EXPERT SYSTEMS, vol.2, Wokingham: Addison-Wesley, 1990, p. 526.

[23] В. А. В., в *До проблеми використання експертних систем у лінгвістиці. Розвиток суспільства та науки в умовах цифрової трансформації 61*, Одеса, 2020.

[24] К. О. І., Гнівань: Історія, сьогоденння, майбутнє, ВІнниця: Книга-Вега, 2004, p. 60.

ДОДАТКИ

ДОДАТОК А

Скріншоти експертної системи

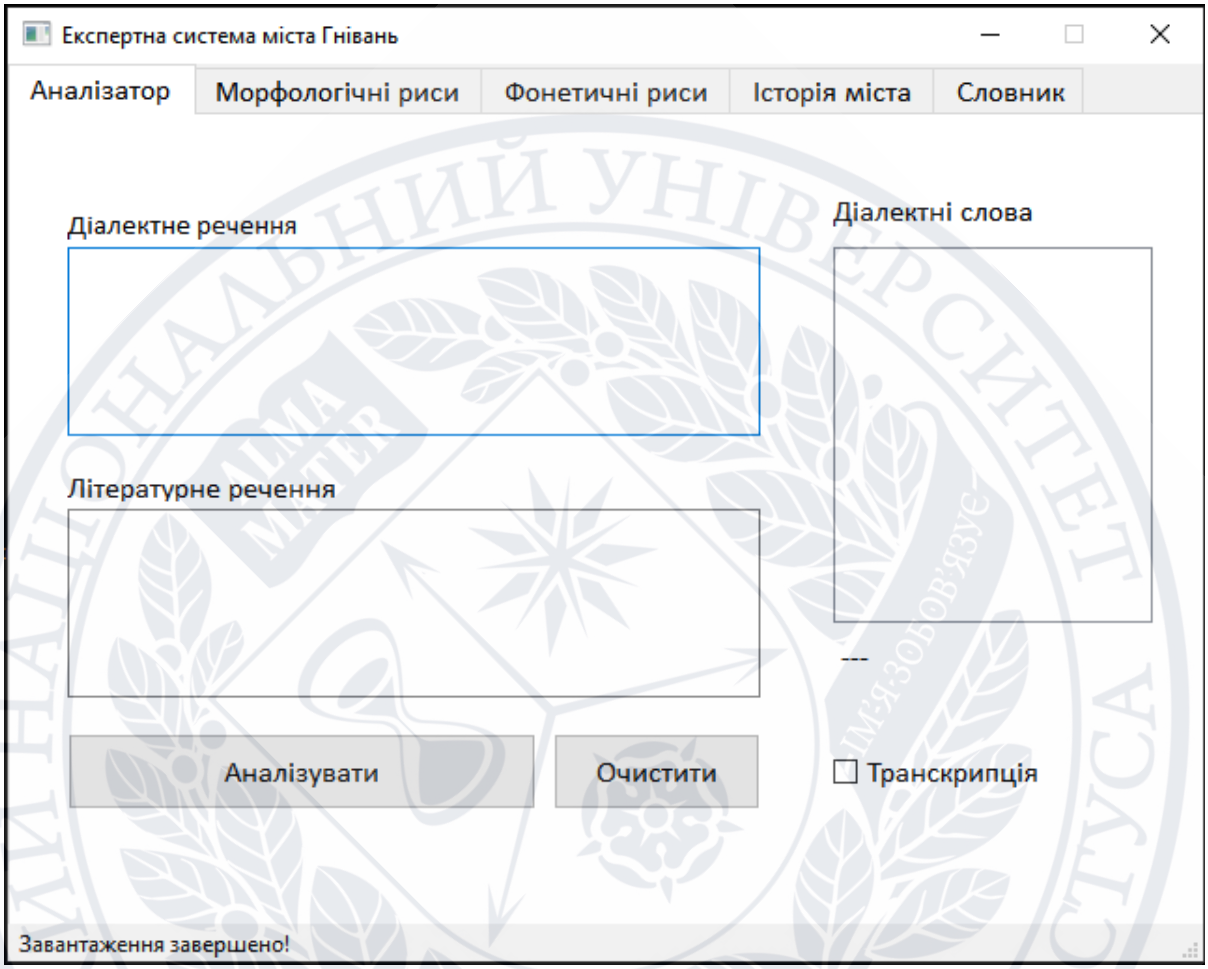

Рис. А-1 Основна вкладка. Експертна система.

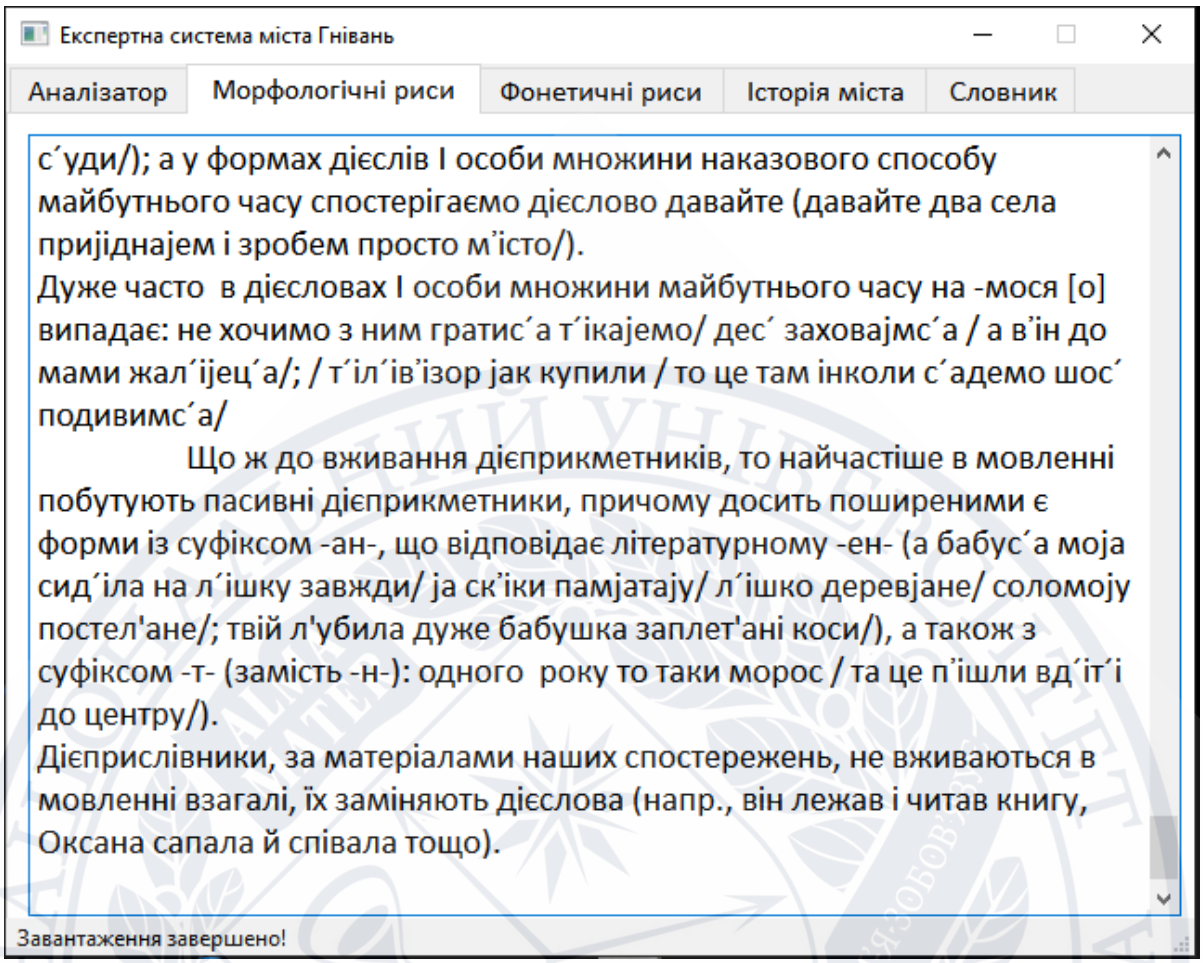

Рис. А-2 Вкладка «Морфологічні риси».

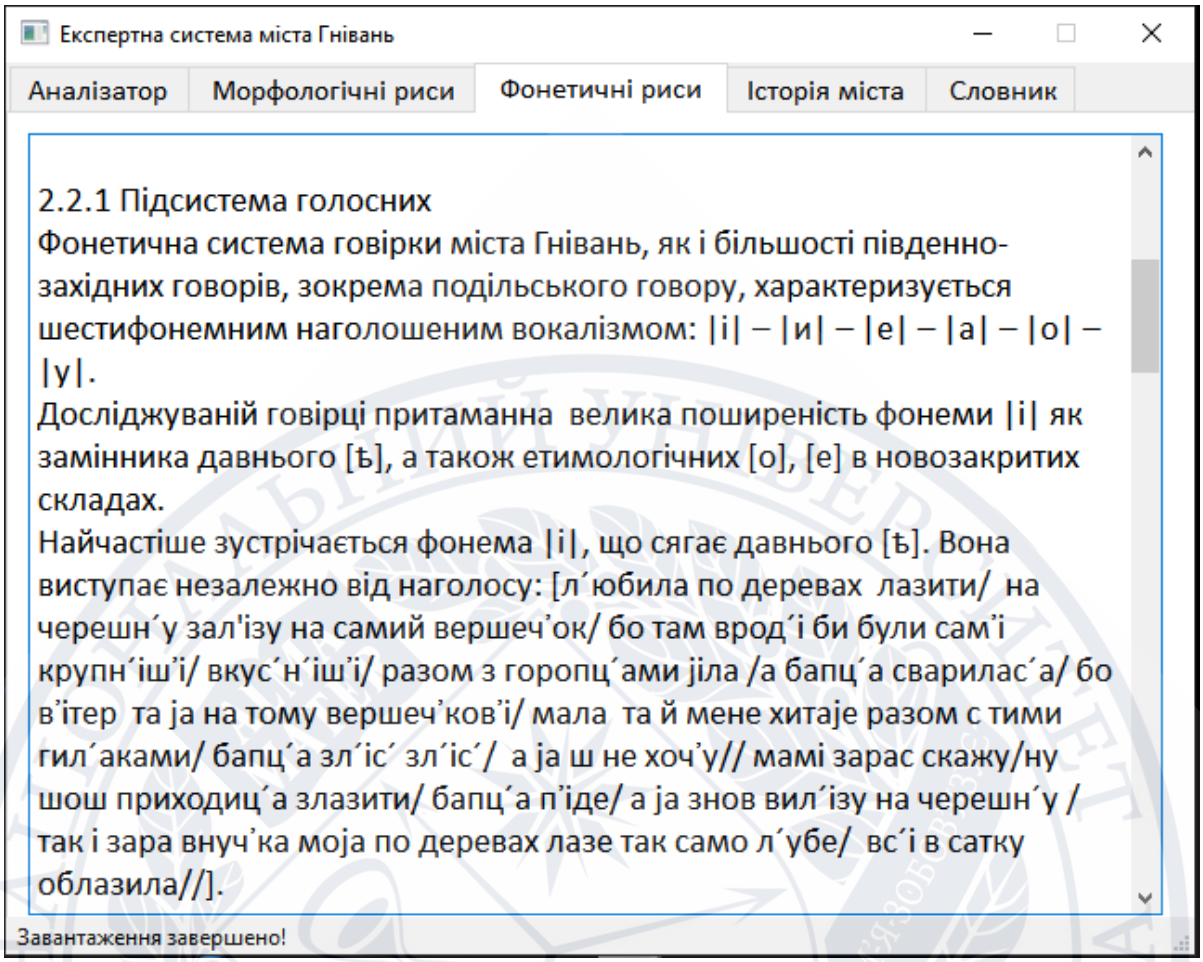

Рис. А-3 Вкладка «Фонетичні риси».

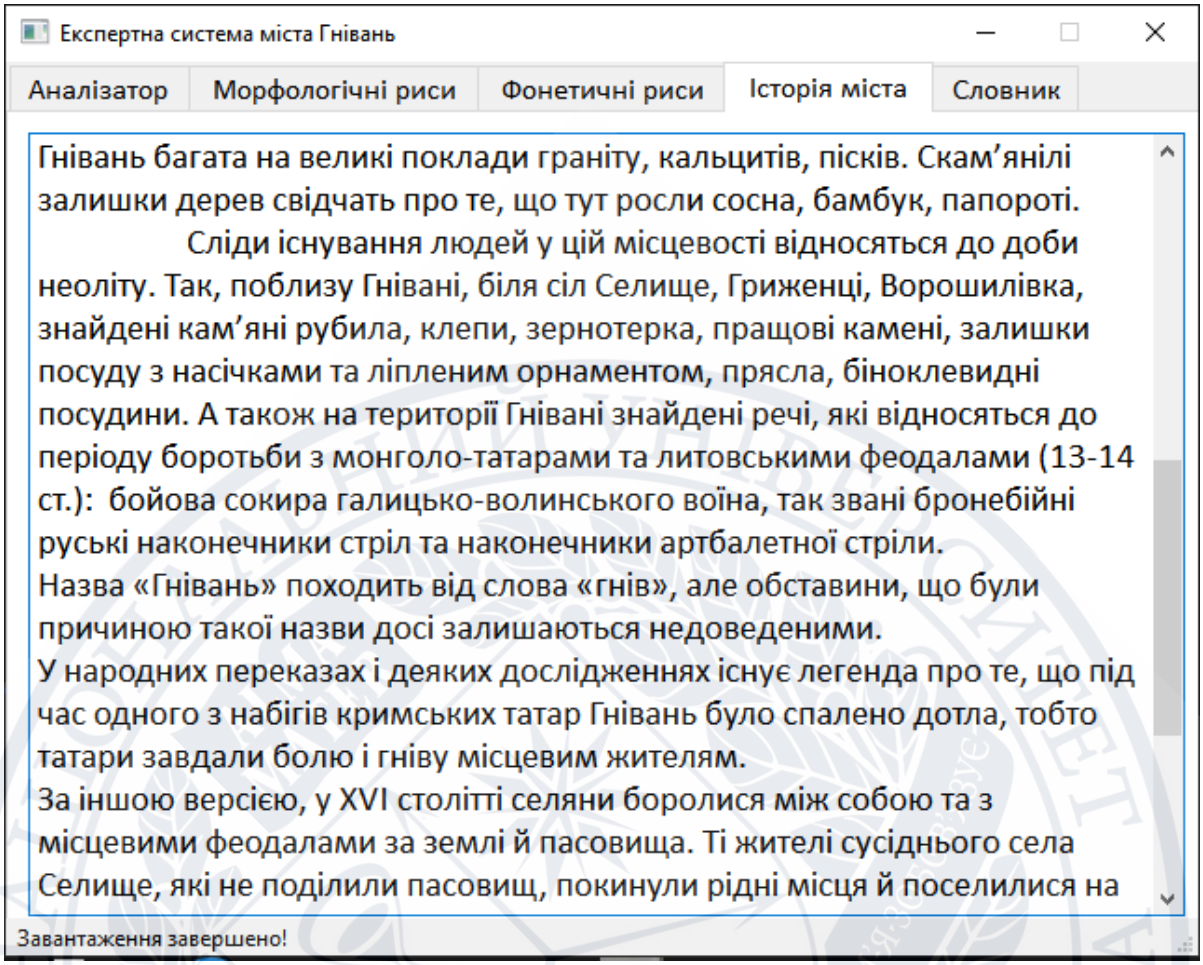

Рис. А-4 Вкладка «Історія міста Гнівань».

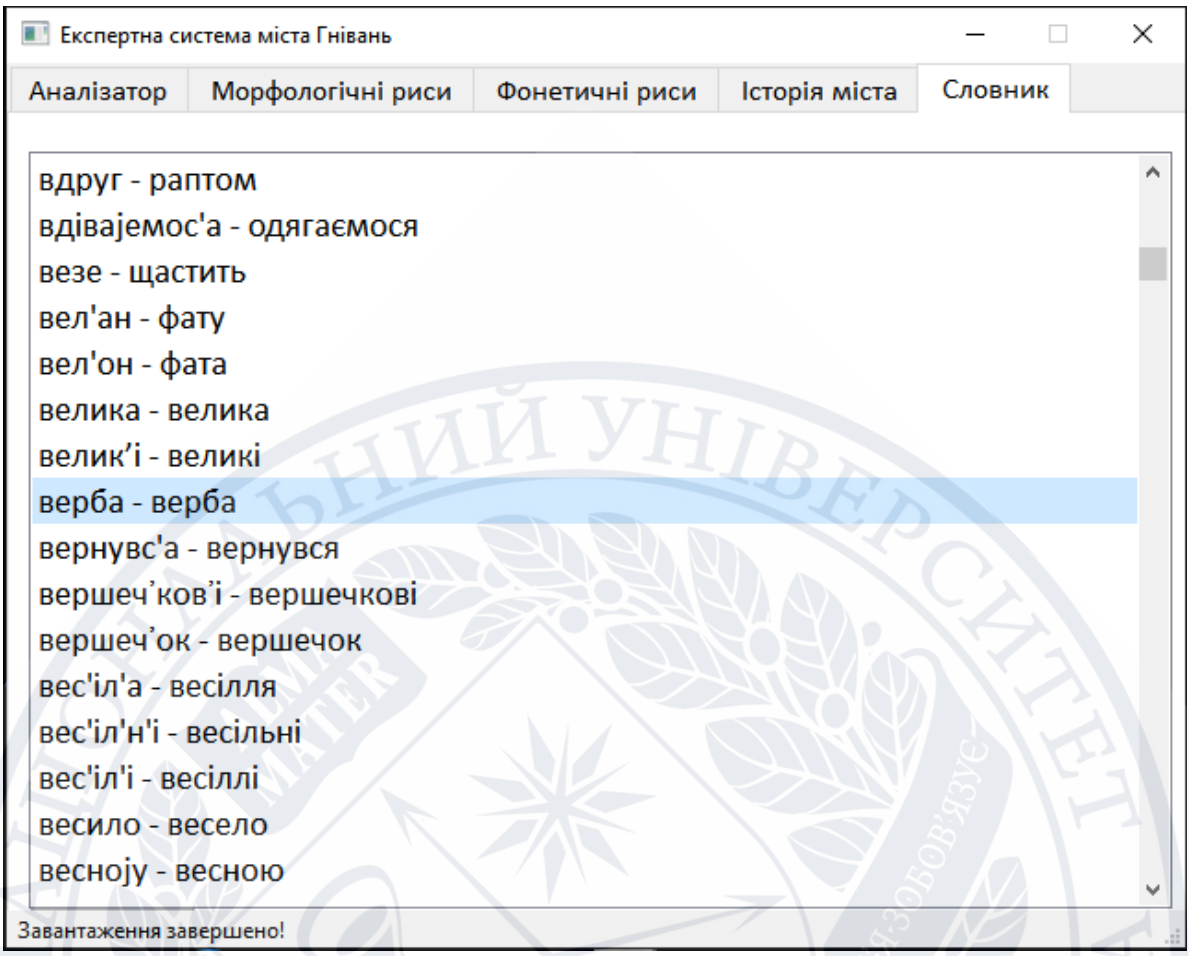

Рис. А-5 Вкладка «Словник».

ДОДАТОК В

Лістинги коду

Лістинг файлу «ExpSys.pro»:

```
QT       += core gui texttospeech

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    Header.h \
    mainwindow.h

FORMS += \
    mainwindow.ui

TRANSLATIONS += \
    ExpSys_uk_UA.ts

# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

DISTFILES += \
    VoiceReconScript.py
```

Лістинг файлу «Header.h»:

```
#ifndef HEADER_H
#define HEADER_H

#pragma once
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
#include <string>
#include <sstream>
#include <Windows.h>
#include <thread>

void LoadDictionary(std::map<std::string, std::string>* dict, std::string
fileName)
{
    std::ifstream fin(fileName);
    std::ofstream fout("test.txt");
    //fin.open(fileName);
    if (!fin.is_open())
    {
        std::cout << "cannot open file" << std::endl;
        return;
    }
    std::string temp1, temp2;
    while (fin)
    {
        std::getline(fin, temp1);
        std::stringstream ss;
        fout << temp1;
        //std::cout << temp1 << std::endl;
        ss << temp1;
        std::getline(ss, temp1, '|');
        std::getline(ss, temp2);

        //std::cout << temp1 << " - " << temp2 << std::endl;
        if (!temp1.empty())
        {
            dict->insert({ temp1, temp2 });
        }
    }
    fin.close();
}

void PrintDictionary(std::map<std::string, std::string>* dict)
{
    std::cout << "-------------" << std::endl;
    for (auto& iter : *dict)
    {
        std::cout << iter.first << " - " << iter.second << std::endl;
    }
    std::cout << "-------------" << std::endl;
}

bool FindDialectWords(std::map<std::string, std::string>* dict, std::string
dial, std::string* lit)
{
    std::string temp;

    for (auto& iter : *dict)
    {
        if (iter.second == dial)
        {
            *lit = iter.first;
```

```
            return true;
        }
    }
    return false;
}

bool ReverseFindDialectWords(std::map<std::string, std::string>* dict,
std::string dial, std::string* lit)
{
    std::string temp;

    for (auto& iter : *dict)
    {
        if (iter.first == dial)
        {
            *lit = iter.second;
            return true;
        }
    }
    return false;
}

/*void DictionaryReverseSearch(std::vector<std::string> &vec, int startIndex,
int endIndex, std::map<std::string, std::string> &dict){
    for (int i = startIndex; i <= endIndex; ++i)
    {
        if (ReverseFindDialectWords(dict, iter, &correctWord))
        {
            temp.append(correctWord);
            temp.append(" ");
        }
        else
        {
            temp.append(iter);
            temp.append(" ");
        }
    }
}*/

void CorrectSentence(std::string* sent, std::map<std::string, std::string>*
dict, bool reverse)
{
    //HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    std::vector<std::string> vec;
    std::stringstream ss;
    std::string str, temp;
    ss << *sent;
    while (ss)
    {
        std::getline(ss, str, ' ');
        vec.push_back(str);
    }
    vec.pop_back();

    std::string correctWord;
    if (reverse)
    {
        //std::thread firstThread
        for (auto& iter : vec)
        {
            if (ReverseFindDialectWords(dict, iter, &correctWord))
            {
                //SetConsoleTextAttribute(hConsole, 4);
                //std::cout << iter << " ";
```

```
                temp.append(correctWord);
                temp.append(" ");
            }
            else
            {
                //SetConsoleTextAttribute(hConsole, 7);
                //std::cout << iter << " ";
                temp.append(iter);
                temp.append(" ");
            }
        }
    }
    else
    {
        for (auto& iter : vec)
        {
            if (FindDialectWords(dict, iter, &correctWord))
            {
                //SetConsoleTextAttribute(hConsole, 4);
                //std::cout << iter << " ";
                temp.append(correctWord);
                temp.append(" ");
            }
            else
            {
                //SetConsoleTextAttribute(hConsole, 7);
                //std::cout << iter << " ";
                temp.append(iter);
                temp.append(" ");
            }
        }
    }
    //std::cout << std::endl;
    *sent = temp;
    //std::cout << *sent << std::endl;
    return;
}

void ReverseCorrect(std::string* sent, std::map<std::string, std::string>* dict)
{
    //HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    std::vector<std::string> vec;
    std::stringstream ss;
    std::string str, temp;
    ss << *sent;
    while (ss)
    {
        std::getline(ss, str, ' ');
        vec.push_back(str);
    }
    vec.pop_back();

    std::string correctWord;
    for (auto& iter : vec)
    {
        if (FindDialectWords(dict, iter, &correctWord))
        {
            //SetConsoleTextAttribute(hConsole, 4);
            //std::cout << iter << " ";
            temp.append(correctWord);
            temp.append(" ");
        }
        else
        {
```

```
            //SetConsoleTextAttribute(hConsole, 7);
            //std::cout << iter << " ";
            temp.append(iter);
            temp.append(" ");
        }
    }
    //std::cout << std::endl;
    *sent = temp;
    //std::cout << *sent << std::endl;
    return;
}

void load() {
    std::ifstream dial("dial.txt");
    std::ifstream lit("lit.txt");
    if(!dial.is_open()){
        std::cout << "cannot open file" << std::endl;
    }

    std::ofstream dict("dict.txt");
    std::string temp, d;
    while (dial)
    {
        std::getline(dial, temp);
        //std::cout << temp << std::endl;
        d.append(temp);
        std::getline(lit, temp);
        //std::cout << temp << std::endl;
        d.append("|");
        d.append(temp);
        //std::cout << d << std::endl;
        dict << d << std::endl;
        d.clear();
    }
    dial.close();
    lit.close();
    dict.close();
}

#endif // HEADER_H
```

Лістинг файлу «mainwindow.h»:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QtWidgets/qmainwindow.h>
#include <QMainWindow>
#include <QListWidget>

#include "ui_mainwindow.h"

#include <QTextToSpeech>


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);

    ~MainWindow();

private slots:
    void speak();

    void on_PB_Analize_clicked();

    void on_LW_WordList_itemClicked(QListWidgetItem *item);

    void on_CB_Transcription_stateChanged(int arg1);

    void on_PB_Speech_released();

private:
    Ui::MainWindow *ui;
    QTextToSpeech *speech;
    QVector<QVoice> voices;
};
#endif // MAINWINDOW_H
```

Лістинг файлу «main.cpp»:

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Лістинг файлу «mainwindow.cpp»:

```
#include "mainwindow.h"
```

```
#include "Header.h"

#include <iostream>
#include <QLoggingCategory>
#include <QTableView>
#include <QTableWidget>

std::map<std::string, std::string> dictionary;

MainWindow::MainWindow(QWidget *parent)
    :   QMainWindow(parent),
        ui(new Ui::MainWindow),
        speech(nullptr)
{
    ui->setupUi(this);

    QLoggingCategory::setFilterRules(QStringLiteral("qt.speech.tts=true \n
qt.speech.tts.*=true"));

    //ui->engine->addItem("Default", QString("default"));
    const auto engines = QTextToSpeech::availableEngines();
    //for (const QString &engine : engines)
    //{
        //ui->engine->addItem(engine, engine);
    //}
    //ui.engine->setCurrentIndex(0);
    //engineSelected(0);

    ui->statusbar->showMessage("Завантаження словника ...");

    ui->tabWidget->setCurrentIndex(0);
    load();
    std::thread loadThread(LoadDictionary, &dictionary, "dict.txt");
    //LoadDictionary(&dictionary, "dict.txt");

    std::string dictLine;

    if(loadThread.joinable()){
        loadThread.join();
    }

    for(auto& line : dictionary){
        dictLine = line.first;
        dictLine += " - ";
        dictLine += line.second;
        ui->listWidget->addItem(dictLine.c_str());
        dictLine.clear();
    }

    ui->statusbar->showMessage("Завантаження завершено!");
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::speak(){
    speech->say(ui->TE_Transcription->toPlainText());
}

void MainWindow::on_PB_Analize_clicked()
{
    ui->statusbar->showMessage("Аналіз почато ...");
```

```
    ui->LW_WordList->clear();
    QString str;
    std::string sentence;
    str = ui->TE_Transcription->toPlainText();
    sentence = str.toUtf8().constData();
    std::stringstream ss;
    ss << sentence;
    std::string temp;
    while (ss >> temp){
        //std::getline(ss, temp);
        ui->LW_WordList->addItem(temp.c_str());
    }
    CorrectSentence(&sentence, &dictionary, !ui->CB_Transcription->isChecked());
    std::cout << ui->CB_Transcription->isChecked() << std::endl;
    ui->TE_Literature->setPlainText(sentence.c_str());
    ui->statusbar->showMessage("Аналіз завершено!");

    //test

}

void MainWindow::on_LW_WordList_itemClicked(QListWidgetItem *item)
{
    std::string str;
    QString qStr;
    qStr = item->text();
    str = qStr.toUtf8().constData();
    CorrectSentence(&str, &dictionary, !ui->CB_Transcription->isChecked());
    ui->L_LIteraturWord->setText(str.c_str());
}

void MainWindow::on_CB_Transcription_stateChanged(int arg1)
{
    if(ui->CB_Transcription->isChecked()){
        ui->L_Govirka->setText("Літературне речення");
        ui->L_Literature->setText("Діалектне речення");
        ui->L_WordsList->setText("Літературні слова");
    }
    else{
        ui->L_Govirka->setText("Діалектне речення");
        ui->L_Literature->setText("Літературне речення");
        ui->L_WordsList->setText("Діалектні слова");
    }
}

void MainWindow::on_PB_Speech_released()
{
    //system("python VoiceReconScript.py");
    //this->speak();
    ui->LW_WordList->clear();
    ui->TE_Transcription->clear();
    ui->TE_Literature->clear();
    ui->L_LIteraturWord->clear();
    ui->statusbar->showMessage("Очищено!");
}
```

Лістинг файлу «mainwindow.ui»:

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
```

```
<class>MainWindow</class>
<widget class="QMainWindow" name="MainWindow">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>640</width>
   <height>480</height>
  </rect>
 </property>
 <property name="sizePolicy">
  <sizepolicy hsizetype="Fixed" vsizetype="Fixed">
   <horstretch>0</horstretch>
   <verstretch>0</verstretch>
  </sizepolicy>
 </property>
 <property name="minimumSize">
  <size>
   <width>640</width>
   <height>480</height>
  </size>
 </property>
 <property name="maximumSize">
  <size>
   <width>640</width>
   <height>480</height>
  </size>
 </property>
 <property name="windowTitle">
  <string>Експертна система міста Гнівань</string>
 </property>
 <widget class="QWidget" name="centralwidget">
  <widget class="QTabWidget" name="tabWidget">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>831</width>
     <height>601</height>
    </rect>
   </property>
   <property name="sizePolicy">
    <sizepolicy hsizetype="Minimum" vsizetype="Minimum">
     <horstretch>0</horstretch>
     <verstretch>0</verstretch>
    </sizepolicy>
   </property>
   <property name="baseSize">
    <size>
     <width>640</width>
     <height>480</height>
    </size>
   </property>
   <property name="font">
    <font>
     <family>Calibri</family>
     <pointsize>12</pointsize>
     <weight>50</weight>
     <bold>false</bold>
    </font>
   </property>
   <property name="locale">
    <locale language="Ukrainian" country="Ukraine"/>
   </property>
```

```
<property name="currentIndex">
 <number>4</number>
</property>
<widget class="QWidget" name="W_Analizator">
 <attribute name="title">
  <string>Аналізатор</string>
 </attribute>
 <widget class="QTextEdit" name="TE_Transcription">
  <property name="geometry">
   <rect>
    <x>30</x>
    <y>70</y>
    <width>371</width>
    <height>101</height>
   </rect>
  </property>
 </widget>
 <widget class="QTextEdit" name="TE_Literature">
  <property name="geometry">
   <rect>
    <x>30</x>
    <y>210</y>
    <width>371</width>
    <height>101</height>
   </rect>
  </property>
  <property name="readOnly">
   <bool>true</bool>
  </property>
 </widget>
 <widget class="QLabel" name="L_Govirka">
  <property name="geometry">
   <rect>
    <x>30</x>
    <y>50</y>
    <width>371</width>
    <height>16</height>
   </rect>
  </property>
  <property name="font">
   <font>
    <family>Calibri</family>
    <pointsize>11</pointsize>
    <weight>50</weight>
    <bold>false</bold>
    <underline>false</underline>
    <kerning>true</kerning>
   </font>
  </property>
  <property name="text">
   <string>Діалектне речення</string>
  </property>
 </widget>
 <widget class="QLabel" name="L_Literature">
  <property name="geometry">
   <rect>
    <x>30</x>
    <y>190</y>
    <width>371</width>
    <height>16</height>
   </rect>
  </property>
  <property name="text">
   <string>Літературне речення</string>
```

```
 </property>
</widget>
<widget class="QLabel" name="L_WordsList">
 <property name="geometry">
  <rect>
   <x>440</x>
   <y>40</y>
   <width>161</width>
   <height>20</height>
  </rect>
 </property>
 <property name="text">
  <string>Діалектні слова</string>
 </property>
</widget>
<widget class="QListWidget" name="LW_WordList">
 <property name="geometry">
  <rect>
   <x>440</x>
   <y>70</y>
   <width>171</width>
   <height>201</height>
  </rect>
 </property>
</widget>
<widget class="QLabel" name="L_LIteraturWord">
 <property name="geometry">
  <rect>
   <x>440</x>
   <y>280</y>
   <width>171</width>
   <height>16</height>
  </rect>
 </property>
 <property name="text">
  <string> --- </string>
 </property>
</widget>
<widget class="QPushButton" name="PB_Analize">
 <property name="geometry">
  <rect>
   <x>30</x>
   <y>330</y>
   <width>251</width>
   <height>41</height>
  </rect>
 </property>
 <property name="text">
  <string>Аналізувати</string>
 </property>
 <property name="checkable">
  <bool>false</bool>
 </property>
</widget>
<widget class="QCheckBox" name="CB_Transcription">
 <property name="geometry">
  <rect>
   <x>440</x>
   <y>330</y>
   <width>171</width>
   <height>41</height>
  </rect>
 </property>
 <property name="text">
```

```
    <string>Транскрипція</string>
   </property>
  </widget>
  <widget class="QPushButton" name="PB_Speech">
   <property name="geometry">
    <rect>
     <x>290</x>
     <y>330</y>
     <width>111</width>
     <height>41</height>
    </rect>
   </property>
   <property name="text">
    <string>Очистити</string>
   </property>
  </widget>
 </widget>
 <widget class="QWidget" name="W_Morf">
  <attribute name="title">
   <string>Морфологічні риси</string>
  </attribute>
  <widget class="QPlainTextEdit" name="PTE_Morf">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>10</y>
     <width>621</width>
     <height>421</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <family>Calibri</family>
     <pointsize>14</pointsize>
     <weight>50</weight>
     <bold>false</bold>
    </font>
   </property>
   <property name="readOnly">
    <bool>true</bool>
   </property>
   <property name="plainText">
    <string>Морфологічні особливості говірки міста Гнівань
          Текст 1
    </string>
   </property>
  </widget>
 </widget>
 <widget class="QWidget" name="W_Fon">
  <attribute name="title">
   <string>Фонетичні риси</string>
  </attribute>
  <widget class="QPlainTextEdit" name="PTE_Fon">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>10</y>
     <width>611</width>
     <height>421</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <family>Calibri</family>
```

```
     <pointsize>14</pointsize>
     <weight>50</weight>
     <italic>false</italic>
     <bold>false</bold>
    </font>
   </property>
   <property name="readOnly">
    <bool>true</bool>
   </property>
   <property name="plainText">
    <string>Фонетичні особливості говірки Гнівані
            Текст 2
    </string>
   </property>
  </widget>
 </widget>
 <widget class="QWidget" name="W_City">
  <attribute name="title">
   <string>Історія міста</string>
  </attribute>
  <widget class="QPlainTextEdit" name="PTE_City">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>10</y>
     <width>621</width>
     <height>421</height>
    </rect>
   </property>
   <property name="font">
    <font>
     <family>Calibri</family>
     <pointsize>14</pointsize>
     <weight>50</weight>
     <bold>false</bold>
    </font>
   </property>
   <property name="readOnly">
    <bool>true</bool>
   </property>
   <property name="plainText">
    <string>Коротка історична довідка про місто Гнівань
            Текст 3
    </string>
   </property>
   <property name="backgroundVisible">
    <bool>false</bool>
   </property>
  </widget>
 </widget>
 <widget class="QWidget" name="W_Dictionary">
  <property name="enabled">
   <bool>true</bool>
  </property>
  <attribute name="title">
   <string>Словник</string>
  </attribute>
  <widget class="QPushButton" name="pushButton">
   <property name="geometry">
    <rect>
     <x>10</x>
     <y>402</y>
     <width>91</width>
     <height>31</height>
```

```
      </rect>
     </property>
     <property name="text">
      <string>PushButton</string>
     </property>
    </widget>
    <widget class="QListWidget" name="listWidget">
     <property name="geometry">
      <rect>
       <x>10</x>
       <y>20</y>
       <width>621</width>
       <height>411</height>
      </rect>
     </property>
     <property name="font">
      <font>
       <family>Calibri</family>
       <pointsize>14</pointsize>
       <weight>50</weight>
       <bold>false</bold>
      </font>
     </property>
     <property name="layoutDirection">
      <enum>Qt::LeftToRight</enum>
     </property>
    </widget>
   </widget>
  </widget>
 </widget>
 <widget class="QStatusBar" name="statusbar"/>
</widget>
<resources/>
<connections/>
</ui>
```

ВІДГУК

на кваліфікаційну (бакалаврську) роботу Орлова Сергія Сергійовича з теми:

«»

Оцінка якості виконання кожного розділу та кваліфікаційної (бакалаврської) роботи в цілому _____

_____

Оцінка праці здобувача та результатів кваліфікаційної (бакалаврської) роботи _____

_____

Висновок про наявність плагіату _____

Висновок щодо можливості допуску кваліфікаційної (бакалаврської) роботи до захисту _____

_____

Кваліфікаційна (бакалаврська) робота Орлова Сергія Сергійовича «» відповідає/ не відповідає вимогам, які висуваються до кваліфікаційних (бакалаврських) робіт, і може бути рекомендована/ не рекомендована до захисту.

Керівник: _____ О. С. Ветров