Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ГОЛОСОВОГО ПОМІЧНИКА З 2D ВІЗУАЛІЗАЦІЄЮ З ВИКОРИСТАННЯМ PYTHON, C#, BLENDER ТА MYSQL

DEVELOPMENT OF A VOICE ASSISTANT WITH 2D VISUALIZATION USING PYTHON, C#, BLENDER AND MYSQL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-42
Случак Денис Сергійович

Керівник:
Гульчук Юрій Миколайович

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення*»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*20*» *12* 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Случаку Денису Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка голосового помічника з 2D візуалізацією з використанням Python, C#, Blender та MySQL».*
Керівник роботи: *асистент Гульчук Ю. М.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Python, C#, Unity, Visual Studio Code, Blender, MySQL, методичні вказівки до виконання кваліфікаційної роботи бакалавра.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *провести аналіз аналогічних рішень, визначити вимоги до функціональності, спроєктування архетектури програмного продукту, реалізувати візуалізованого персонажа, розробити систему прийому, забезпечити гнучкість налаштувань, провести тестування.*
5. Перелік графічного матеріалу: *12 рисунків.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	Гульчук Ю. М.		
Специфікація вимог до розробленої системи	Гульчук Ю. М.		
Розробка об'єкта проектування	Гульчук Ю. М.		
Нормоконтроль	Вознюк А. В.		
Гарант ОП	Ліщина Н. М.		
Показник запозичень тексту	2,48 %		
Академічна доброчесність	Гульчук Ю. М.		

7. Дата видачі завдання «20» грудня 2024 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Денис СЛУЧАК

Керівник кваліфікаційної роботи _____ Юрій ГУЛЬЧУК

АНОТАЦІЯ

Случак Д. С. Розробка голосового помічника з 2D візуалізацією з використанням Python, C#, Blender та MySQL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі було проведено аналіз предметної області, розглянуто способи для створення голосових помічників та сучасні методи розробки, які зараз мають високу тенденцію. Також було проведено огляд технологій, які б могли допомогти в роботі з синтезом та розпізнаваням голосу.

У другому розділі було розглянуто розробку архітектури програмного продукту, обгрунтовано вибір інструментарію. Також було розглянуто структуру проєкту, базу даних та основні алгоритми.

У третьому розділі було реалізовано функціональний прототип, проведено його тестування всієї системи та була дана оцінка ефективності продукту.

У висновках виражено узагальнений результат роботи, підтверджено досягнення мети розробки та описані думки щодо подальшого розвитку.

Ключові слова: голосовий помічник, розпізнавання мовлення, Python, Unity, 2D-візуалізація, сокет-з’єднання, користувацький інтерфейс, віртуальний асистент, анімація персонажа, налаштування користувача, управління голосом, C#, Google Speech API.

ABSTRACT

Sluchak D. Development of a Voice Assistant with 2D Visualization Using Python, C#, Blender and MySQL. Manuscript. Qualification work of the bachelor's program "Software engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusion, and a list of references.

The first section analyzed the subject area, considered ways to create voice assistants, and modern development methods that are currently in high demand. It also reviewed technologies that could help in working with voice synthesis and recognition.

The second section examined the development of the software product architecture, justified the choice of tools, and examined the project structure, database, and basic algorithms.

In the third section, a functional prototype was implemented, and its entire system was thoroughly tested to assess the product's effectiveness.

The conclusions summarize the generalized result of the work, confirm the achievement of the development goal, and outline thoughts on future development.

Keywords: voice assistant, speech recognition, Python, Unity, 2D visualization, socket connection, user interface, virtual assistant, character animation, user settings, voice control, C#, Google Speech API.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі дедалі більше стає використання голосових помічників у звичайних діях. Зараз все більше набуває популярності візуалізація таких помічників. 2D візуалізація в таких випадках збільшує привабливість, отримані емоції. Тому розробка голосових помічників з 2D візуалізацією є викликом для розробників програмного забезпечення.

Таким чином, метою даної кваліфікаційної роботи є розробка функціонального прототипу голосового помічника з 2D візуалізацією, що реагує на голосові команди та відображає відповіді.

Об’єктом кваліфікаційної роботи є процес взаємодії за допомогою голосу користувача з іншими програмнами.

Предметом кваліфікаційної роботи є програма реалізацію прототипу голосового помічника на мові програмування Python з методами 2D візуалізації у середовищі Unity.

Для досягнення поставленої мети були поставлені наступні завдання:

– провести аналіз аналогічних програмних рішень та визначити їх переваги й обмеження;

– визначити вимоги до функціональності голосового помічника та способу його візуалізаці;

– спроєктувати архітектуру програмного продукту, включаючи вибір мов програмування, інструментів розпізнавання мовлення та візуалізації;

– реалізувати візуалізованого персонажа у середовищі Unity з анімаційною реакцією на сигнали;

– розробити систему прийому голосових команд, синтезу мовлення та взаємодії зі сценаріями;

– забезпечити гнучкість налаштувань: вибір зовнішності персонажа, голосу, вмикання/вимикання GUI;

– провести тестування роботи програмного продукту та оцінити його відповідність поставленим вимогам.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ ГОЛОСОВОГО ПОМІЧНИКА З 2D ВІЗУАЛІЗАЦІЄЮ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У сучасному світі стали важливою складовою цифрових продуктів та сервісів. Їх застосовують у смартфонах, машинах, розумних колонках, системах розумного дому тощо. Серед найбільш відомих прикладів – Google Assistant [1], Apple Siri [2] та Amazon Alexa [3]. Ці помічники здатні використовувати широкий спектр завдань: надавати інформацію, керувати пристроями, створювати нагадування, вести діалоги тощо.

Google Assistant – це один із найпотужніших та найпопулярніших голосових помічників, який активно використовується на мільйонах пристроїв по всьому світу. Його можна зустріти на смартфонах, розумних колонках, телевізорах, автомобілях і навіть побутовій техніці. Його головна перевага – тісна інтеграція з екосистемою Google: пошуковиком, сервісами типу Gmail чи Google Calendar, картами тощо. Це дозволяє не лише отримувати швидкі відповіді на запитання, а й ефективно керувати щоденними справами – наприклад, створювати події в календарі, запускати таймери, читати повідомлення чи вмикати музику. Асистент підтримує спілкування кількома мовами, вміє підлаштовуватися під контекст запитів і навіть підтримує певну форму «живого» діалогу. Завдяки використанню штучного інтелекту й машинного навчання, Google Assistant постійно вдосконалюється, і взаємодія з ним стає дедалі більш природною.

Siri – це перший серед масових голосових асистентів, який з’явився ще у 2011 році й досі є невід’ємною частиною пристроїв Apple. Вона працює на iPhone, iPad, Apple Watch, Mac і навіть у HomePod – розумних колонках компанії. Siri виділяється, насамперед, своєю глибокою інтеграцією з операційними системами Apple: iOS, macOS та іншими. Її часто

використовують для надсилання повідомлень, пошуку інформації, встановлення будильників, керування музикою чи розумною технікою через HomeKit. Сильна сторона Siri – безпека та конфіденційність: частина обробки запитів відбувається безпосередньо на пристрої, без передавання даних у хмару. Проте, порівняно з конкурентами, Siri часто виглядає трохи обмеженою у спілкуванні: її відповіді менш гнучкі, а ведення діалогу – менш «людське». Проте для користувачів екосистеми Apple вона залишається зручною та надійною помічницею у повсякденних справах.

Alexa – це голосовий асистент, розроблений компанією Amazon, який найбільше асоціюється з розумними колонками лінійки Echo. Вона створена передусім для дому. І справді, саме в системах «розумного житла» Alexa розкриває свої найкращі сторони. За її допомогою можна вмикати світло, регулювати температуру, запускати улюблені плейлисти або замовляти товари з Amazon – просто сказавши кілька слів. Alexa підтримує сотні тисяч додаткових функцій – так званих «навичок», які розробляють сторонні автори. Це дозволяє адаптувати асистента під дуже різні сценарії: від медитації перед сном до читання казок дитині. Alexa вирізняється гнучкістю, але, на жаль, найкраще вона працює англійською. Українська підтримка наразі відсутня, що значно обмежує її застосування в нашому середовищі. Та попри це, в країнах, де англійська є основною, Alexa займає одне з провідних місць серед цифрових асистентів – завдяки широким можливостям і відкритій платформі для розробників.

Незважаючи на потужну функціональність, більшість сучасних голосових помічників орієнтовані на аудіо або текстову взаємодію, без візуального представлення співрозмовника. Це обмежує емоційне сприйняття системи та зменшує її привабливість.

Існуючі сервіси, що частково реалізують ідею візуального аватару. Наприклад, Replika AI [4] дозволяє користувачеві спілкуватися з віртуальним персонажем, який має певні емоційні реакції та підтримує голосову комунікацію. Проте цей сервіс фокусується переважно на психологічному

спілкуванні і не є універсальним голосовим помічником. Крім того, такі рішення здебільшоно не підтримують українську мову та не дозволяють гнучко налаштовувати візуальну частину під власні потреби.

У зв’язку з цим виникла потреба створення власного голосового помічника з 2D візуалізацією, який би поєднував в собі переваги голосового інтерфейсу з анімованим зображенням персонажа. Основні особливості розробки:

– інтерактивна анімація персонажа, що відображає стан системи – «слухаю», «думаю», «говорю»;

– підтримка української мови на рівні розпізнавання та синтезу мовлення;

– персоналізація візуального стилю;

– вибір статі, голосу, анімаційної реакції.

Таким чином, запропонований проєкт спрямований на створення сучасного, зручного та емоційно залученого інтерфейсу.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Для досягнення поставленої мети – створення голосового помічника з 2D візуалізацією, що здатний розпізнавати мовлення, генерувати відповіді та візуально реагувати на етапи діалогу – необхідно вирішити наступні завдання:

– провести аналіз аналогічних програмних рішень та визначити їх переваги й обмеження;

– визначити вимоги до функціональності голосового помічника та способу його візуалізаці;

– спроєктувати архітектуру програмного продукту, включаючи вибір мов програмування, інструментів розпізнавання мовлення та візуалізації;

– реалізувати візуалізованого персонажа у середовищі Unity з анімаційною реакцією на сигнали;11

– розробити систему прийому голосових команд, синтезу мовлення та взаємодії зі сценаріями;

– забезпечити гнучкість налаштувань: вибір зовнішності персонажа, голосу, вмикання/вимикання GUI;

– провести тестування роботи програмного продукту та оцінити його відповідність поставленим вимогам.

Висновки до розділу 1

У результаті проведеного аналізу встановлено, що сучасні голосові помічники, попри свою багатофункціональність та широке застосування в повсякденному житті, переважно реалізують взаємодію з користувачем у вигляді звукового або текстового інтерфейсу, що значно обмежує емоційну складову спілкування. Водночас спостерігається зростаючий інтерес до створення візуалізованих асистентів, здатних не лише взаємодіяти голосом, але й відображати свою активність за допомогою анімаційних або графічних елементів. Розглянуті приклади, зокрема такі як Replika AI, демонструють потенціал подібних рішень, однак не є універсальними голосовими помічниками, не підтримують повноцінну персоналізацію і, в більшості випадків, не орієнтовані на україномовного користувача. Це підтверджує доцільність розробки власного програмного продукту з урахуванням особливостей візуальної взаємодії, гнучких налаштувань і підтримки української мови.

Сформульоване завдання на кваліфікаційну роботу передбачає поетапну реалізацію цілісного голосового асистента з 2D-візуалізацією. Запропонований підхід охоплює як технічні аспекти (розпізнавання мовлення, синтез, візуалізація через Unity), так і взаємодію з користувачем через інтуїтивний та персоналізований інтерфейс. У подальших розділах буде детально описано архітектуру, етапи реалізації та результати тестування розробленої системи.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Голосові помічники активно впроваджуються в повсякденне життя користувачів завдяки розвитку технологій штучного інтелекту, обробки голосу та синтезу мовлення. Нажаль зараз більшість сучасних помічників обмежені голосовим або ж текстовим інтерфейсом, без підтримки візуалізації, що знижує ефективність та емоційність від використання таких програм.

Для підвищення ефективності та зручності використання програмою для користувача доцільно інтегрувати до голосового помічника, яка б додала емоційності в повсякденне використання та ефективності в плані розуміння роботи програми.

На основі аналізу аналогів та предметної області були сформовані такі функціональні вимоги:

– розпізнавання голосових команд користувача;

– генерація голосової відповіді (text-to-speech);

– відображення 2D персонажа з базовими анімаціями;

– вибір персонажа;

– наявність простого GUI з налаштуваннями;

– підключення до системи взаємодії.

Нефункціональні вимоги:

– підтримка української мови;

– простота інтерфейсу;

– відсутність залежності від закритих сервісів.

Для реалізації функціональності голосового помічника система поділяється на кілька взаємопов’язаних модулів, кожен з яких виконує визначене завдання:

1) модуль голосової взаємодії відповідає за приймання аудіо від користувача, розпізнавання тексту, передачу результату на обробку та синтез відповіді;

2) модуль обробки команд передбачає: отримання текстової команди, виконування відповідної дії або генерування відповіді;

3) модуль 2D-візуалізації персонажа реагує на статус голосового модуля та анімує відповідно персонажа;

4) GUI-модуль налаштувань дає змогу змінювати візуальні й голосові параметри.

Кожен із цих модулів виконує свої функції, щоб виконати завдання, яке надасть користувач (рис. 2.1).

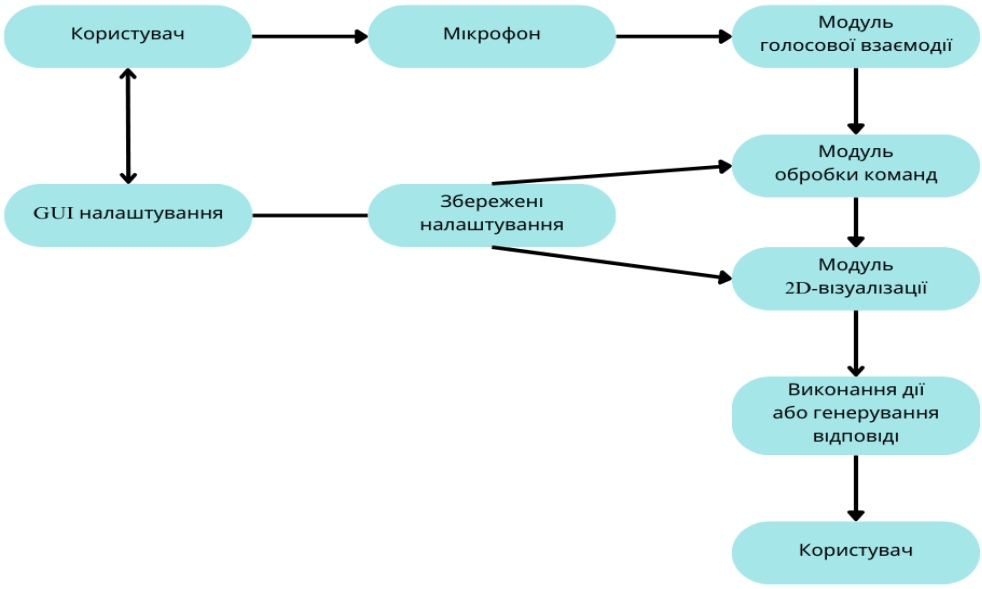

Рисунок 2.1 – Функціональна схема

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

У процесі створення голосового помічника з 2D-візуалізацією було застосовано ряд сучасних інструментальних засобів, мов програмування та

середовищ розробки. Вибір кожного з них їхньою функціональність, популярностю в розробці подібних програм та сумісностю їх роботи між собою. В стоворені програми було використано достатньо інструментарію для розкриття багатьох функцій, які будуть ефективно справлятися з поставленими їм задачами.

Unity – це сучасний кросплатформений ігровий рушій, який широко використовується для створення інтерактивних додатків, ігор, симуляцій та візуалізацій у двомірному просторі. Його розробкою займається компанія Unity Technologies, яка постійно вдосконалює функціональність середовища та розширює його можливості. Завдяки своїй універсальності Unity активно використовується не лише в індустрії комп'ютерних ігор, а й у сфері освіти, архітектурного проєктування, медицини, кіноіндустрії, промисловості та навіть для створення доповненої (AR) і віртуальної реальності (VR).

Однією з ключових переваг цього рушія є його підтримка широкого спектру платформ. Розроблені в Unity проєкти можуть бути скомпільовані та запущені на найпопулярніших операційних системах, зокрема Windows, macOS, Linux, Android, IOS, а також на консолях, таких як Xbox і PlayStation, і навіть у браузерах через WebGL. Це відкриває значні можливості для кросплатформеного розповсюдження програмного продукту та мінімізує витрати часу і ресурсів на адаптацію коду під різні середовища. Ще однією перевагою Unity є потужна та активна спільнота розробників, яка постійно створює нові плагіни, інструменти та навчальні матеріали. Завдяки цьому навіть початківці можуть швидко знайти відповіді на свої питання та отримати підтримку. Unity також має власний офіційний магазин – Asset Store, де можна придбати або безкоштовно завантажити графіку, анімації, моделі, скрипти, шаблони та інші ресурси, що значно прискорює розробку проєктів. Значним плюсом є інтеграція рушія з мовою програмування C#, що робить процес розробки більш зрозумілим, контрольованим та ефективним. Використання C# дозволяє створювати гнучкі сценарії взаємодії об'єктів, організовувати складну логіку гри або системи, а також легко взаємодіяти з зовнішніми сервісами,

бібліотеками чи навіть локальними модулями – як-от, у випадку даного проєкту, зв’язок з Python-ядром через сокети.

Щодо графічних можливостей, Unity надає потужну підтримку як для 2D, так і для 3D графіки. Це дозволяє використовувати платформу як для створення високореалістичних візуалізацій у просторі, так і для простіших, але не менш важливих у сучасних умовах, двовимірних інтерфейсів і персонажів. У межах розробки голосового помічника було обрано саме 2D-графіку, що дозволило створити просту та легку візуалізацію анімованого персонажа, не перевантажуючи системні ресурси.

Узагальнюючи, Unity є надзвичайно гнучким і багатофункціональним інструментом, який забезпечує всі необхідні засоби для розробки інтерактивних додатків різної складності. Його кросплатформеність, інтеграція з C#, активна спільнота та підтримка 2D/3D графіки роблять його ідеальним вибором як для новачків, так і для досвідчених розробників (рис. 2.2).

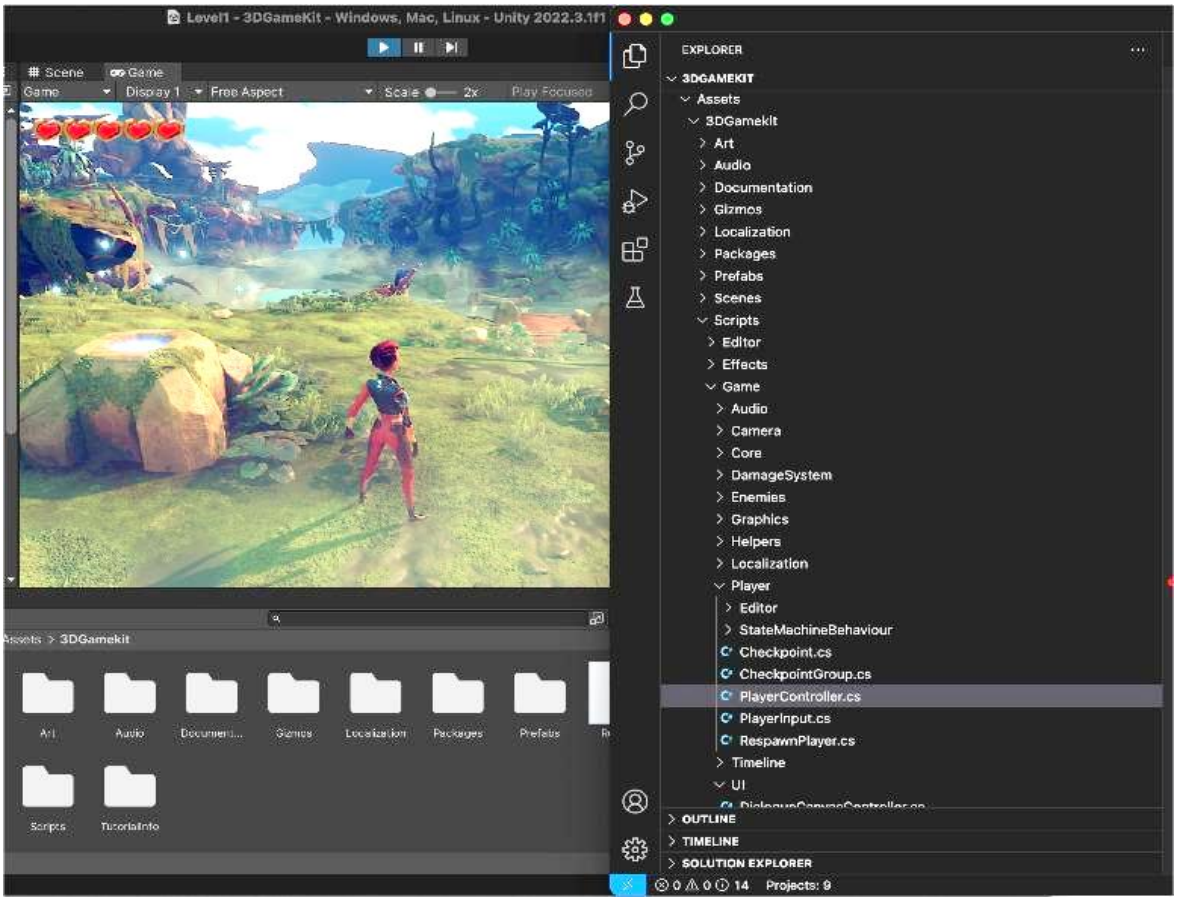

Рисунок 2.2 – Робота з Unity [5]

В проекті Unity виконує роль графічного рушія, який забезпечує візуальну частину, а саме обробку анімації, створення сцени взаємодії з користувачем, а також допомагає взаємодіяти з налаштуваннями.

Blender – це сучасна безкоштовна платформа з відкритим вихідним кодом, яка забезпечує широкий функціонал для 3D-моделювання, анімації, цифрової скульптури, візуалізації, створення 2D-графіки та редагування відео. Програму активно розвиває міжнародна спільнота під керівництвом Blender Foundation. Завдяки підтримці численних форматів файлів та великому набору професійних інструментів, Blender став універсальним рішенням як для художників, так і для технічних фахівців. Інтерфейс програми розташовано на рисунку 2.3.

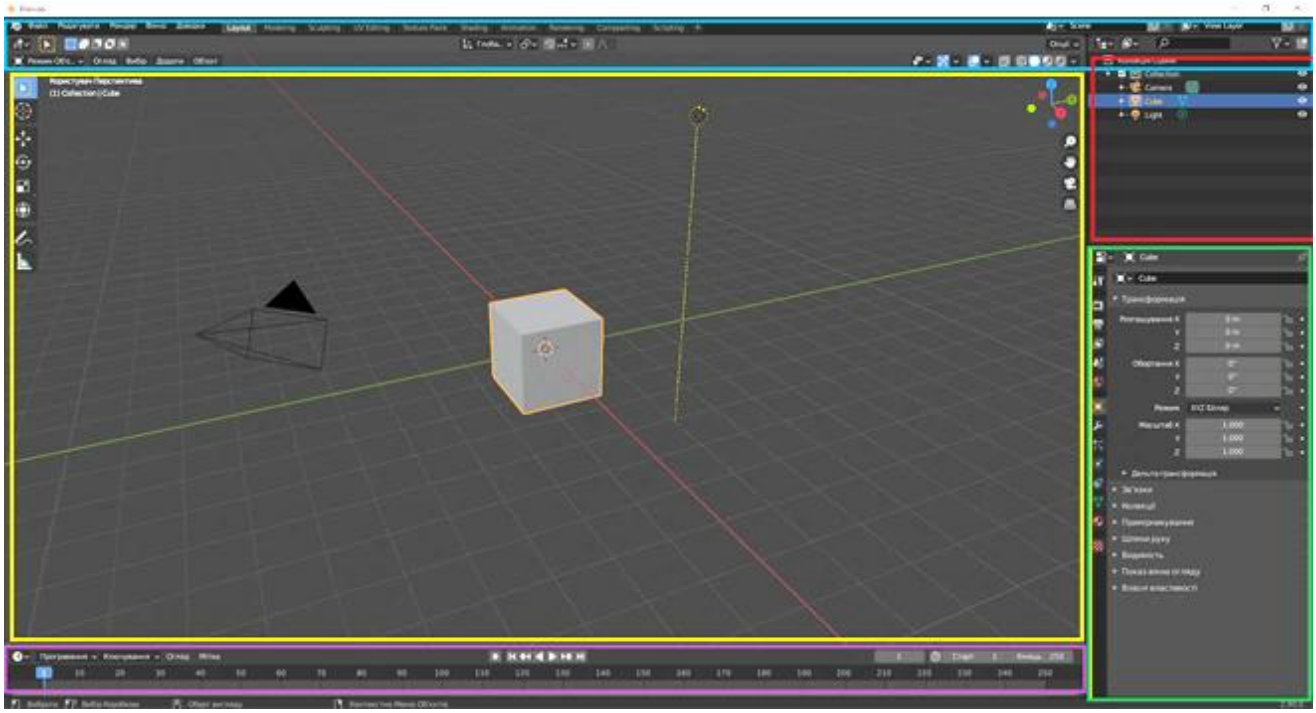

Рисунок 2.3 – Інтерфейс програми Blender [6]

Blender має багато переваг перед його конкурентами. Для цієї програми є велика кількість відео уроків, які допоможуть як в створені, так і в інтеграції в проект. Також ця програма має широкий функціонал, що без сумнівів допоможе зручно розробити всіх необхідні моделі для проекту. Ну а токож

одним з самих найбільших плюсів цієї програми в тому, що вона є повністю безкоштовною.

В проекті Blender застосовувався для створення 2D моделей, які стали частиною візуалізації голосового помічника, та успішно були інтегровані в Unity за допомогою безкоштовної бібліотеки, яка перевела ці моделі в VRM формат, які без всяких проблем підійшли для проекту.

Visual Studio Code – це легкий, багатофункціональний текстовий редактор з відкритим вихідним кодом, розроблений компанією Microsoft. Він активно використовується розробниками програмного забезпечення завдяки своїй гнучкості, продуктивності та широкому набору функціональних можливостей. Однією з ключових переваг Visual Studio Code є кросплатформенність – він доступний для операційних систем Windows, macOS та Linux, що дозволяє працювати на будь-якому середовищі без необхідності адаптації інструменту. Редактор підтримує велику кількість мов програмування, таких як JavaScript, Python, C++, C#, Java, PHP, HTML, CSS та багато інших, що забезпечується завдяки системі розширень. Через вбудований магазин розширень можна легко додавати нові функції, теми, інструменти форматування коду, інтеграції з фреймворками, базами даних і навіть середовищами виконання. Інтерфейс програми зображено на рисунку 2.4.

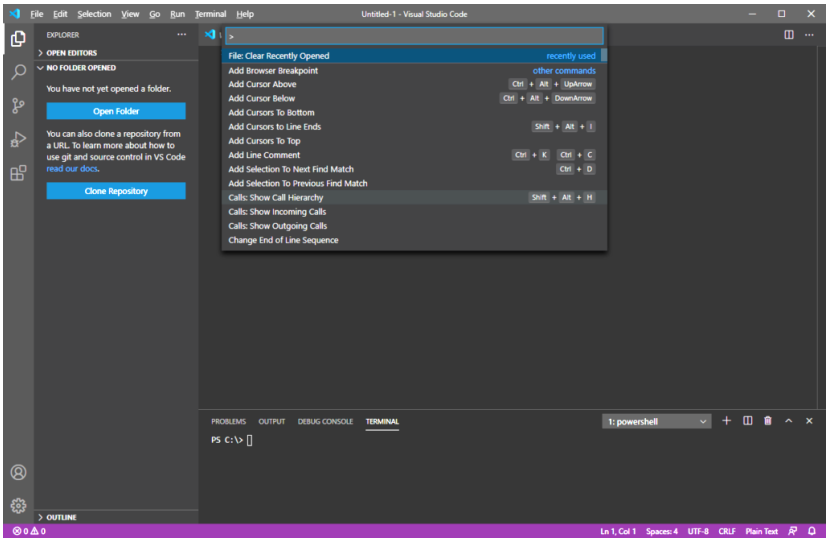

Рисунок 2.4 – Інтерфейс програми VS Code [7]

У процесі реалізації проєкту здійснювалося створення та редагування C#-скриптів для Unity, які відповідали за взаємодію з користувачем, анімацію персонажів, обробку подій, керування інтерфейсом та отримання даних від серверної частини.

Паралельно розроблялися Python-скрипти, які забезпечували роботу серверної логіки, зокрема обробку статусів, а також інтеграцію голосових функцій, таких як генерація й відтворення синтезованої мови, передача даних у реальному часі через сокети або HTTP-запити. Спільна взаємодія клієнтської (Unity) та серверної (Python) частин дозволила створити інтерактивне середовище з живою реакцією персонажів на події та голосову взаємодію.

C# – це об’єктно-орієнтована мова програмування, розроблена компанією Microsoft для платформи .NET. Вона була створена як сучасна, безпечна, типізована та продуктивна мова, яка поєднує в собі зрозумілий синтаксис, високу продуктивність і широкі можливості для створення різноманітних програмних рішень. C# запозичила зручність синтаксису Java, а також гнучкість і потужність C++, водночас уникаючи багатьох типових помилок, притаманних мовам нижчого рівня (рис. 2.5).

Program.cs

```
// 1. Define your data model
public record Issue(int Id, string Component, Severity Level, DateTime OpenedOn);
public enum Severity { Critical, High, Medium, Low }

// 2. Seed some sample issues
var issues = new[]
{
    new Issue(1, "Auth",     Severity.Critical, new(2025, 4, 10)),
    new Issue(2, "Payments", Severity.High,     new(2025, 4, 12)),
    new Issue(3, "Auth",     Severity.Low,      new(2025, 4, 14)),
    new Issue(4, "UI",       Severity.Medium,   new(2025, 4, 15)),
    new Issue(5, "Search",   Severity.Medium,   new(2025, 4, 17)),
    new Issue(6, "Payments", Severity.High,     new(2025, 4, 18))
};

// 3. Build the dashboard in one fluent pipeline
var dashboard =
    issues
        .GroupBy(i => i.Component)                                  // group by component
        .Select(g => new                                            // project each group
        {
            Name      = g.Key,                                      // component name
            OpenCount = g.Count(),                                  // how many are open
            Worst     = g.MinBy(i => i.Level)!.Level,               // highest severity
            AvgAge    = g.Average(i => (DateTime.UtcNow - i.OpenedOn).TotalDays)
        })
        .OrderByDescending(r => r.OpenCount);                       // sort by most open

// 4. Print the result
foreach (var row in dashboard)
    Console.WriteLine(
        $"{row.Name,-8} | {row.OpenCount,2} open | " +
        $"worst: {row.Worst,-8} | avg age: {row.AvgAge:N1} d");
```

Рисунок 2.5 – Зображення коду C# [8]

Середовище Unity зробило C# надзвичайно популярною мовою для розробки 2D та 3D ігор, додатків доповненої та віртуальної реальності, навчальних симуляторів тощо.

Завдяки цьому мова здобула широке поширення як серед професійних розробників, так і серед початківців. Загалом, C# є універсальним і потужним інструментом для створення сучасного програмного забезпечення, який поєднує легкість вивчення з можливостями для реалізації складних і масштабованих рішень.

Мова програмування C# активно використовується у проєкті для реалізації логіки користувацького інтерфейсу, що включає обробку взаємодії користувача з елементами GUI (кнопками, перемикачами, слайдерами тощо), динамічне оновлення екрана, відображення станів і меню [9]. Також за допомогою C# реалізовано керування станами персонажів – тобто контроль за їх анімаціями, реакціями на події, перемиканням між моделями або голосами, взаємодією з середовищем. Всі ці стани моделюються через класи, змінні стану та анімаційні тригери. Крім цього, C# застосовується для інтеграції з серверною частиною проєкту. Це включає встановлення з’єднання з сервером, отримання та обробку даних, оновлення UI згідно з відповідями сервера, а також обмін повідомленнями між клієнтською частиною та бекендом. Таким чином, C# виконує центральну роль у розробці функціональної логіки клієнтської частини проєкту, забезпечуючи взаємодію між користувачем, інтерфейсом, персонажами та сервером.

Python – це високорівнева, інтерпретована мова програмування загального призначення, що вирізняється лаконічним і зрозумілим синтаксисом, що робить її особливо зручною для новачків, а також ефективною для досвідчених розробників (рис. 2.6). Вона підтримує декілька парадигм програмування, включаючи об’єктно-орієнтовану, процедурну та функціональну, що дозволяє адаптувати стиль розробки до конкретних завдань. Одна з ключових переваг Python – це величезна стандартна бібліотека та велика кількість зовнішніх модулів, які значно розширюють її функціональність.

Існують бібліотеки для всього – від обробки зображень і роботи з мережею до штучного інтелекту, машинного навчання, мовного розпізнавання та аналізу даних. Python отримав широке визнання в академічних колах, наукових дослідженнях і промислових рішеннях, завдяки своїй відкритості, активній спільноті та швидкому розвитку

```python
from itertools import repeat
import imageio.v3 as imageio

filenames = ["a.png", "b.png", "c.png"]

images = []

for filename in filenames:
    images.append(imageio.imread(filename))

imageio.imwrite('test.gif', images, duration=500, loop = 0)
```

Рисунок 2.6 – Зображення коду Python [10]

Завдяки своїй гнучкості, читаємості коду та підтримці на всіх сучасних платформах, Python став однією з найпопулярніших мов програмування у світі й активно використовується як у навчанні, так і в комерційних IT-проєктах.

У межах реалізації програмного продукту мова Python використовується для створення голосового модуля, що забезпечує інтелектуальну взаємодію між користувачем та системою в режимі реального часу.

Зокрема, реалізовано функціонал розпізнавання мовлення, який дозволяє інтерпретувати голосові команди користувача. Це досягається за допомогою спеціалізованих бібліотек, які забезпечують трансформацію аудіосигналу в текстову форму. Розпізнаний текст надалі аналізується для виявлення інструкцій або запитів, які потребують обробки. Додатково реалізовано систему синтезу мовлення, що дає змогу перетворювати текстову відповідь системи у звучання, тим самим створюючи ефект живого голосового спілкування. Завдяки цьому додаток не лише «розуміє» користувача, а й «відповідає» йому у зручній

формі. Крім мовної обробки, Python використовується для реалізації логіки опрацювання голосових команд: визначення наміру користувача, запуск відповідних дій, активація анімацій або оновлення інтерфейсу в клієнтській частині програми. Ще одним важливим компонентом є WebSocket-сервер, написаний на Python. Він забезпечує двосторонній зв'язок між серверною частиною (голосовий модуль) та клієнтською частиною, реалізованою в Unity. Це дозволяє в режимі реального часу передавати повідомлення, наприклад, сигнали про поточний статус, або результати розпізнавання мовлення, які відображаються у вигляді дій чи анімацій. Загалом, Python виступає як основний інструмент для реалізації інтелектуальної мовної взаємодії в системі, виконуючи роль посередника між користувачем і візуальною частиною застосунку.

MySQL – це одна з найпопулярніших у світі систем керування реляційними базами даних з відкритим вихідним кодом (рис. 2.7). Вона була спочатку розроблена компанією MySQL AB, а згодом придбана і активно розвивається компанією Oracle Corporation.

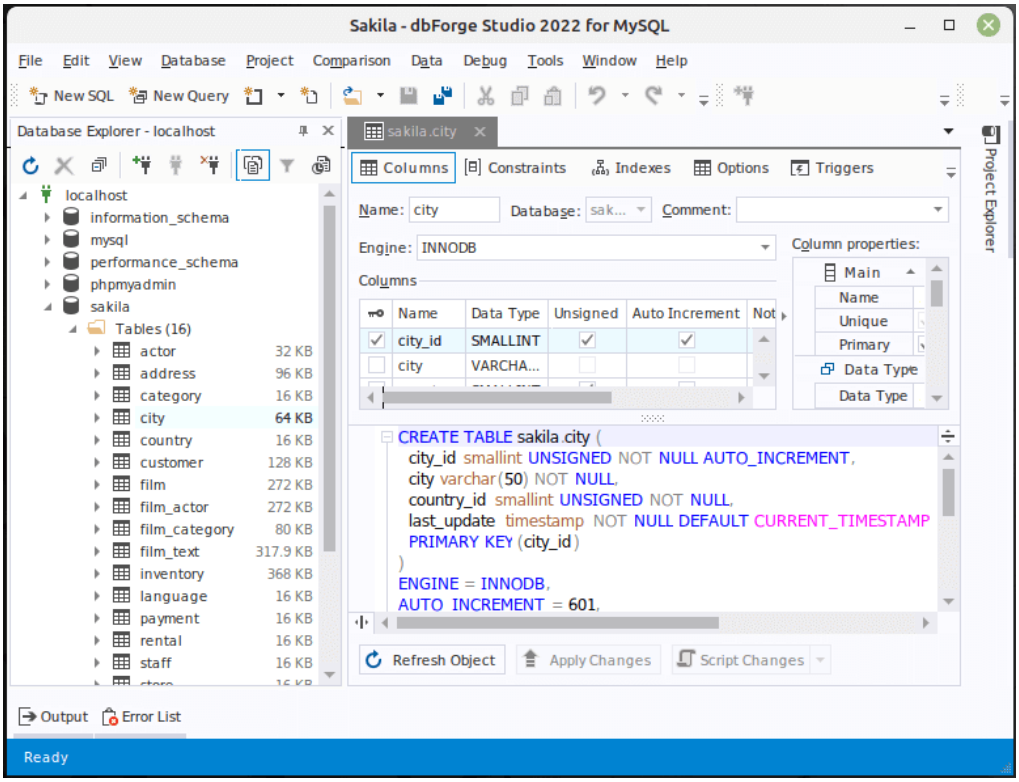

Рисунок 2.7 – Інтерфейс програми MySQL [11]

Завдяки своїй стабільності, швидкодії та широкому набору функцій, MySQL є стандартом де-факто в багатьох веб-проєктах, корпоративних системах та IT-інфраструктурах. Система базується на реляційній моделі, де дані організовані у вигляді таблиць з чітко визначеними зв’язками між ними. Вона повністю підтримує структуровану мову запитів SQL, яка використовується для створення, зміни, пошуку й управління даними у базі.

MySQL підтримується на більшості операційних систем (Windows, Linux, macOS), легко інтегрується з мовами програмування, такими як Python, PHP, Java, C#, а також з популярними фреймворками та CMS. Важливим фактором є також велика міжнародна спільнота користувачів і розробників, яка забезпечує стабільну підтримку, документацію та регулярні оновлення безпеки. Завдяки поєднанню надійності, продуктивності та простоти у використанні, MySQL широко використовується в інформаційних системах, інтернет-магазинах, SaaS-платформах, мобільних додатках і хмарних рішеннях.

У рамках даного проєкту MySQL виконує роль основної системи керування базою даних, яка забезпечує надійне зберігання, структурування та обробку інформації, необхідної для коректної роботи голосового інтерфейсу та користувацьких функцій. Для зручності роботи з MySQL використовується SQL-запитна мова, яка дозволяє гнучко формувати вибірки, фільтрувати, оновлювати чи видаляти записи, а також здійснювати об’єднання даних з кількох таблиць. Таким чином, MySQL забезпечує критично важливу інфраструктуру для надійного збереження та аналізу даних, що лежать в основі взаємодії між користувачем і голосовою системою, забезпечуючи персоналізацію, стабільність та можливість подальшого розвитку продукту.

Висновки до розділу 2

У цьому розділі було сформовано повний перелік вимог до розроблюваної системи голосового помічника з 2D-візуалізацією. Було визначено функціональні та нефункціональні вимоги, що охоплюють всі

ключові аспекти взаємодії користувача із системою: розпізнавання голосу, генерація мови, анімаційна реакція персонажа, налаштування через графічний інтерфейс, а також вимоги до продуктивності та зручності використання.

Описана архітектура системи базується на модульному підході, що включає обчислювальне ядро на Python для обробки голосових команд і середовище Unity для візуалізації та керування персонажем. Передбачено взаємодію між компонентами за допомогою сокет-з’єднання, що забезпечує гнучкість і розширюваність системи.

Також проведено обґрунтування вибору програмного та апаратного забезпечення, зокрема використання мови програмування Python, середовища Unity, редактора Visual Studio Code та Blender для роботи з 3D- і 2D-моделями. Ці інструменти забезпечують ефективну розробку, підтримку міжплатформенності та візуальну якість системи.

Таким чином, визначені вимоги та обрана технологічна база створюють надійне підґрунтя для подальшої розробки, моделювання архітектури та реалізації функціоналу голосового помічника.

РОЗДІЛ 3
РОЗРОБКА ГОЛОСОВОГО ПОМІЧНИКА З 2D ВІЗУАЛІЗАЦІЄЮ

3.1 Практична реалізація об’єкта проєктування

Розробка проєкту розпочалася з планування проєкту, а саме приблизної моделі візуалізації, продумуванням того, що саме повинен робити голосовий помічник. На першому етапі увага була зосереджена на визначенні функціональних вимог до системи, аналізі цільового використання та умов її експлуатації. Було важливо не лише сформувати загальне уявлення про зовнішній вигляд та логіку взаємодії помічника з користувачем, а й забезпечити технічну доцільність реалізації таких функцій у середовищі, що підтримує роботу з голосом, штучним інтелектом та системними ресурсами операційної системи.

Оскільки проєкт мав бути реалізований із залученням мови програмування Python та середовища Unity, ще на початкових етапах розробки постало питання вибору оптимальної архітектурної моделі, яка б не тільки відповідала технічним вимогам, але й забезпечувала максимальну зручність у реалізації задуманого функціоналу. Замість того, щоб сліпо слідувати усталеним стандартам або шаблонним рішенням, було прийнято рішення керуватись індивідуальним підходом до побудови структури проєкту. Це стало результатом глибокого аналізу можливостей кожної з платформ, а також особистої оцінки того, як найкраще поєднати гнучкість програмування на Python з широкими можливостями графічної візуалізації в Unity. Отже, поділ системи на два незалежні, але взаємодіючі модулі – не був вимогою або обмеженням, продиктованим технічними умовами, а саме свідомим вибором, який я зробив з урахуванням особливостей власного підходу до розробки. Основна ідея полягала в тому, щоб забезпечити чітке розмежування обов’язків між складовими системи: обчислювальне ядро мало відповідати за логіку, аналіз вхідних голосових команд, генерацію відповідей, а також взаємодію з різноманітними API, в той час як Unity повинна була слугувати платформою

для створення інтерфейсу користувача та візуального представлення роботи асистента. Такий підхід надавав суттєві переваги. По-перше, він дозволив уникнути перевантаження одного середовища зайвими завданнями, які краще виконуються в іншому. Зокрема, Unity, попри свою потужність у візуалізації, не призначена для глибокої логічної обробки або роботи зі штучним інтелектом на рівні мовної моделі. Навпаки, Python, завдяки великій кількості бібліотек і фреймворків, таких як OpenAI API, SpeechRecognition [12], pyttsx3 та інші, надає всі необхідні інструменти для обробки мовлення та роботи зі штучним інтелектом. Саме тому було вирішено максимально розвантажити Unity від логіки та делегувати її Python-ядру. По-друге, такий розподіл дозволяє у подальшому легко масштабувати проєкт або змінювати окремі його компоненти без необхідності повністю перебудовувати всю систему. Наприклад, при появі нових моделей мовлення або вдосконалених голосових API можна просто оновити або замінити Python-частину, залишивши візуальний інтерфейс без змін. Це ж стосується й зворотної ситуації – за потреби змінити вигляд асистента, його стиль або механіку візуалізації, достатньо оновити Unity-проєкт, не торкаючись при цьому обчислювального ядра. Таким чином, обраний підхід – це не просто зручне технічне рішення, а стратегія розробки, яка поєднує в собі гнучкість, масштабованість і ясність структури. Завдяки цьому вдалося створити не просто функціональний голосовий асистент, а систему, яка легко адаптується до нових завдань, має чітко визначені межі відповідальності кожного модуля і може слугувати базою для майбутніх удосконалень або розширень. Це дозволяє розглядати проєкт не як завершений продукт, а як платформу з відкритими можливостями для подальшого розвитку.

Після завершення етапу концептуального проєктування наступним логічним і критично важливим кроком стало створення основного обчислювального ядра голосового помічника. Цей компонент мав відігравати центральну роль у всій системі, об’єднуючи в собі механізми голосового вводу, обробки природної мови, генерації відповідей, синтезу голосу та взаємодії з

іншими компонентами. Враховуючи потребу у роботі з голосовими командами, гнучкість обробки запитів користувача, а також необхідність інтеграції з сучасними API штучного інтелекту, було обрано мову програмування Python як основну технологічну платформу для реалізації цього модуля. Вибір на користь Python був не випадковим і не базувався лише на його популярності. Ключову роль у прийнятті цього рішення відіграли наявні потужні бібліотеки, орієнтовані на обробку природної мови, синтез і розпізнавання мовлення, мережеву взаємодію, а також інтеграцію з API від сторонніх сервісів. Саме завдяки цьому вдалося досягти високої швидкості розробки, мінімізувати кількість «низькорівневої» роботи, а також сфокусуватися на логіці самого асистента. Для реалізації модуля голосового вводу було обрано бібліотеку speech_recognition, яка забезпечує зручний інтерфейс до декількох сервісів розпізнавання мовлення, включаючи як локальні, так і хмарні рішення. Вона дозволяє не тільки здійснювати запис аудіо з мікрофона користувача, але й розпізнавати сказані слова та перетворювати їх у текстовий запит. У свою чергу, для синтезу голосу та озвучування відповідей системи було реалізовано підключення до API від ElevenLabs (ліст. 3.1) – одного з найсучасніших сервісів, здатного генерувати реалістичне мовлення з урахуванням інтонацій, пауз і навіть емоційної модуляції.

Лістинг 3.1 – API від ElevenLabs

```
ELEVEN_API_KEY =
"sk_ee598ed073cc2778d61a23e777186d07dcc2df285e3b125a"
VOICE_IDS = {
    "female": "nCqaTnIbLdME87OuQaZY",
    "male": "GVRiwBELe0czFUAJj0nX"
}
def speak(text: str):
    url = f"https://api.elevenlabs.io/v1/text-to-speech/{VOICE_ID}"
    headers = {
        "xi-api-key": ELEVEN_API_KEY,
        "Content-Type": "application/json"
    }
```

кінець лістингу 3.1

Такий підхід дозволив досягти ефекту живого спілкування з помічником і підвищити рівень залучення користувача [13]. Архітектурно Python-модуль функціонує у вигляді нескінченного циклу, який перебуває у стані очікування на ініціацію взаємодії. Активація може відбуватись як голосом – через визначене ключове слово, так і вручну – через натискання кнопки. Після активації здійснюється запис аудіо, його обробка та розпізнавання в текстову форму. Отриманий текст передається у вигляді запиту до мовної моделі GPT, що функціонує через OpenAI API (ліст. 3.2).

Лістинг 3.2 – OpenAI API

```
OPENAI_API_KEY = "sk-proj-w9-P0BU-iGi-QE_RxjTxGRZfr7sRoN74541-
brbj0uu0mP_0tMduAHcDOGu1cFFQTExgYvTTSsT3BlbkFJ_PzGjLqovKStoTO87G6H7_
eGklj7mSnLa4WnG-rAvNFrk6BGZbbAZPnqbN7eMFeSUoVtHZ1zEA"

client = openai.OpenAI(api_key=OPENAI_API_KEY)

messages = [
    {"role": "system", "content": "You are an intelligent
assistant."}
]
```

кінець лістингу 3.2

Генерація відповіді відбувається на стороні OpenAI, після чого відповідь повертається до Python-модуля, зберігається в історії діалогу (для можливості підтримки контексту) та паралельно надсилається до модуля синтезу голосу для озвучення. Якщо користувач обрав візуальний режим взаємодії, текстова форма відповіді також може бути продубльована у вигляді виводу на екран. Особливу увагу було приділено забезпеченню ефективного та стабільного зв’язку між Python-ядром і візуальним інтерфейсом, реалізованим у середовищі Unity. Оскільки обидві частини системи працюють незалежно одна від одної в рамках окремих процесів і використовують різні середовища виконання, необхідно було знайти спосіб обміну інформацією, який був би достатньо гнучким, швидким і не створював складних залежностей. Після аналізу можливих підходів – таких як REST API, файловий обмін або використання баз даних –

найоптимальнішим рішенням виявився сокет-зв’язок (ліст. 3.3), що дозволяє реалізувати комунікацію в режимі реального часу.

Лістинг 3.3 – Сокет-зв’язок

```
def settings_server():
HOST = '127.0.0.1'
PORT = 65433
print("Сервер налаштувань запущено...")

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    while True:
    conn, addr = s.accept()
    with conn:
    data = conn.recv(1024).decode('utf-8')
    if data:
    try:
    payload = json.loads(data)
    settings["voice"] = payload.get("voice", "female")
    settings["volume"] = float(payload.get("volume", 1.0))
    print(f" Отримано налаштування від Unity: {settings}")
    except Exception as e:
    print(" Помилка обробки налаштувань:", e)
```

кінець лістингу 3.3

Сокет-зв’язок був реалізований у вигляді простої, але надійної системи обміну повідомленнями між сервером (Python) і клієнтом (Unity). Python виступає в ролі сервера, що відкриває порт для прослуховування вхідних підключень. Зі свого боку, Unity підключається до цього порту як клієнт і очікує на надходження повідомлень (ліст. 3.4).

Лістинг 3.4 – Порт Unity

```
try
{
TcpClient client = new TcpClient("127.0.0.1", 65433);
NetworkStream stream = client.GetStream();

byte[] data = Encoding.UTF8.GetBytes(json);
stream.Write(data, 0, data.Length);

stream.Close();
```

```
client.Close();

Debug.Log("Відправлено налаштування в Python: " + json);
}
catch (SocketException ex)
 {
Debug.LogError(" Не вдалось підключитись до Python: " + ex.Message);
}
```

кінець лістингу 3.4

Завдяки такій архітектурі стало можливим передавати інформацію про поточний стан асистента, що дало змогу створити візуальні індикатори, які в реальному часі змінюють свій вигляд залежно від етапу взаємодії. Наприклад, у момент, коли система починає слухати користувача, до Unity надходить повідомлення «listening», яке активує відповідну анімацію або світловий ефект. У випадку обробки запиту – індикатор змінюється на «processing», а у момент генерації відповіді – на «speaking». Такий підхід дозволив не лише візуально відобразити процес, але й значно покращити взаємодію користувача з системою, створивши ефект живого асистента, який реагує на команди у реальному часі. Крім того, використання сокетів дало змогу уникнути зайвих затримок, забезпечити стабільність передачі даних та розділити логіку на два самостійні процеси, які можуть масштабуватись незалежно один від одного. У майбутньому така архітектура дозволяє розширити функціонал як у напрямку складніших мовних обчислень на Python, так і у бік покращення графічної складової Unity без потреби вносити глибокі зміни до іншої частини системи.

Одним із ключових елементів роботи голосового асистента є модуль розпізнавання мовлення, що відповідає за перетворення усної мови користувача на текстові команди. Для реалізації цієї функціональності було використано бібліотеку speech_recognition (ліст. 3.5) – популярний інструмент Python, який забезпечує взаємодію з декількома сторонніми API для розпізнавання голосу, зокрема Google Speech Recognition.

Лістинг 3.5 – Speech_recognition

```
def listen_and_get_text():
    mic_index = get_best_microphone_index()
    r = sr.Recognizer()
    with sr.Microphone(device_index=mic_index) as source:
        print("Слухаю, говоріть...")
        r.adjust_for_ambient_noise(source)
        audio = r.listen(source)
    try:
        text = r.recognize_google(audio, language="uk-UA")
        print(f" Розпізнано: {text}")
        return text
    except sr.UnknownValueError:
        print(" Не зрозумів аудіо.")
        return None
    except sr.RequestError as e:
        print(f" Помилка сервісу розпізнавання: {e}")
        return None
```

кінець лістингу 3.5

На завершальному етапі реалізації проєкту було створено графічну частину системи, яка є ключовим компонентом у забезпеченні зручної та зрозумілої взаємодії між користувачем і голосовим помічником. Основне призначення цього модуля полягає не лише у відображенні статусів обробки голосових запитів, але й у створенні ефекту живої присутності, що досягається завдяки анімованим персонажам та інтерактивним елементам. У процесі розробки візуальної складової було обрано середовище Unity, що надало широкі можливості для реалізації як статичних, так і динамічних компонентів інтерфейсу. На основній сцені Unity було розміщено двовимірних персонажів, стилізованих під аніме, які реагують на події та зміни станів системи. Крім того, в інтерфейс було інтегровано набір графічних елементів, які дозволяють користувачеві взаємодіяти з програмою та змінювати певні параметри її функціонування. Візуальний стан персонажів змінюється залежно від того, який сигнал надходить із боку обчислювального ядра, створеного на Python. Для цього в Unity реалізовано спеціальний скрипт під назвою CharacterAnimator, який відповідає за прийом повідомлень по сокет-з’єднанню, їхню обробку та запуск відповідних анімацій. Наприклад, якщо система

надсилає повідомлення про початок прослуховування запиту, персонаж починає анімацію, що імітує уважність; у випадку обробки запиту – анімацію роздумів, а у разі відповіді – міміку мовлення. Фрагмент коду, що реалізує цю логіку, подано в лістингу 3.6.

Лістинг 3.6 – CharacterAnimator

```
public class CharacterAnimator : MonoBehaviour
{
public Animator animator;

public void SetState(string state)
    {
        switch (state)
        {
            case "listening":
                animator.Play("ListeningAnimation");
                break;
            case "thinking":
                animator.Play("ThinkingAnimation");
                break;
            case "speaking":
                animator.Play("SpeakingAnimation");
                break;
            default:
                animator.Play("Idle");
                break;
        }
    }
}
```

кінець лістингу 3.6

У наведеному коді використовуються стандартні можливості Unity для керування анімацією через компонент Animator. Метод SetState приймає текстовий параметр, що відповідає поточному стану системи, і в залежності від нього відтворює відповідну анімацію. Назви анімацій мають збігатися з тими, що попередньо налаштовані в Animator Controller, що прив’язаний до персонажа в сцені. Окрім основної сцени, було створено окремий функціональний інтерфейс для налаштування параметрів роботи помічника. У цьому меню користувач має змогу обрати стать віртуального персонажа, змінити голос, яким система відповідає, скоригувати рівень гучності мовлення,

а також активувати або деактивувати графічний інтерфейс, якщо це необхідно. Усі ці параметри зберігаються в конфігураційному середовищі або надсилаються до Python-модуля через сокет-з’єднання, що забезпечує оперативне реагування на зміни налаштувань. Ініціалізація зовнішнього Python-процесу також була реалізована безпосередньо із середовища Unity. Для цього використовується стандартний механізм запуску процесів через клас System.Diagnostics.Process, що дозволяє автоматично запускати скрипт ядра під час завантаження основної сцени. Таким чином, користувачеві не потрібно виконувати додаткові дії, оскільки вся система активується в автоматичному режимі. Особливу увагу було приділено перевірці стабільності та правильності роботи системи. У процесі тестування було змодельовано низку типових сценаріїв використання, серед яких були голосові запити на отримання інформації про погоду, розповідь жарту та відповіді на загальні пізнавальні питання. Система успішно виконувала усі етапи обробки запиту: починаючи від захоплення голосу, його перетворення на текст, аналізу змісту, формування відповіді та завершуючи озвученням результату і синхронною анімацією персонажа відповідно до активного стану. Завдяки використанню багатопоточності на стороні Python, що реалізується за допомогою модуля threading, вдалося уникнути затримок і забезпечити паралельне виконання окремих процесів. Це дозволило зберегти високу продуктивність і плавність роботи навіть на системах із середніми технічними характеристиками, без втрати якості взаємодії.

У результаті проведеної роботи було створено повноцінний візуальний інтерфейс, тісно інтегрований із логічною частиною системи. Поділ функціоналу між Unity та Python забезпечив гнучкість та масштабованість архітектури, що дозволяє надалі легко модернізувати систему, змінювати візуальну складову, додавати нові сценарії, моделі або інші зовнішні компоненти. Система є зручною у використанні, стабільною в роботі та готовою до подальшого розвитку в рамках більш складних інтерактивних рішень.

Візуальна частина голосового асистента реалізована на ігровому рушії Unity, що забезпечує широкі можливості для створення інтерактивного, динамічного та гнучкого графічного інтерфейсу. Основна мета інтерфейсу – забезпечити користувача зручним і привабливим способом взаємодії з асистентом, а також надати доступ до функцій налаштування без потреби звертатися до технічної частини проєкту. Після запуску програми користувач потрапляє на головний екран, де розміщується головне меню. Візуально це меню оформлено в мінімалістичному стилі, який не перевантажує інтерфейс зайвими елементами, зберігаючи чітку ієрархію доступних функцій. Головне меню включає кілька ключових елементів: кнопку запуску голосового помічника, кнопку переходу до налаштувань, а також кнопку виходу з програми. У підменю налаштувань реалізовано можливість гнучкого керування зовнішнім виглядом та поведінкою асистента. Однією з базових функцій є вибір між двома віртуальними персонажами, що візуалізують роботу асистента (ліст. 3.7).

Лістинг 3.7 – Вибір персонажа

```
public class ModelSelectorUI : MonoBehaviour
{
public TMP_Dropdown modelDropdown;

    public void OnStartButtonPressed()
    {
        int selected = modelDropdown.value;
        PlayerPrefs.SetInt("SelectedModel", selected);
        PlayerPrefs.Save();
        SceneManager.LoadScene("SampleScene");
    }
}
```

кінець лістингу 3.7

Користувач має змогу перемикатися між чоловічою та жіночою моделлю, що надає більшу персоналізацію інтерфейсу. Кожна модель представлена у вигляді 2D анімованого персонажа, який візуально реагує на поточний стан системи (ліст. 3.8).

Лістинг 3.8 – Виведення вибраної моделі

```
public class ModelLoader : MonoBehaviour
{
    public GameObject boyModel;
    public GameObject girlModel;

    void Start()
    {
        int selectedModel = PlayerPrefs.GetInt("SelectedModel", 0);

        boyModel.SetActive(selectedModel == 0);
        girlModel.SetActive(selectedModel == 1);

        SetLayerRecursively(boyModel, selectedModel == 0 ? 7 : 0);
        SetLayerRecursively(girlModel, selectedModel == 1 ? 7 : 0);
    }

    void SetLayerRecursively(GameObject obj, int layer)
    {
        obj.layer = layer;
        foreach (Transform child in obj.transform)
        {
            SetLayerRecursively(child.gameObject, layer);
        }
    }
}
```

кінець лістингу 3.8

Ще одним важливим параметром налаштувань є положення асистента на екрані. З метою забезпечення зручності для користувача та уникнення перекриття важливих елементів інших програм, асистент може бути переміщений в один із чотирьох кутів екрану: верхній лівий, верхній правий, нижній лівий або нижній правий (ліст. 3.9).

Лістинг 3.9 – Розміщення асистента

```
int selectedPosition = PlayerPrefs.GetInt("AssistantPosition", 3)
int posX = 0;
int posY = 0;
switch (selectedPosition)
{
    case 0: // Лівий верхній
        posX = 0;
        posY = 0;
        break;
    case 1: // Лівий нижній
```

```
            posX = 0;
            posY = screenHeight - windowHeight;
            break;
        case 2: // Правий верхній
            posX = screenWidth - windowWidth;
            posY = 0;
            break;
        case 3: // Правий нижній
            posX = screenWidth - windowWidth;
            posY = screenHeight - windowHeight;
            break;
}
```

кінець лістингу 3.9

Така гнучкість дозволяє користувачеві адаптувати інтерфейс до власного робочого середовища. Окрім візуальних параметрів, у меню передбачена можливість регулювання гучності асистента. Користувач може змінювати гучність голосових відповідей за допомогою повзунка, що відображається в реальному часі. Це дозволяє забезпечити комфортне звукове сприйняття навіть у змінних умовах оточення – наприклад, у шумному або навпаки, надто тихому середовищі. Функціональні кнопки головного меню – «Старт» і «Вихід» – відповідають за початок роботи з асистентом та завершення сеансу відповідно. Натискання кнопки запуску активує ядро асистента та ініціює візуалізацію обраного персонажа на екрані. Після цього система переходить у режим очікування голосової команди. Кнопка виходу закриває застосунок, зупиняючи всі активні процеси, зокрема й Python-ядро, що працює у фоновому режимі.

Завдяки вказаним елементам управління, система є інтуїтивно зрозумілою та адаптивною до потреб кінцевого користувача. Це досягається за рахунок логічного розташування меню, чітких візуальних індикаторів станів, а також можливості змінювати параметри взаємодії в реальному часі. Користувач має змогу легко орієнтуватися у функціоналі системи навіть без спеціальної підготовки. Плавні переходи між режимами, анімовані реакції персонажа та зручні налаштування створюють комфортне середовище для голосової взаємодії. Візуальна складова, розроблена у Unity, не лише виконує декоративну функцію, але й підсилює технічні процеси, роблячи їх більш

наочними та емоційно залучаючими. Завдяки використанню 2D-анімації [14] з анімешною стилізацією, користувач отримує яскравий та привабливий візуальний образ помічника, який реагує на команди, змінює емоції та положення залежно від контексту розмови. Це значно підвищує рівень занурення в інтерфейс та сприяє позитивному емоційному сприйняттю всієї системи. Окрім цього, підтримка динамічного перемикання між жіночим та чоловічим голосом дозволяє персоналізувати досвід взаємодії, що особливо важливо для створення відчуття індивідуального підходу. Кожен користувач може налаштувати голосового помічника відповідно до власних уподобань, що робить систему більш гнучкою та відкритою до подальших розширень. Усе це забезпечує не лише високий рівень функціональності, але й формує довіру до цифрового помічника як до надійного партнера у повсякденних задачах. Взаємодія з системою стає не просто ефективною, а й приємною, що значно підвищує шанси її успішного застосування у навчальних, побутових чи навіть терапевтичних контекстах.

3.2 Тестування та налагодження системи

Після завершення етапів розробки основних модулів голосового помічника, включаючи обчислювальне ядро на Python, систему голосового вводу-виводу, сокет-зв’язок та 2D-візуалізацію в Unity, наступним надзвичайно важливим кроком стала фаза тестування та налагодження. Цей етап відіграє ключову роль у забезпеченні стабільності, надійності та передбачуваної поведінки системи в умовах реального використання. Тестування дозволяє не лише виявити та усунути помилки, але й оптимізувати функціонування окремих компонентів, забезпечити узгодженість їхньої взаємодії, а також перевірити відповідність програмного продукту визначеним функціональним вимогам.

Процес тестування був організований поетапно, з акцентом як на модульне, так і на інтеграційне тестування. Основна увага зосереджувалась на

чотирьох ключових аспектах системи: коректність розпізнавання мовлення, стабільність генерації відповідей, безперебійність передачі даних між модулями через сокет-зв’язок, а також адекватність візуального відображення статусів асистента в інтерфейсі Unity. Окремо тестувалася логіка взаємодії через меню налаштувань, зокрема можливість зміни статі моделі, гучності озвучення, положення асистента на екрані, а також функціональність кнопок запуску і виходу.

На першому етапі тестування було перевірено модуль розпізнавання мовлення, реалізований за допомогою бібліотеки speech_recognition. Було протестовано роботу з мікрофоном у різних умовах навколишнього шуму. Виявлено, що точність розпізнавання значно знижується в шумному середовищі або при розмитій вимові користувача. Щоб мінімізувати цю проблему, було рекомендовано використовувати гарнітури з якісним шумозаглушенням. Додатково було внесено зміни в параметри обробки аудіо, зокрема додано коротку паузу перед записом, що дозволяє користувачеві приготуватися до вимови команди, а системі – краще «почути» початок фрази.

На наступному етапі увага зосередилась на тестуванні логіки взаємодії з OpenAI API. Було проведено серію запитів із різною тривалістю, складністю та структурою. Результати показали, що у більшості випадків відповідь повертається протягом 2-5 секунд, що є прийнятним для створення ефекту живої взаємодії. Однак інколи виникали затримки, спричинені мережевими умовами або навантаженням на API. Для компенсації таких ситуацій у систему було додано механізм повторної спроби запиту з повідомленням користувача про можливу затримку. Це підвищило надійність діалогу та зменшило кількість критичних збоїв у спілкуванні.

Ще однією критично важливою частиною тестування стала перевірка коректності роботи сокет-зв’язку між Python-ядром та Unity-інтерфейсом. Цей компонент забезпечує передачу статусів та команд, що синхронізують візуальну анімацію з реальним процесом взаємодії. Для цього був реалізований простий TCP-сервер на стороні Python і TCP-клієнт у Unity (C#). Спочатку

спостерігались певні затримки в передачі повідомлень, особливо коли Python-відповідь надсилалась одразу після завершення обробки мовного запиту. З’ясувалося, що проблема полягала у відсутності буферизації даних або їх неповному отриманні в Unity. Рішенням стало використання чіткої структури передачі даних із спеціальними роздільниками та додатковою перевіркою на стороні клієнта. Після цього інформація почала передаватися коректно та своєчасно.

Окремо слід зазначити тестування візуального інтерфейсу в Unity. У проєкті була реалізована система станів персонажа, що відображається через зміну анімацій або поз персонажа. Анімації призначались залежно від отриманого сигналу від Python, і перевірялось, наскільки вчасно вони спрацьовують. У деяких випадках Unity затримувала відтворення анімації через недосконалу синхронізацію з потоком подій. Проблема вирішилась додаванням буферного кадру та оптимізацією порядку обробки анімаційних тригерів. Додатково було протестовано перемикання персонажа (чоловік / жінка) у реальному часі. Для цього було реалізовано завантаження відповідного спрайту зі змінною позою. Тут також виникали помилки, пов’язані з асинхронним завантаженням ресурсів. Після впровадження попереднього кешування спрайтів ці збої зникли.

Особливу увагу було приділено меню налаштувань, де реалізовані опції вибору моделі асистента (чоловіча / жіноча), зміна гучності, а також позиціонування асистента на екрані. Для гучності був реалізований повзунок, який передає значення до аудіо-потоку, контролюючи гучність голосу в режимі реального часу. Спершу спостерігалась проблема, коли зміна гучності не застосовувалась миттєво через затримку оновлення властивостей аудіо-джерела. В результаті було внесено зміну в код, який миттєво викликав оновлення параметрів відтворення одразу після зміни значення слайдера.

Щодо позиціонування моделі асистента, користувач отримав можливість переміщати персонажа між чотирма кутами екрана. Це реалізовано за допомогою зміни координат RectTransform на сцені. Під час тестування

виявилось, що при зміні роздільної здатності екрана положення зміщується некоректно. Проблема вирішилась через прив’язку позиції до відносних координат Canvas (anchor points), а не абсолютних піксельних значень. Таким чином, асистент адекватно змінює позицію навіть при масштабуванні вікна або зміні режиму перегляду (наприклад, з віконного на повноекранний).

Анімаційне меню, яке було реалізоване із застосуванням плавних ефектів появи, зникнення та перемикання вкладок, також не обійшлося без проблем. Спочатку ефекти працювали не завжди стабільно – іноді переходи переривались або зникали не до кінця. Провівши аналіз, було виявлено, що матеріал, застосований до UI-компонентів, не підтримував повністю прозорість при використанні анімаційних кривих. Після оновлення до новішої версії матеріалу з підтримкою transparency-blending проблеми зникли. Це підкреслює важливість правильного вибору шейдерів та матеріалів навіть для 2D-інтерфейсів.

Також був проведений аналіз системних ресурсів, що використовуються програмою під час активної взаємодії. Було виявлено, що Python-ядро, особливо при обробці тривалих запитів або генерації відповіді, може тимчасово споживати значний обсяг оперативної пам’яті. Однак ці сплески короткотривалі та не призводять до аварійного завершення роботи. Було запропоновано обмежити розмір вхідного тексту та встановити таймаут для довготривалих запитів до API, щоб уникнути потенційного зависання системи при повільному з’єднанні з мережею.

Інтеграційне тестування проводилось у формі симуляції повного сценарію використання голосового помічника: користувач активує програму, говорить фразу, отримує відповідь, змінює налаштування, запускає новий діалог тощо. Такі сценарії тестувались багаторазово з різними варіантами дій. У результаті було виявлено кілька логічних помилок – наприклад, у певних випадках при надмірно швидкій зміні налаштувань Unity не встигав оновити всі змінні до нового значення. Це викликало асинхронну поведінку деяких GUI-

елементів. Було додано додаткову перевірку готовності до оновлення, що вирішило проблему.

У ході тестування також перевірялись можливості масштабування та адаптації системи до інших мов чи моделей. Для цього було проведено експеримент зі зміною мови запиту (наприклад, англійська замість української). Результати показали, що модуль розпізнавання та генерації можна легко адаптувати до інших мов, змінивши відповідні параметри функцій розпізнавання та API. Це свідчить про гнучкість системи та можливість подальшого розширення.

Таким чином, завдяки ретельному тестуванню вдалося виявити і виправити низку технічних та логічних недоліків, що суттєво покращили загальну продуктивність, стабільність та зручність користування голосовим помічником. Ретельна увага до деталей на цьому етапі дозволила забезпечити високий рівень якості кінцевого продукту.

3.3 Створення бази даних

Створення бази даних є фундаментальним етапом розробки будь-якої програмної системи, адже саме вона забезпечує надійне збереження, організацію та швидкий доступ до інформації, необхідної для коректної роботи продукту. У рамках розробки голосового помічника з 2D-візуалізацією особлива увага приділяється створенню гнучкої, масштабованої і ефективної бази даних, яка здатна зберігати інформацію про голосові команди користувача та відповідні дії, які система має виконувати у відповідь. Вибір правильного підходу до проєктування та реалізації бази даних є запорукою успішної інтеграції голосового помічника з іншими компонентами системи, а також забезпечення зручності подальшого розвитку та підтримки продукту.

Для реалізації бази даних обрано систему управління базами даних MySQL – одну з найбільш поширених і надійних реляційних СУБД з відкритим кодом. Вибір MySQL обумовлений кількома важливими факторами. По-перше,

MySQL має широку підтримку у спільноті розробників, постійну документацію та активний розвиток, що робить її безпечною і стійкою платформою для зберігання даних. По-друге, ця СУБД демонструє високу продуктивність і стабільність при обробці як невеликих, так і великих обсягів інформації, що особливо актуально для систем реального часу, якою є голосовий помічник. По-третє, MySQL підтримує стандартний мову структурованих запитів SQL, що забезпечує гнучкість і можливість реалізації складних запитів, необхідних для ефективної роботи з даними. І, нарешті, інтеграція MySQL з мовою програмування Python здійснюється легко за допомогою спеціалізованих бібліотек, таких як mysql-connector-python, що є перевагою для розробників, які працюють у середовищі Python. Проєктування структури бази даних розпочалося з визначення основних функціональних вимог, що висуваються до системи. Зокрема, база повинна зберігати голосові команди, що вводить користувач, а також інформацію про дії, які потрібно виконати у відповідь на ці команди. Для цього була спроєктована основна таблиця commands, яка слугує центральним елементом бази. Вона містить такі поля: унікальний ідентифікатор id з автоматичним збільшенням, що забезпечує унікальність кожного запису; текст команди command_text, який користувач вимовляє для активації певної дії; тип дії action_type, що визначає, чи йдеться про відкриття веб-сайту, запуск додатку чи іншу операцію; а також конкретне значення дії action_value, яке може бути URL-адресою, шляхом до програми чи будь-якою іншою інформацією, необхідною для виконання команди. Створення таблиці здійснюється за допомогою стандартного SQL-запиту (ліст. 3.10).

Лістинг 3.10 – SQL-запит

```
CREATE TABLE commands (
    id INT AUTO_INCREMENT PRIMARY KEY,
    command_text VARCHAR(255) NOT NULL,
    action_type VARCHAR(50) NOT NULL,
    action_value TEXT NOT NULL
);
```

кінець лістингу 3.10

Дана структура дозволяє зберігати інформацію у форматі, який є зручним для швидкого пошуку та обробки. Наприклад, за допомогою запитів можна легко знаходити конкретні команди за текстом, фільтрувати їх за типом дії та отримувати необхідні параметри для виконання. Для початкової наповнюваності таблиці розроблено приклади записів, які ілюструють різні типи команд (ліст. 3.11).

Лістинг 3.11 – Приклади команд

```
INSERT INTO commands (command_text, action_type, action_value)
VALUES
('відкрий ютуб', 'url', 'https://www.youtube.com'),
('відкрий оперу', 'app',
'C:\\Users\\strimfars\\AppData\\Roaming\\Microsoft\\Windows\\Start
Menu\\Programs\\Браузер Opera GX.lnk'),
('відкрий ферму', 'app', 'E:\\Farming Simulator
25\\FarmingSimulator2025.exe');
```

кінець лістингу 3.11

Таким чином, база готова зберігати як команди для відкриття веб-сайтів, так і запуску локальних програм, що є ключовим для функціональності голосового помічника.

Інтерактивна робота з базою даних у проєкті забезпечується мовою програмування Python. Застосування Python дозволяє реалізувати логіку отримання голосових команд, їх розпізнавання, пошуку у базі та виконання відповідних дій. Для зв’язку з MySQL використовується бібліотека mysql-connector-python, яка надає простий і зручний інтерфейс для виконання SQL-запитів (ліст. 3.12).

Лістинг 3.12 – MySQL-connector-python

```
db_config = {
    'user': 'strimfars',
    'password': 'qse465F.',
    'host': 'localhost',
    'database': 'assistant_db',
    'raise_on_warnings': True,
    'autocommit': True
```

```
}
def run_command_from_db(command_text):
    try:
        conn = mysql.connector.connect(**db_config)
        cur = conn.cursor()
        query = """
            SELECT action_type, action_value
            FROM commands
            WHERE  LOWER(%s)  LIKE  CONCAT('%',  LOWER(command_text),
'%')
            LIMIT 1
        """
        cur.execute(query, (command_text,))
        result = cur.fetchone()
        cur.close()
        conn.close()
        if result:
            action_type, action_value = result
            print(f" Знайдено: {action_type} → {action_value}")
            run_action(action_type, action_value)
            return True
        else:
            print(" Команду не знайдено.")
            return False
    except mysql.connector.Error as err:
        print(" MySQL-помилка:", err)
        return False
```

кінець лістингу 3.12

Підсумовуючи, створена база даних на основі MySQL забезпечує надійне, гнучке і продуктивне сховище інформації, що є необхідною складовою частиною голосового помічника. Інтеграція з Python дозволяє ефективно керувати командами, отримувати та обробляти голосовий ввід користувача, а також виконувати необхідні дії у системі. Такий підхід гарантує стабільність і зручність у подальшій розробці та підтримці програмного продукту.

3.4 Алгоритм обробки голосових команд

Основна логіка функціонування голосового помічника зосереджена навколо обробки вхідних голосових сигналів. У цьому процесі важливою є не лише точність розпізнавання, а й стабільність взаємодії між обчислювальним

ядром, реалізованим мовою Python, та візуальним середовищем Unity. Завдяки використанню бібліотеки speech_recognition вдається ефективно захоплювати та розпізнавати голосові команди українською мовою. Після запуску програма переходить у стан очікування мовлення: мікрофон активується, і система починає слухати користувача в режимі реального часу. Уловлений аудіосигнал надсилається до хмарного сервісу Google Speech-to-Text [15], де проходить автоматичне розпізнавання мовлення. У відповідь сервіс повертає текст, який потім аналізується локально. Програма виявляє ключові слова або фрази, що дозволяє класифікувати запит і визначити необхідну дію. Наприклад, команда «привіт» може викликати просту вітальну відповідь, а команда «зміни колір» активує зміну візуального параметра у сцені. Отримана інтерпретація команди формує текст відповіді або інструкцію, яка через сокет-з'єднання надсилається до Unity. У візуальному середовищі працює слухач – скрипт SocketReceiver.cs, що постійно очікує на вхідні повідомлення. Після отримання даних Unity запускає відповідні анімації або інші реакції. Наприклад, фраза «говори» може активувати відтворення синтезованої мови, а «поміняй позу» – викликати анімаційне переключення стану персонажа. Архітектура проєкту спеціально побудована з поділом на обчислювальний та візуальний рівень, що забезпечує високу масштабованість і гнучкість. Завдяки такому підходу можна легко розширювати систему, додаючи нові сценарії, команди, або адаптуючи її під інші мови. Важливо, що обмін даними між Python і Unity здійснюється через мережеві сокети, що дає змогу запускати ці компоненти навіть на різних пристроях або операційних системах. Це створює передумови для можливого розгортання системи у хмарному середовищі або на вбудованих платформах. Особливу увагу в процесі реалізації було приділено обробці помилок. Якщо мікрофон відсутній, недоступний або якість сигналу занадто низька, бібліотека speech_recognition може згенерувати винятки UnknownValueError або RequestError. У таких випадках система не завершує роботу, а обробляє помилку: виводиться відповідне повідомлення на екран або через голос («Я вас не розчув», «Будь ласка, повторіть»), після чого програма знову повертається

до стану очікування команди. Такий механізм забезпечує стабільність і безперервність роботи системи без потреби втручання користувача. Ще однією важливою особливістю є синхронізація між статусами Python-модуля та візуальної сцени в Unity. Наприклад, під час активного прослуховування система надсилає статус «слухаю», при розпізнаванні – «думаю», а під час озвучування відповіді – «говорю». Кожен із цих статусів активує відповідну візуальну або анімаційну реакцію персонажа. Це не лише покращує користувацький досвід, а й робить взаємодію природнішою та зрозумілішою. Уся логіка роботи помічника побудована як безперервний цикл обробки голосових команд. Користувач промовляє фразу, система її розпізнає, обробляє, формує відповідь, передає її в Unity, де вона візуалізується або озвучується. Після цього програма знову повертається у режим очікування. Повторення цього циклу дозволяє забезпечити постійну інтерактивність і підтримувати діалог у реальному часі. Такий підхід не лише робить систему зручною у користуванні, а й відкриває перспективи для розширення її функціоналу: від підтримки контекстного аналізу запитів до інтеграції штучного інтелекту для персоналізованих відповідей.

Таким чином, побудована архітектура голосового помічника забезпечує стабільну й масштабовану взаємодію між мовним ядром і візуальним інтерфейсом. Завдяки грамотному розподілу відповідальностей, розширюваності компонентів та обробці типових помилок система є ефективною платформою для подальшого розвитку як у навчальному, так і в комерційному середовищі.

3.5 Інтерфейс користувача та меню налаштувань

Одним із ключових елементів у побудові голосового помічника з 2D-візуалізацією є зручний та інтуїтивно зрозумілий інтерфейс користувача. Відповідний інтерфейс було реалізовано за допомогою інструментів Unity, що дозволяють створювати інтерактивне графічне середовище з можливістю

гнучкого налаштування. Це забезпечує комфортне користування навіть для новачків, які не мають попереднього досвіду у взаємодії з подібними системами.

Головне меню містить кілька основних елементів, серед яких кнопка запуску, кнопка виходу та розділ налаштувань (рис. 3.1).

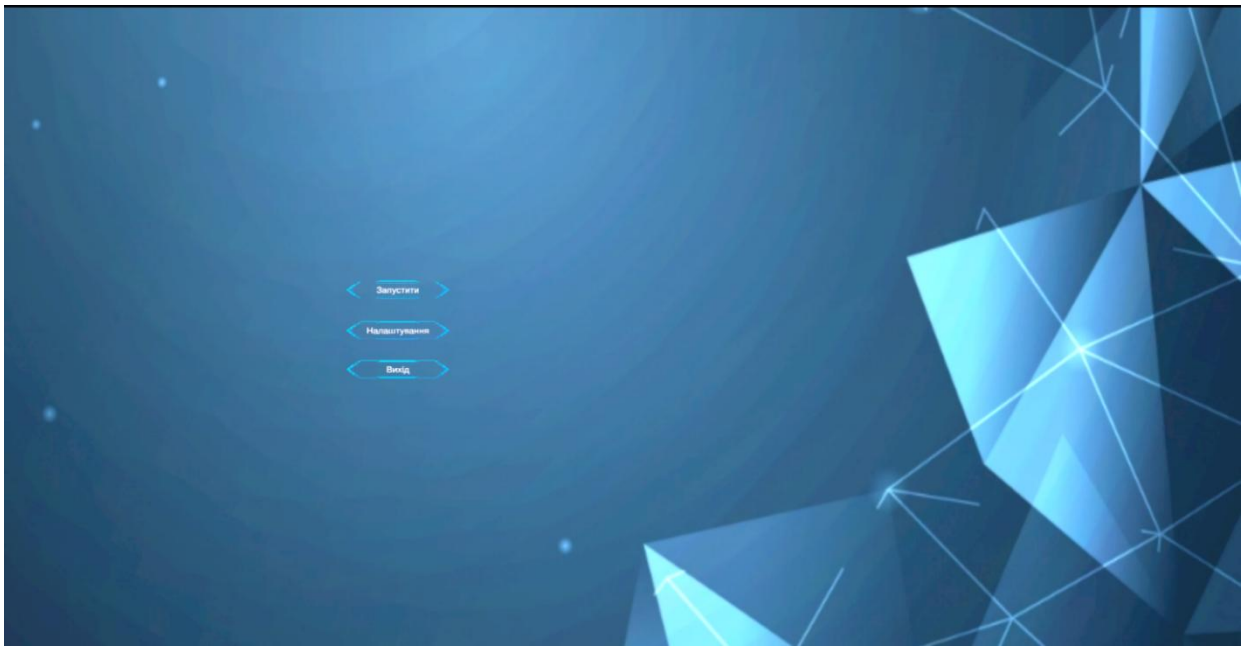

Рисунок 3.1 – Головне меню

Запуск активує режим голосового взаємозв’язку з користувачем, тоді як вихід дозволяє завершити роботу застосунку. У налаштуваннях передбачено можливість змінювати зовнішній вигляд персонажа, обираючи між чоловічою та жіночою моделлю. Це забезпечує варіативність візуального представлення асистента, роблячи взаємодію більш персоналізованою. Також у користувача є змога змінити положення персонажа на екрані – вибравши одне з чотирьох стандартних розташувань у кутах. Такий підхід дозволяє розміщувати помічника там, де він найменше заважає роботі користувача або, навпаки, де його краще видно під час активної взаємодії.

Не менш важливим компонентом є регулювання гучності. Ця функція реалізована через графічний повзунок, що дозволяє плавно змінювати рівень гучності голосу асистента залежно від зовнішніх умов або особистих уподобань

користувача. Меню налаштувань можна побачити на рисунку 3.2. Під час реалізації меню було також запроваджено анімаційні ефекти для покращення візуального сприйняття – зокрема, плавна поява та зникнення елементів, підсвічування при наведенні курсору та інші ефекти взаємодії.

Рисунок 3.2 – Меню налаштувань

На початковому етапі тестування анімаційне меню працювало не зовсім так, як було задумано: елементи не завжди з’являлись коректно, іноді виникали затримки або миготіння. Після детального аналізу з’ясувалося, що проблема полягала у некоректному використанні матеріалів – а саме, анімації працювали з шейдерами, які не були оновлені до останньої версії. Після оновлення матеріалу на більш відповідний тип проблема була повністю усунена.

3.6 Архітектура системи та взаємодія модулів

Архітектура голосового помічника побудована на основі моделі клієнт-сервер, де обчислювальна логіка та візуальне представлення функціонують як окремі, але взаємопов’язані частини системи. У такій моделі обробка мовлення

та формування відповіді зосереджені на стороні сервера, роль якого виконує Python-скрипт. Натомість клієнтська частина, реалізована у Unity, відповідає за виведення інформації, графічне представлення та взаємодію з користувачем. Такий підхід дозволяє забезпечити чітке розділення відповідальностей, покращує підтримуваність і дає змогу окремо розвивати кожен з модулів.

Зв’язок між сервером і клієнтом реалізовано через сокет-з’єднання за допомогою протоколу TCP/IP. На стороні Python при запуску створюється сокет-сервер, який очікує на підключення. Unity виступає в ролі клієнта, що ініціює з’єднання з сервером. Після встановлення каналу зв’язку починається безперервний обмін повідомленнями. Система побудована таким чином, що після виявлення голосової активності мікрофон записує мовлення, яке далі передається на обробку за допомогою хмарного API Google Speech-to-Text. Результатом цього етапу є текстовий рядок з розпізнаною командою. Отриманий текст проходить етап аналізу, під час якого скрипт на Python виконує порівняння із заздалегідь визначеним переліком ключових слів або конструкцій. На основі виявлених елементів формується текстова відповідь, яка одночасно виконує роль інструкції для візуального інтерфейсу. Наприклад, якщо система розпізнає команду «Привіт», вона може сформувати повідомлення: «Говорю: Привіт» або «Анімація: Привітання». Це повідомлення передається через сокет до клієнтської частини, де C#-скрипти Unity здійснюють його обробку. В залежності від типу повідомлення система може відобразити відповідну репліку, активувати анімацію, змінити зовнішній вигляд персонажа або виконати дію в інтерфейсі. Уся логіка роботи системи базується на подієво-орієнтованому підході: кожна команда, подія або зміна стану викликає відповідну реакцію. Після завершення однієї дії система автоматично повертається до стану очікування нової команди. Це забезпечує циклічну та безперервну взаємодію з користувачем. У разі помилок – наприклад, якщо голос не розпізнано або з’єднання втрачено – система переходить у стан очікування, повідомляючи про помилку через графічний або звуковий канал. Особливу увагу під час розробки було приділено поділу системи на логічні

компоненти, кожен з яких виконує окрему роль. Python-скрипти відповідають за аудіозапис, розпізнавання мовлення, логічну інтерпретацію команд і генерацію текстових відповідей. Unity зосереджена на графічному представленні, реалізації анімацій та GUI, обробці команд, отриманих із сокетів, та взаємодії з користувачем. Комунікаційна частина – це шар, який пов’язує ці два модулі між собою. Через сокети передаються як команди, так і сигнали стану системи. Unity отримує повідомлення про те, що система «слухає», «думає» чи «говорить», і відповідно реагує – наприклад, змінює вираз обличчя персонажа або відтворює аудіофрагмент. Система є розширюваною: до неї можна додати нові типи команд, розширити логіку обробки мовлення або інтегрувати інші сервіси (наприклад, синтез мовлення чи емоційне розпізнавання). Гнучкість такої архітектури дозволяє адаптувати її до різних цілей – освітніх, розважальних або інформаційних.

Таким чином, побудована система забезпечує надійну, гнучку та масштабовану основу для голосової взаємодії з графічним інтерфейсом. Використання архітектури клієнт-сервер у поєднанні з сокетами та чітко організованим розподілом функцій між модулями створює зручну платформу для подальшого розвитку голосових технологій у контексті інтерактивних 2D-додатків. Візуальний рівень доповнює обчислювальний, створюючи цілісну систему, здатну забезпечити природну й ефективну взаємодію між користувачем і цифровим асистентом.

Висновки до розділу 3

Підсумовуючи результати третього розділу, можна стверджувати, що реалізований голосовий помічник є успішним прикладом поєднання сучасних технологій розпізнавання мовлення та інтерактивної графіки. На практиці було побудовано ефективне обчислювальне ядро, здатне взаємодіяти з користувачем у реальному часі, а також створено інтуїтивний інтерфейс з естетично привабливою візуалізацією.

Під час розробки було враховано потреби майбутніх користувачів – реалізовано меню налаштувань із можливістю кастомізації зовнішнього вигляду асистента, його положення на екрані та рівня гучності. Це зробило систему зручною та гнучкою. Незважаючи на деякі труднощі на етапі налагодження, зокрема з роботою анімаційного меню, всі технічні проблеми були успішно усунені, що підтверджує ефективність обраних рішень і якість тестування.

Отже, розроблена система може слугувати основою для подальших досліджень і вдосконалення – зокрема, розширення набору команд, інтеграції з іншими платформами або додаванням підтримки емоційних реакцій персонажа. Вона відкриває нові можливості для використання голосової взаємодії в різноманітних освітніх, розважальних або допоміжних застосунках.

ВИСНОВКИ

У процесі виконання завдань було закладено концептуальну й технічну основу для створення голосового помічника з 2D візуалізацією. Кожен із кроків, передбачених цим етапом, сприяв поетапному формуванню ідеї, її структуризації, аналізу доцільності використання певних технологій, а також практичній реалізації функціональних компонентів системи. У цьому розділі представлено підсумки виконання кожного з визначених завдань.

Першим кроком став аналіз існуючих програмних рішень, що мають подібну функціональність. Було розглянуто популярні голосові помічники, такі як Google Assistant, Siri, Alexa, а також проєкт ReplikaAI. У результаті аналізу вдалося визначити ключові переваги цих систем: зручність голосового управління, точність розпізнавання мовлення, адаптація до користувача. Водночас були виявлені обмеження – обмежений візуальний зворотний зв’язок, відсутність підтримки української мови, закритість коду. Це обґрунтувало необхідність створення власного рішення з поліпшеними можливостями візуалізації та гнучкістю налаштувань.

На наступному етапі було визначено вимоги до функціональності системи. Сформульовано функціональні та нефункціональні вимоги, які охоплюють підтримку української мови, можливість розпізнавання й синтезу мовлення, наявність персонажа, що реагує на діалог, налаштування голосу та зовнішності, зручний інтерфейс тощо. Завдяки чітко визначеним вимогам стало можливим послідовне проєктування і реалізація окремих компонентів системи.

Під час проєктування архітектури було обрано розподілену модель, де ядро логіки на Python відповідає за розпізнавання мовлення, генерацію відповідей і синтез голосу, а Unity реалізує візуальне представлення. У якості модуля перетворення мовлення в текст використано технологію Speech-to-Text, а для озвучення тексту – сервіс ElevenLabs, який забезпечує якісну нейромережеву озвучку з підтримкою різних голосів. Обмін даними між

модулями реалізовано через сокет-зв’язок, що забезпечує швидку і стабільну комунікацію.

У Unity було створено 2D-персонажа з анімаційною реакцією на сигнали від Python-модуля. Персонаж відображає зміну станів: «Слухає», «Думає», «Говорить», використовуючи відповідні анімації та візуальні ефекти. Це забезпечує ефективний зворотний зв’язок для користувача та підвищує інтерактивність системи. Анімації організовано через систему тригерів, які синхронізуються з логікою мовного модуля.

Також було розроблено систему обробки голосових команд і генерації відповідей. За допомогою Speech-to-text здійснюється точне розпізнавання мовлення, далі обробка тексту може здійснюватися через мовні моделі, а потім сформовану відповідь озвучує ElevenLabs. Цей ланцюг забезпечує динамічну мовну взаємодію з користувачем у напівреальному часі. Визначені сценарії дозволяють адаптувати реакції під конкретні запити.

У рамках наступного завдання було реалізовано гнучкі налаштування системи. У графічному інтерфейсі користувач може змінювати зовнішній вигляд персонажа, перемикати між жіночим і чоловічим голосом, регулювати гучність, а також вмикати/вимикати графічний інтерфейс. Це забезпечує індивідуалізацію взаємодії та зручність використання в різних середовищах.

На завершальному етапі проведено тестування системи. Перевірено коректність розпізнавання мовлення в різних умовах, якість озвучення відповідей через ElevenLabs, швидкість реакції візуального інтерфейсу, стабільність з’єднання між модулями. Система успішно пройшла тестування в умовах низького рівня шуму. Зафіксовано потребу в інтеграції шумоподавлення для кращої роботи в реальному середовищі. Загалом, функціональні модулі продемонстрували працездатність та відповідність початковим вимогам.

Таким чином, усі завдання, були успішно виконані. Результатом стала інтерактивна система голосового помічника з візуалізацією, що поєднує сучасні мовні технології з графічним інтерфейсом.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Google Assistant, your own personal Google. Assistant. URL: https://assistant.google.com (date of access: 28.03.2025).

2. Siri. Apple. URL: https://www.apple.com/siri (date of access: 08.04.2025).

3. Amazon.com. URL: https://www.amazon.com/alexa (date of access: 08.04.2025).

4. Replika. URL: https://replika.ai (date of access: 13.04.2025).

5. Unity image. Home – Microsoft Developer Blogs. URL: https://devblogs.microsoft.com/visualstudio/wp–content/uploads/sites/4/2023/08/word–image–244229–1.png (date of access:10.05.2025).

6. Blender image. URL: https://resources.cdn.miyklas.com.ua/ed8d184c–f2d1–4f0c–9331–df3ca3db6063/Без%20імені–w1316.png (date of access: 10.05.2025).

7. VScode image. URL: https://romul.name/wp–content/uploads/2020/09/visual–studio–code–interface.png (date of access: 11.05.2025).

8. C# image. URL: https://substackcdn.com/image/fetch/w_1456,c_limit,f_webp,q_auto:good,fl_progressive:steep/https://substack–post–media.s3.amazonaws.com/public/images/a42147fd–d123–465d–8b05–4b8736f7bb7a_2751x2288.png (date of access: 12.05.2025).

9. Tutorials – Unity Learn Toolkit. URL: https://learn.unity.com/tutorial/introduction–to–ui–toolkit (date of access: 25.04.2025).

10. Python image. URL: https://miro.medium.com/v2/resize:fit:1100/format:webp/1*xkMRrM0xMQzdd4GFye4J7A.png (date of access: 13.05.2025).

11. MySQL image. URL: https://i0.wp.com/linuxthebest.net/wp–content/uploads/2024/02/dbforge–studio–for–mysql.png?resize=774,591&ssl=1 (date of access: 15.05.2025).

12. SpeechRecognition. PyPI. URL:https://pypi.org/project/Speech–Recognition / (date of access: 27.03.2025).

13. Joska de Langen. Playing and Recording Sound in Python – Real Python. URL: https://realpython.com/playing–and–recording–sound–python/ (date of access: 07.05.2025).

14. Tutorials – Unity Learn 2D. URL: https://learn.unity.com/ tutorial/2d–animation (date of access: 10.04.2025).

15. Cloud Speech–to–Text Documentation. Google Cloud. URL: https://cloud.google.com/speech–to–text/docs (date of access: 15.03.2025).