

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій
Кафедра системного проектування

Допустити до захисту
Завідувач кафедри
_____доц. Шувар Р. Я.
«___»_____2022 р.

Кваліфікаційна робота

Магістр
(освітній ступінь)

Дослідження та розробка ботів для автоматизації ігрових процесів під управлінням людини та штучного інтелекту

Виконав:
студент групи ФЕІм – 22
спеціальності 122 – Комп'ютерні науки
_____Товкач Б. М.

Науковий керівник:
_____доц. Демків Л. С.
«_____»_____2022 р.

Рецензент:
_____проф. Монастирський Л.С.

Львів 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет _____ Електроніки та комп'ютерних технологій
Кафедра _____ Системного проектування
Освітній ступінь _____ Магістр
Галузь знань _____ 12 Інформаційні технології
(шифр і назва)
Спеціальність _____ 122 Комп'ютерні науки
(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри _____ доц. Шувар Р. Я.

“ _____ ” _____ 20__ року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ (МАГІСТЕРСЬКУ) РОБОТУ СТУДЕНТУ

Товкач Богдан Михайлович

(прізвище, ім'я, по батькові)

1. Тема роботи _____ «Дослідження та розробка ботів для автоматизації ігрових процесів під управлінням людини та штучного інтелекту»
керівник роботи _____ Демків Лідія Степанівна, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені Вченою радою факультету від “ 29 ” червня 2022 року № 27/22
2. Строк подання студентом роботи: _____ до 12 грудня
3. Вихідні дані до роботи: Комп'ютеризована система взаємодій, програмний інтерфейс, графічний інтерфейс, віконний додаток, автоматизація процесів, штучний інтелект, технології розпізнавання, оптимізація, система моніторингу, ресурси ПК.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Розробка віконних додатків, графічного інтерфейсу, інтеграції функцій у програму. Розробка складних рішень для системи взаємодій; Розробка єдиної системи, реалізації функцій розпізнавання, моніторингу та взаємодії з іншими програмами. Міжпрограмна взаємодія з операційною системою; Створення власного повноцінного віконного додатка для автоматизації процесу інших програм та їхнього аналізу.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): Макет віконного додатку, ігрове середовище, графіки завантаження ресурсів ПК.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Технічний розділ	доц. Демків Л. С.		

7. Дата видачі завдання 30 червня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної (магістерської) роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі	14.08.2022	Виконано
2	Огляд технологій та підготовка матеріалу, опрацювання літератури	21.08.2022	Виконано
3	Постановка етапів розробки проєкта	15.09.2022	Виконано
4	Розробка макета вікна програми	18.09.2022	Виконано
5	Реалізація функції комп'ютерного зору та розпізнавання мовлення	25.10.2022	Виконано
6	Реалізація функції взаємодії з грою	01.11.2022	Виконано
7	Інтеграція функціональних модулів в одну програму	07.11.2022	Виконано
8	Тестування програми в штучному середовищі	13.11.2022	Виконано
9	Аналіз результатів тестування та корекція програми	15.11.2022	Виконано
10	Збір даних з довготривалого користування, підготовка до аналізу	28.11.2022	Виконано
11	Аналіз отриманих вихідних даних програми, дослідження результатів	07.12.2022	Виконано

Студент Товкач Б. М.
(підпис) (прізвище та ініціали)

Керівник роботи Демків Л. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розроблено віконний додаток з інтеграцією таких функцій, як: обробка зображень, розпізнавання об'єктів, програмна взаємодія з операційною системою та аналіз вихідних даних програми. Створено інтерфейс для постійного моніторингу активності бота та управління програми людиною. Як приклад, використано бота для автоматизації ігрового процесу.

Досліджено зміни показників навантаження системи комп'ютера (в залежності від оптимізації програми), в тому числі часткове перенесення задач з центрального процесора на дискретну відеокарту.

ABSTRACT

A window application has been developed with the integration of such functions as: image processing, object recognition, software interaction with the operating system, and analysis of program output data. An interface has been created for constant monitoring of the bot's activity and human control of the program. As an example, a bot was used to automate the gameplay.

Researched the changes in computer system load indicators (depending on program optimization), including partial transfer of tasks from the central processor to a discrete video card, were studied.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ШІ	Штучний інтелект
КЗ	Комп'ютерний зір
ПК	Персональний комп'ютер
АП	Автоматизація процесу
ТЗ	Технічні засоби
ПЗ	Програмне забезпечення
ОС	Операційна система
CPU	Центральний процесор
GPU	Дискретний графічний акселератор
RAM	Оперативна пам'ять
NET	Інтернет мережа

ЗМІСТ

ВСТУП	5
РОЗДІЛ I. ОБРОБКА ІНФОРМАЦІЇ. АВТОМАТИЗАЦІЯ ПРОЦЕСІВ.	
КОМП'ЮТЕРНА ВЗАЄМОДІЯ	7
1.1. Методи обробки інформації	7
1.2. Задачі автоматизації	10
1.3. Комп'ютерна система взаємодії	13
РОЗДІЛ II. ТЕХНОЛОГІЇ ПРОГРАМНОЇ РОЗРОБКИ ТА ВЗАЄМОДІЇ.	19
2.1. Підхід до розробки архітектури програми	19
2.2. Бібліотека PySimpleGUI для розробки графічних інтерфейсів	21
2.3. Windows API для взаємодії з операційною системою	30
РОЗДІЛ III. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. РЕАЛІЗАЦІЯ	
ПРОЄКТУ	36
3.1. Постановка задачі та архітектура проєкту	36
3.2. Проєктування дизайну віконного додатку	38
3.3. Написання модулів для роботи комп'ютерного зору та розпізнавання	
мовлення	43
3.4. Реалізація програмної взаємодії з грою	48
3.5. Інтеграція роботи модулів у програмі та тестування	53
РОЗДІЛ IV. ТЕСТУВАННЯ РОЗРОБКИ. ДОСЛІДЖЕННЯ ОТРИМАНИХ	
РЕЗУЛЬТАТІВ	60
4.1. Запуск програми в тестовому середовищі	60
4.2. Аналіз результатів тестування та корекція програми	62
4.3. Збір даних із довготривалого користування в “бойовому” режимі	64
4.4. Аналіз результатів дослідження	68

ВИСНОВКИ.....	76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	78
ДОДАТОК А.....	80
ДОДАТОК Б	103
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ	104
1. Аналіз потенційних небезпек	104
2. Заходи щодо забезпечення безпеки.....	105
3. Заходи щодо виробничої санітарії та гігієни праці	110
4. Заходи з пожежної безпеки.....	112
5. Заходи безпеки у надзвичайних ситуаціях.....	113

ВСТУП

Існує наявна потреба людини у технологіях, які б могли замінити, монотонну і важку роботу, аби відвести вільний час на цікаві справи. Останні десять років стрімко розвивається напрям розробки ботів. Бот — це віртуальний робот, який функціонує на основі спеціальної програми, що виконує автоматично, та/або за заданим розкладом будь-які дії через інтерфейси, призначені для людей. Боти в ігрових середовищах це програми, які використовуються гравцями для полегшення ігрового процесу. У мультиплеєрах ці програми є тим, що не подобається одним, але є вигідним для інших, тому що їх власники отримують значну перевагу. Боти є як: і програмами, які вмішуються в ігровий процес програмно, взаємодіючи із файлами гри, графічними, що збирають інформацію отриману з відеокарти для аналізу на основі, якої керують віртуально периферією ПК так і звичайними, що сліднують одному статичному сценарію.

Створення систем людино-машинного інтерфейсу тісно пов'язане з ергономікою, але не тотожно їй. Проектування людино-машинного інтерфейсу, охоплює: створення робочого місця (крісла, столу, або пульта керування), розміщення приладів і органів керування, освітлення робочого місця, а, можливо, і мікроклімат. Людина потребує графічного інтерфейсу для повноцінної взаємодії з ботом, що полегшить аналіз та моніторинг, як гри, так і програми.

Програми, які допомагають, або надають перевагу, можуть застосовуватися в іграх для оптимізації ігрового процесу. Також, розробка засобів моніторингу та аналізу із подальшою реалізацією у системи управління робототехнікою показує результат на прикладі Boston Dynamics. Програмний бот (робот) — це корисний інструмент, який використовують, зокрема, як умілі гравці, так і ні.

У перших двох розділах розглянуто: методика обробки інформації, розробка програмного забезпечення та автоматизація процесів. Наведено

технології та інструменти для реалізації проєкту, формується задача та архітектура рішення. Описано: розробка віконного інтерфейсу, способи взаємодії з операційною системою через API, використання бібліотек для роботи з аудіо та відео потоком, їхня обробка та аналіз.

У наступних розділах розповідається про реалізацію проєкту, його труднощі розробки та результати. Описано етапи розробки та структура проєкту, розробка модулів та подальша інтеграція у програму. Відбувається тестування, аналіз та представлення отриманих результатів, проводиться дослідження.

Ідея: розробка і дослідження оптимізованого, ефективного бота для спрощення взаємодії людини та програми з перспективами його модернізації.

Мета: реалізація віконного додатка з інтеграцією функцій розпізнавання, взаємодії з ОС та системами моніторингу. Оптимізація ключових моментів ігрового процесу та програми для зменшення навантаження на систему ПК.

Завдання: ознайомлення з передовими технологіями чи інструментами для розробки ефективного ПЗ, пошук бібліотек для роботи з графічним інтерфейсом та програмної взаємодії з ОС Windows. Розробка та реалізація бота для взаємодії з грою та людиною, дослідження результатів роботи програми та порівняння з прототипами, описаними у попередній роботі.

РОЗДІЛ I. ОБРОБКА ІНФОРМАЦІЇ. АВТОМАТИЗАЦІЯ ПРОЦЕСІВ. КОМП'ЮТЕРНА ВЗАЄМОДІЯ

1.1. Методи обробки інформації

Універсальність сучасних інформаційних систем впливає з їхньої здатності представляти інформацію в електронному вигляді як цифрові сигнали та автоматично маніпулювати нею на надзвичайно високих швидкостях. Інформація зберігається в двійкових пристроях, які є основними компонентами цифрової техніки. Оскільки ці пристрої існують лише в одному з двох станів, інформація в них представлена у вигляді відсутності або наявності енергії (електричного імпульсу). Два стани двійкових пристроїв зручно позначати двійковими цифрами, або бітами, нулем (0) і одиницею (1).

Контент-аналіз (див. рис. 1.1.1) зображень виконується двома основними методами: обробкою зображень і розпізнаванням образів. Обробка зображень — це набір обчислювальних методів для аналізу, покращення, стиснення та реконструкції зображень. Розпізнавання шаблонів — це процес скорочення інформації: призначення візуальних або логічних шаблонів класам на основі особливостей цих шаблонів та їхніх зв'язків. Етапи розпізнавання образів включають вимірювання об'єкта для ідентифікації відмінних атрибутів, виділення ознак для визначальних атрибутів і віднесення об'єкта до класу на основі цих ознак. І обробка зображень, і розпізнавання образів мають широке застосування в різних сферах, включаючи астрономію, медицину, промислову робототехніку та дистанційне зондування за допомогою супутників.



Рис. 1.1.1. Етапи контент-аналізу.

Безпосередньою метою аналізу вмісту цифрового мовлення є перетворення окремих звукових елементів у їхні буквено-цифрові еквіваленти. Після такого представлення мова може бути піддана тим же технікам аналізу контенту, що й текст природною мовою, тобто індексування та лінгвістичний аналіз. Перетворення мовних елементів на їхні алфавітно-цифрові відповідники є інтригуючою проблемою, оскільки «форма» звуків мовлення втілює широкий спектр багатьох акустичних характеристик і тому, що лінгвістичні елементи мовлення не можна чітко відрізнити один від одного. Техніка, яка використовується в обробці мовлення, полягає в класифікації спектральних зображень звуку та зіставленні отриманих цифрових спектрографів із попередньо збереженими «шаблонами», щоб ідентифікувати буквено-цифровий еквівалент звуку. (Аверс цього методу, цифрово-аналогове перетворення таких шаблонів у звук, є відносно простим підходом до створення синтетичного мовлення).

Обробка мовлення є складною, а також дорогою з точки зору ємності пам'яті та обчислювальних вимог. Сучасні системи розпізнавання мовлення можуть ідентифікувати обмежені словникові запаси та частини виразно вимовленої мови та можуть бути запрограмовані на розпізнавання тональних особливостей окремих мовців. Коли з'являться більш стійкі та надійні методи, а процес стане обчислювальним (як очікується з паралельними комп'ютерами), люди зможуть взаємодіяти з комп'ютерами за допомогою голосових команд і запитів на рутинній основі. У багатьох ситуаціях це може зробити клавіатуру застарілою як пристрій для введення даних.

Цифрова інформація зберігається у вигляді складних шаблонів, які дозволяють звертатися до найменших елементів символічного вираження та працювати з ними, а також із більшими рядками, такими як слова чи речення, а також із зображеннями та звуком.

З точки зору зберігання цифрової інформації, корисно розрізняти «структуровані» дані, такі як інвентаризація об'єктів, які можуть бути представлені короткими рядками символів і цифрами, і «неструктуровані» дані,

такі як текст документів природною мовою, або мальовничі зображення. Основна мета всіх структур зберігання — сприяти обробці елементів даних на основі їхніх зв'язків; таким чином структури змінюються залежно від типу відносин, які вони представляють. Вибір конкретної структури зберігання регулюється релевантністю зв'язків, які вона дозволяє представити вимогам до обробки інформації завдання або системи.

В інформаційних системах, сховище яких складається з неструктурованих баз даних записів природною мовою, метою є пошук записів (або їх частин) на основі наявності в записах слів або коротких фраз, які складають запит. Оскільки існує індекс як окремий файл, який надає інформацію про розташування слів і фраз у записах бази даних, зв'язки, які представляють інтерес (наприклад, суміжність слів), можна обчислити за допомогою індексу. Отже, сам текст бази даних може зберігатися як простий упорядкований послідовний файл записів. Більшість обчислень використовують індекс, і вони звертаються до текстового файлу лише для того, щоб отримати записи або ті частини, які задовольняють результат обчислень. Послідовна файлова структура залишається популярною з програмним забезпеченням для пошуку документів, призначеним для використання з персональними комп'ютерами та базами даних CD-ROM.

Коли зв'язки між елементами даних потрібно представити як частину записів, щоб зробити бажані операції над цими записами більш ефективними, зазвичай використовуються два типи «з'єднаних» структур: ієрархічна та мережева. В ієрархічній файловій структурі записи розташовані за схемою, що нагадує генеалогічне дерево, із записами, пов'язаними один з одним зверху вниз. У файловій структурі мережі записи впорядковуються в групи, відомі як набори; їх можна з'єднати будь-якою кількістю способів, що забезпечує значну гнучкість. Як в ієрархічній, так і в мережевій структурах зв'язки відображаються за допомогою «вказівників» (тобто ідентифікаторів, таких як адреси чи ключі), які стають частиною записів.

Інший тип структури зберігання бази даних (реляційна структура) стає все більш популярною. Її основною перевагою перед ієрархічною та мережевою

структурами є здатність обробляти непередбачені зв'язки даних без показчиків. Реляційні структури зберігання — це двовимірні таблиці, що складаються з рядків і стовпців, подібно до концептуального бібліотечного каталогу, згаданого вище. Елегантність реляційної моделі полягає в її концептуальній простоті, наявності теоретичних основ (реляційна алгебра) і здатності пов'язаного з нею програмного забезпечення обробляти зв'язки даних без використання показчиків. Реляційна модель спочатку використовувалася для баз даних, що містять високоструктуровану інформацію. У 1990-х роках вона значною мірою замінила ієрархічну та мережеву моделі, а також стала моделлю вибору для великомасштабних програм керування інформацією, як текстових, так і мультимедійних.

Можливість зберігання великих обсягів повного тексту на економному носії (цифровому оптичному диску) відновила інтерес до вивчення структур зберігання, які дозволяють більш потужні методи пошуку та обробки для роботи з когнітивними об'єктами, відмінними від слів, для сприяння більш широкому семантичному аналіз змісту та контексту, а також концептуально організовувати текст у логічні одиниці, а не ті, що диктуються умовностями друку.

Щоб люди могли сприймати та розуміти інформацію, вона має бути представлена у вигляді друку та зображення на папері; як друк і зображення на плівці або на відеотерміналі; як звук через радіо чи телефонний зв'язок; як друк, звук і відео в кінофільмах, телевізійних передачах або на лекціях і конференціях; або під час особистих зустрічей. За винятком живих зустрічей та аудіоінформації, такі дисплеї все більше походять із даних, що зберігаються в цифровому вигляді, а вихідними носіями є відео, друк і звук.

1.2. Задачі автоматизації

Автоматизація — це використання електроніки та пристроїв, керованих комп'ютером, для керування процесами. Метою автоматизації є підвищення ефективності та надійності (не плутати з автоматизованим маркет-мейкером). Однак у більшості випадків автоматизація замінює працю. Насправді

сьогоднішні економісти побоюються, що нові технології згодом значно підвищать рівень безробіття.

Сьогодні на багатьох виробничих підприємствах роботизовані складальні лінії поступово виконують функції, які раніше виконували люди (див. рис. 1.2.1). Термін «виробництво» означає перетворення сировини та компонентів у готову продукцію, як правило, у великих масштабах на заводі.

Автоматизація охоплює багато ключових елементів, систем і робочих функцій практично в усіх галузях. Це особливо поширене у виробництві, транспорті, експлуатації об'єктів і комунальних послуг. Крім того, системи національної оборони стають все більш автоматизованими.

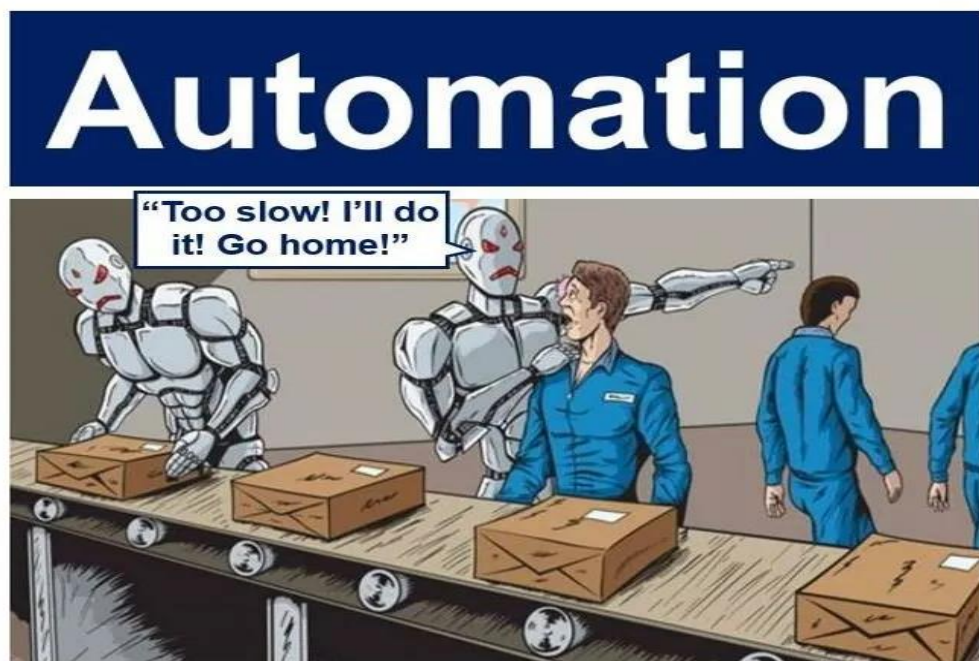


Рис. 1.2.1. Автоматизація, як наслідок розвитку технологій.

Базова автоматизація виконує прості рудиментарні завдання та автоматизує їх. Цей рівень автоматизації стосується оцифрування роботи за допомогою інструментів для оптимізації та централізації рутинних завдань, наприклад використання спільної системи обміну повідомленнями замість того, щоб мати інформацію у відключених силосах. Управління бізнес-процесами (BPM) і роботизована автоматизація процесів (RPA) є типами базової

автоматизації.

Автоматизація процесів забезпечує одноманітність і прозорість бізнес-процесів. Зазвичай це обробляється спеціальним програмним забезпеченням і бізнес-додатками. Використання автоматизації процесів може підвищити продуктивність і ефективність вашого бізнесу. Він також може надати нове розуміння бізнес-завдань і запропонувати рішення. Інтелектуальний аналіз процесів і автоматизація робочого процесу є типами автоматизації процесів.

Автоматизація інтеграції полягає в тому, що машини можуть імітувати людські завдання та повторювати дії, коли люди визначають машинні правила. Одним із прикладів є «цифровий працівник». В останні роки люди визначили цифрових працівників як програмних роботів, навчених працювати з людьми для виконання певних завдань. Вони мають певний набір навичок, і їх можна «найняти» для роботи в команді.

Найскладнішим рівнем автоматизації є автоматизація штучного інтелекту (AI). Додавання ШІ означає, що машини можуть «навчатися» та приймати рішення на основі минулих ситуацій, з якими вони стикалися та аналізували. Наприклад, у сфері обслуговування клієнтів віртуальні помічники можуть зменшити витрати, одночасно розширюючи можливості як клієнтів, так і агентів, створюючи оптимальний досвід обслуговування клієнтів.

Сучасна ера автоматизації робочого процесу почалася в 2005 році з впровадженням BPM. З випуском Siri від Apple у 2011 році тенденція полягала у переході від фізичних роботів до програмного забезпечення для автоматизації:

- Машинне навчання запускає нові процеси, змінюючи маршрут процесів і надання рекомендацій щодо дій;
- Гіперавтоматизація — це злиття машинного навчання, програмного забезпечення та засоби автоматизації для максимального збільшення кількості процесів автоматизації;
- Системи ШІ зможуть автоматизувати налаштування та використання роботів; прогнозна та ймовірнісна обробка для навчання та взаємодії.
- Роботи виконуватимуть кілька завдань, прийматимуть рішення та

працюватимуть автономно, включаючи самодіагностику та обслуговування;

- Пріоритетом буде програмне забезпечення робочого процесу, яке потребує мінімального програмування або взагалі не потребує кодування, зробити автоматизацію процесів доступною для організації.

ШІ та машинне навчання в автоматизації:

- Автоматизація охоплює будь-яку звичайну діяльність критичні для бізнесу. Основна автоматизація запрограмована на виконання а повторюване завдання, щоб людям не довелося цього робити.
- ШІ запрограмовано з логікою та правилами, щоб імітувати прийняття рішень людиною. ШІ може використовуватися для виявлення загроз, таких як зміни в поведінці користувача або збільшення передачі даних.
- Машинне навчання використовує дані та досвід для навчання без додаткового програмування. З кожним новим набором даних він пропонує більш складну та інформовану інформацію.

Стосовно роботи людей і роботів професор Варді сказав:

Наші школи повинні перебудувати свої навчальні програми, щоб учні отримували кращу підготовку з математики, інженерії, технологій і природничих наук. Зростає потреба в працівниках із навичками STEM як розробниками програмного забезпечення, системними аналітиками, біомедичними інженерами та в деяких інших галузях.

1.3. Комп'ютерна система взаємодії

Людино-комп'ютерна взаємодія (HCI) — це дослідження в області проектування та використання комп'ютерних технологій, яке зосереджується на інтерфейсах між людьми (користувачами) і комп'ютерами. Дослідники HCI спостерігають за тим, як люди взаємодіють з комп'ютерами, і розробляють технології, які дозволяють людям взаємодіяти з комп'ютерами новими способами. Пристрій, який забезпечує взаємодію між людиною та комп'ютером, відомий як «людина-комп'ютерний інтерфейс (HCI)».

Як галузь дослідження взаємодія людини та комп'ютера знаходиться на перетині інформатики, поведінкових наук, дизайну, медіа-досліджень та кількох інших галузей дослідження. Цей термін був популяризований Стюартом К. Кардом, Алленом Ньюеллом і Томасом П. Мораном у їхній книзі «Психологія взаємодії людини з комп'ютером» 1983 року. Цей термін має на меті передати, що, на відміну від інших інструментів зі специфічним і обмеженим використанням, комп'ютери мають багато застосувань, які часто передбачають відкритий діалог між користувачем і комп'ютером. Поняття діалогу порівнює взаємодію людини з комп'ютером до взаємодії людини з людиною: аналогія, яка має вирішальне значення для теоретичних міркувань у цій галузі.

Люди взаємодіють з комп'ютерами багатьма способами, і інтерфейс між ними має вирішальне значення для сприяння цій взаємодії. HCI також іноді називають людино-машинною взаємодією (HMI), людино-машинною взаємодією (MMI) або комп'ютерно-людською взаємодією (CHI). Настільні програми, інтернет-браузери, кишенькові комп'ютери та комп'ютерні кіоски використовують поширені сучасні графічні інтерфейси користувача (GUI) [4]. Голосові інтерфейси користувача (VUI) використовуються для систем розпізнавання мовлення та синтезу, а нові мультимодальні та графічні інтерфейси користувача (GUI) дозволяють людям взаємодіяти з втіленими символічними агентами у спосіб, який неможливо досягти за допомогою інших парадигм інтерфейсу. Зростання сфери взаємодії між людиною та комп'ютером привело до підвищення якості взаємодії та призвело до появи багатьох нових напрямків дослідження. Замість того, щоб розробляти звичайні інтерфейси, різні галузі досліджень зосереджуються на концепціях мультимодальності [5] над унімодальністю, інтелектуальних адаптивних інтерфейсів над інтерфейсами на основі команд/дій та активних інтерфейсів над пасивними інтерфейсами [6].

Асоціація обчислювальної техніки (ACM) визначає взаємодію людини з комп'ютером як «дисципліну, яка займається розробкою, оцінкою та впровадженням інтерактивних обчислювальних систем для використання людиною та вивченням основних явищ, які їх оточують» [4]. Важливим аспектом

НСІ є задоволеність користувачів (або задоволеність кінцевого користувача комп'ютером).

Завдяки міждисциплінарному характеру НСІ люди з різним досвідом роблять свій внесок у його успіх. Погано спроектовані людино-машинні інтерфейси можуть призвести до багатьох неочікуваних проблем. Класичним прикладом є аварія на острові Трі-Майл-Айленд, аварія ядерного розплавлення, де розслідування дійшли висновку, що дизайн інтерфейсу людина-машина принаймні частково відповідальний за катастрофу [7][8][9]. Подібним чином аварії в авіації стали результатом рішень виробників використовувати нестандартні пілотажні прилади або компонування квадранта дросельної заслінки: навіть якщо нові конструкції були запропоновані як кращі в базовій взаємодії людини з машиною, пілоти вже вкорінили «стандартну» схему. Таким чином, концептуально хороша ідея мала непередбачені результати.

Інтерфейс людина-комп'ютер можна описати як точку зв'язку між людиною-користувачем і комп'ютером. Потік інформації між людиною та комп'ютером визначається як цикл взаємодії. Цикл взаємодії має кілька аспектів, зокрема:

- Візуальна взаємодія: візуальна взаємодія людини з комп'ютером є, мабуть, найпоширенішою областю дослідження взаємодії людина-комп'ютер (НСІ);
- На основі аудіо: Взаємодія на основі аудіо між комп'ютером і людиною є ще однією важливою областю систем НСІ. Ця область стосується інформації, отриманої різними звуковими сигналами;
- Середовище завдань: умови та цілі, поставлені перед користувачем;
- Машинне середовище: середовище комп'ютера підключено, наприклад, до ноутбука в кімнаті студента коледжу;
- Області інтерфейсу: зони, що не перекриваються, охоплюють процеси, пов'язані з людьми та самими комп'ютерами, тоді як області, що перекриваються, охоплюють лише процеси, пов'язані з їхньою взаємодією;

- Вхідний потік: потік інформації починається в середовищі завдань, коли користувач має певне завдання, яке вимагає використання свого комп'ютера;
- Вихід: потік інформації, що виникає в машинному середовищі;\
- Зворотний зв'язок: циклічне проходження через інтерфейс, який оцінює, модерує та підтверджує процеси, коли вони проходять від людини через інтерфейс до комп'ютера та назад.
- Відповідність: це відповідає дизайну комп'ютера, користувачу та завданню для оптимізації людських ресурсів, необхідних для виконання завдання.

Взаємодія людина-комп'ютер вивчає способи, якими люди використовують або не використовують обчислювальні артефакти, системи та інфраструктури. Значна частина досліджень у цій галузі спрямована на покращення взаємодії людини з комп'ютером шляхом покращення зручності використання комп'ютерних інтерфейсів [10]. Дедалі частіше обговорюється те, як саме слід розуміти зручність використання, як вона пов'язана з іншими соціальними та культурними цінностями, коли вона є бажаною властивістю комп'ютерних інтерфейсів, а коли вона не є бажаною [11][12].

Значна частина досліджень у сфері взаємодії людини та комп'ютера спрямована на:

- Методи проектування нових комп'ютерних інтерфейсів, таким чином оптимізуючи дизайн для бажаних властивостей, таких як можливість навчання, можливість пошуку, ефективність використання;
- Методи реалізації інтерфейсів, наприклад, за допомогою програмних бібліотек;
- Методи оцінювання та порівняння інтерфейсів щодо зручності використання та інших бажаних властивостей;
- Методи вивчення використання людиною комп'ютера та його соціокультурних наслідків у більш широкому плані;

- Методи визначення того, чи є користувач людиною чи комп'ютером;
- Моделі та теорії використання людиною-комп'ютером, а також концептуальні основи для проектування комп'ютерних інтерфейсів, такі як когнітивістські моделі користувача, теорія діяльності або етнометодологічні пояснення використання людиною-комп'ютером [13];
- Перспективи, які критично відображають цінності, що лежать в основі обчислювального дизайну, використання комп'ютера та дослідницької практики HCI [14].

Бачення того, чого дослідники в цій галузі прагнуть досягти, може відрізнитися. Переслідуючи когнітивістську точку зору, дослідники HCI можуть прагнути узгодити комп'ютерні інтерфейси з ментальною моделлю своєї діяльності, яку мають люди. Переслідуючи посткогнітивістську перспективу, дослідники HCI можуть прагнути узгодити комп'ютерні інтерфейси з існуючими соціальними практиками чи існуючими соціокультурними цінностями.

Під час оцінки поточного інтерфейсу користувача або розробки нового інтерфейсу користувача враховуються такі принципи експериментального дизайну (див. рис. 1.3.1):

- На початку зосереджено увагу на користувачеві (користувачах) і завданні (завданнях): встановлюється кількість користувачів, необхідних для виконання завдання (завдань), і визначено, хто має бути відповідним користувачем (хтось, хто ніколи не користувався інтерфейсом і буде не використовувати інтерфейс у майбутньому, швидше за все, є недійсним користувачем). Крім того, визначається завдання(я), які користувачі виконуватимуть, і частота їх виконання;
- Емпіричне вимірювання: інтерфейс тестується реальними користувачами, які щодня стикаються з інтерфейсом. Результати можуть відрізнитися залежно від рівня продуктивності користувача, і типова взаємодія людини з комп'ютером не завжди може бути представлена. Визначаються кількісні характеристики зручності використання, такі як кількість користувачів, які виконують завдання (завдання), час для виконання завдання (завдань) і

кількість помилок, допущених під час виконання завдання (завдань);

- Ітеративний дизайн: після визначення користувачів, завдань і емпіричних вимірювань, які слід включити, виконуються такі кроки ітераційного проектування:
 - Спроектуйте інтерфейс користувача;
 - Тест;
 - Проаналізуйте результати;
 - Повторіть.

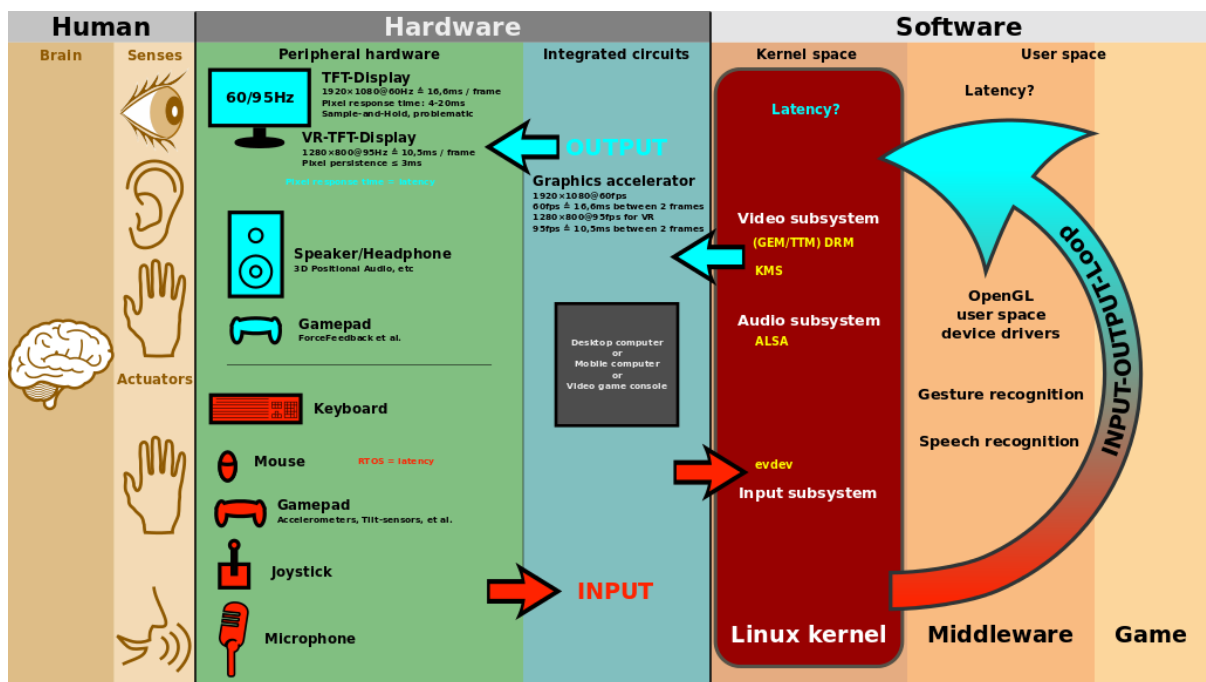


Рис. 1.3.1. Взаємодія людини з обладнанням.

Ітеративний процес проектування повторюється, доки не буде створено розумний, зручний інтерфейс [15].

РОЗДІЛ II. ТЕХНОЛОГІЇ ПРОГРАМНОЇ РОЗРОБКИ ТА ВЗАЄМОДІЇ.

2.1. Підхід до розробки архітектури програми

Взагалі, немає загальноприйнятого терміна «архітектура програмного забезпечення». Проте коли справа стосується практики, то для більшості розробників і так зрозуміло який код є хорошим, а який поганим. Хороша архітектура це насамперед вигідна архітектура, що робить процес розробки та супроводу програми простим та ефективним. Програму з гарною архітектурою легше розширювати та змінювати, а також тестувати, налагоджувати та розуміти. Тобто насправді можна сформулювати список цілком розумних та універсальних критеріїв:

- Ефективність системи:

Насамперед програма, звичайно ж, повинна вирішувати поставлені завдання та добре виконувати свої функції, причому у різних умовах. Сюди можна віднести такі характеристики, як надійність, безпека, продуктивність, здатність справлятися зі збільшенням навантаження (масштабованість) тощо;

- Гнучкість системи:

Будь-який додаток доводиться змінювати з часом — змінюються вимоги, додаються нові. Чим швидше і зручніше можна внести зміни до наявного функціоналу, чим менше проблем і помилок це викличе — тим гнучкіша і конкурентоспроможніша система. Тому в процесі розробки варто оцінювати те, що виходить, щодо того, як це потім, можливо, доведеться змінювати. Зміна одного фрагмента системи має впливати на її інші фрагменти. По можливості, архітектурні рішення не повинні «вирубуватись у камені», і наслідки архітектурних помилок мають бути в розумній мірі обмежені. "Гарна архітектура дозволяє відкладати прийняття ключових рішень" (Боб Мартін) і мінімізує "ціну" помилок;

- Розширюваність системи:

Можливість додавати до системи нові сутності та функції, не порушуючи її основної структури. На початковому етапі в систему має сенс закладати лише

основний та найнеобхідніший функціонал (принцип YAGNI — you ain't gonna need it, «Вам це не знадобиться»). Але при цьому архітектура повинна дозволяти легко нарощувати додатковий функціонал у міру потреби. Причому так, щоб внесення найімовірніших змін вимагало найменших зусиль.

Вимога, щоб архітектура системи мала гнучкість і розширюваність (тобто була здатна до змін та еволюції) є настільки важливою, що вона навіть сформульована у вигляді окремого принципу — «Принципу відкритості/закритості» (Open-Closed Principle — другий із п'яти принципів SOLID) : Програмні сутності (класи, модулі, функції тощо) повинні бути відкритими для розширення, але закритими для модифікації. Іншими словами: Має бути можливість розширити/змінити поведінку системи без зміни/переписування вже наявних частин системи.

Це означає, що додаток слід проектувати так, щоб зміна його поведінки та додавання нової функціональності досягалося шляхом написання нового коду (розширення), і при цьому не доводилося б змінювати вже наявного код. У такому разі поява нових вимог не спричинить модифікацію наявної логіки, а зможе бути реалізована насамперед внаслідок її розширення. Саме цей принцип є основою «плагінної архітектури» (Plugin Architecture). Внаслідок яких технік це може бути досягнуто:

- Масштабованість процесу розробки:

Можливість скоротити термін розробки рахунок додавання до проєкту нових людей. Архітектура повинна дозволяти розпаралелити процес розробки, щоб багато людей могли працювати над програмою одночасно;

- Тестованість:

Код, який легше тестувати, міститиме менше помилок та надійніше працюватиме. Але тести не лише покращують якість коду. Багато розробників приходять до висновку, що вимога «хорошої тестованості» є також спрямовувальною силою, що автоматично веде до гарного дизайну, і одночасно одним з найважливіших критеріїв, що дозволяють оцінити його якість: "Використовуйте принцип «тестованості» класу як «лакмусовий папірець»

хорошого дизайну класу. Навіть, якщо ненаписані жодного рядка тестового коду, відповідь на це питання в 90% випадків допоможе зрозуміти, наскільки все «добре» або «погано» з його дизайном" (ідеальна архітектура).

2.2. Бібліотека PySimpleGUI для розробки графічних інтерфейсів

PySimpleGUI — це пакет Python, який дозволяє програмістам Python усіх рівнів створювати GUI. Достатньо вказати своє вікно GUI за допомогою «макета», який містить віджети (у PySimpleGUI вони називаються «Елементами»). Отриманий макет використовується для створення вікна за допомогою одного з 4 підтримуваних фреймворків для відображення цього вікна та взаємодії з ним. Підтримувані фреймворки включають tkinter, Qt, WxPython або Remi. Термін «обгортка» іноді використовується для таких видів пакунків.

Код PySimpleGUI є простішим і коротшим, ніж написання безпосередньо з використанням основного фреймворку, оскільки PySimpleGUI реалізує більшу частину «шаблонного коду». Крім того, інтерфейси спрощено, щоб вимагати якомога менше коду для отримання бажаного результату. Залежно від використовуваної програми та фреймворку програма PySimpleGUI може потребувати від 1/2 до 1/10 обсягу коду для створення ідентичного вікна, використовуючи безпосередньо один із фреймворків.

Хоча мета полягає в тому, щоб інкапсулювати/приховати конкретні об'єкти та код, які використовуються фреймворком графічного інтерфейсу користувача, який працює поверх, за потреби можна отримати прямий доступ до залежних від фреймворків віджетів і вікон. Якщо параметр або функція ще не представлені або доступні за допомогою API PySimpleGUI, не відгороджені від фреймворку. Можна розширити можливості, не змінюючи безпосередньо сам пакет PySimpleGUI.

Створення простого графічного інтерфейсу користувача (GUI), який працює на кількох платформах, може бути складним. Але це не повинно бути так. Можна використовувати Python і пакет PySimpleGUI для створення привабливих інтерфейсів користувача, які сподобаються користувачам.

PySimpleGUI — це нова бібліотека GUI Python, яка останнім часом викликає великий інтерес.

PySimpleGUI був запущений у 2018 році, тому це відносно новий пакет порівняно з wxPython або PyQt. PySimpleGUI має чотири порти:

- Tkinter
- PyQt
- wxPython
- Remi

PySimpleGUI обгортає частини кожного з цих пакетів і полегшує їх використання. Однак кожен із портів потрібно встановлювати окремо.

PySimpleGUI огортає весь Tkinter, який постачається з Python. PySimpleGUI містить більшу частину PySide2, але лише невелику частину wxPython. Коли встановлюється PySimpleGUI, отримаємо варіант Tkinter за замовчуванням. Щоб дізнатися більше про Tkinter, можна ознайомитися з програмуванням Python GUI за допомогою Tkinter.

Залежно від того, який варіант PySimpleGUI використовуються, програми, які створюються за допомогою PySimpleGUI, можуть не виглядати рідними для своєї платформи. Але нехай це не завадить вам спробувати PySimpleGUI. PySimpleGUI все ще досить потужний і може виконувати більшість завдань, трохи попрацювавши.

Якщо хтось раніше користувався набором інструментів GUI, можливо, стане відомо про термін віджети. Віджет — це загальний термін, який використовується для опису елементів, які складають інтерфейс користувача (UI), таких як кнопки, мітки, вікна тощо. У PySimpleGUI віджети називаються елементами, які іноді в інших місцях пишуться великими літерами як Елементи.

Одним із основних будівельних блоків PySimpleGUI є “Window()”. Щоб створити “Window()”, можна зробити наступне (див. рис. 2.2.1):

```
# hello_world.py

import PySimpleGUI as sg

sg.Window(title="Hello World", layout=[[[]], margins=(100, 50))).read()
```

Рис. 2.2.1. Приклад примітивної віконної програми.

Отже, “Window()” приймає багато різних аргументів — занадто багато, щоб їх тут перелічувати. Однак для цього прикладу можна дати “Window()” назву та макет і встановити поля, тобто розмір вікна інтерфейсу користувача в пікселях. “read()” повертає будь-які події, які запускаються у “Window()”, як рядок, а також словник значень. Коли запускається цей код, можна побачити щось на зразок цього (див. рис. 2.2.2):

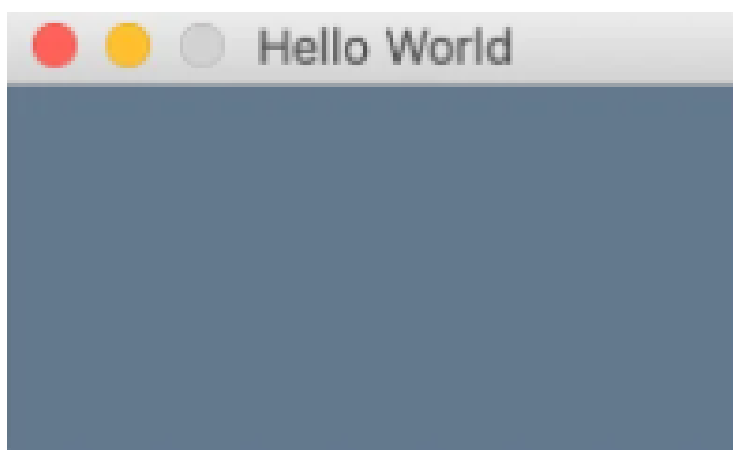


Рис. 2.2.2. Вікно програми.

За допомогою PySimpleGUI можна створювати велику різноманітність різних кросплатформних графічних інтерфейсів. Демонстрації, які входять до пакету, є великими. Можна створювати будь-що: від віджетів робочого столу до повномасштабних інтерфейсів користувача.

Одна з демонстрацій на сторінці GitHub PySimpleGUI — це програма для перегляду зображень. Можливість написати свій власний засіб перегляду зображень на Python — це весело. Можна використовувати цей код для

перегляду власних фотографій або включити його для перегляду фотографій, які завантажуються або читаються з бази даних. Щоб спростити все, можна скористатися вбудованим елементом `Image()` `PySimpleGUI` для перегляду зображень. На жаль, елемент `Image()` може відображати лише формати PNG і GIF у звичайній версії `PySimpleGUI`. Якщо є потреба мати можливість відкривати файли зображень інших типів, можна завантажити `Pillow`, який підтримує формати TIFF, JPG і BMP. Для додаткової інформації, варто переглянути папку демонстрації `PySimpleGUI` на `GitHub`, щоб отримати приклад, який показує, як це зробити.

З іншого боку, якщо встановити порт `PySimpleGUIQt`, стає зрозуміло, що `Qt` підтримує більше форматів зображень із коробки, ніж `Tkinter`. Ось макет того, як має виглядати переглядач зображень у кінці (див. рис. 2.2.3):

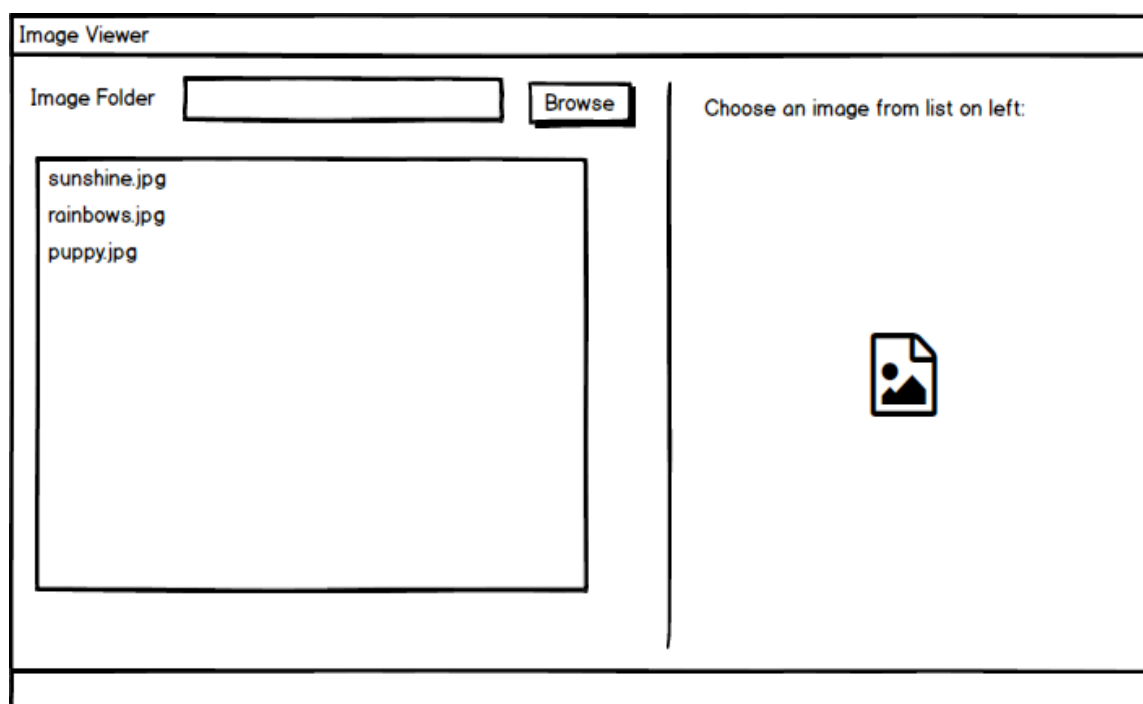


Рис. 2.2.3. Макет вікна.

Для цього прикладу буде багато коду, але відповідно з описом. Тож варто буде розглядати це меншими шматками.

Можна створити файл з іменем `img_viewer.py` у своєму редакторі `Python`. Потім додати такий код (див. рис. 2.2.4):

```
# img_viewer.py

import PySimpleGUI as sg
import os.path

# First the window layout in 2 columns

file_list_column = [
    [
        sg.Text("Image Folder"),
        sg.In(size=(25, 1), enable_events=True, key="-FOLDER-"),
        sg.FolderBrowse(),
    ],
    [
        sg.Listbox(
            values=[], enable_events=True, size=(40, 20), key="-FILE LIST-"
        )
    ],
]
]
```

Рис. 2.2.4. Віджети та їх створення.

Тут, у рядках 3 і 4, імпортуються PySimpleGUI і модуль os Python. Потім у рядках з 8 по 19 створюється вкладений список елементів, які представляють вертикальний стовпець інтерфейсу користувача. Це створить кнопку «Огляд», за допомогою якої можна знайти папку з зображеннями.

Ключовий параметр важливий. Це те, що використовується для ідентифікації конкретного елемента у GUI. Для елемента керування введенням тексту “In()” потрібно надати йому ідентифікатор “-FOLDER-”. Можна використати це пізніше для доступу до вмісту елемента. Також присутня функція (вмикати або вимикати події) для кожного елемента за допомогою параметра “enable_events”.

Елемент “Listbox()” відображає список шляхів до зображень, які потім можна вибрати для відображення. Також можна попередньо заповнити Listbox() значеннями, передавши список рядків.

Коли вперше завантажується інтерфейс користувача, хочеться, щоб “Listbox()” був порожнім, тому потрібно передати йому порожній список. Вмикаються події для цього елемента, встановлюється його розмір і надається йому унікальний ідентифікатор, як відбувалось з елементом input. Тепер можна подивитися на праву колонку елементів (див. рис. 2.2.5):

```

21 # For now will only show the name of the file that was chosen
22 image_viewer_column = [
23     [sg.Text("Choose an image from list on left: ")],
24     [sg.Text(size=(40, 1), key="-TOUT-")],
25     [sg.Image(key="-IMAGE-")],
26 ]

```

Рис. 2.2.5. Огляд колонки.

Список списків у рядках з 22 по 26 створюється три елементи. Перший елемент повідомляє користувачеві, що він повинен вибрати зображення для показу. Другий елемент відображає назву вибраного файлу. Третій показує “Image()”. Варто звернути увагу, що елемент “Image()” також має набір ключів, тому можна легко повернутися до елемента пізніше.

Для отримання додаткової інформації про елемент “Image()” варто переглянути документацію. Наступний фрагмент коду визначає сам макет (див. рис. 2.2.6):

```

28 # ----- Full layout -----
29 layout = [
30     [
31         sg.Column(file_list_column),
32         sg.VSeparator(),
33         sg.Column(image_viewer_column),
34     ]
35 ]

```

Рис. 2.2.6. Приклад компоновання.

Останній список, у рядках з 29 по 35, містить код, який керує тим, як елементи розміщуються на екрані. Цей код містить два елементи “Column()” із “VSeperator()” між ними. “VSeperator()” є псевдонімом для “VerticalSeparator()”. Можна дізнатися більше про те, як працюють “Column()” і “VSeperator()”, прочитавши відповідні сторінки документації. Щоб додати макет у своє вікно, можна зробити це (див. рис. 2.1.7):

```
37 window = sg.Window("Image Viewer", layout)
```

Рис. 2.2.7. Компонування для вікна.

Після того, як вийшло розібратися з інтерфейсом користувача, можна подивитися на код циклу подій. Ось перша інструкція (див. рис. 2.2.8):

```
39 while True:
40     event, values = window.read()
41     if event == "Exit" or event == sg.WIN_CLOSED:
42         break
```

Рис. 2.2.8. Цикл подій.

Цикл подій містить логіку самої програми. Тут витягуються події та значення з вікна. Подія буде ключовим рядком будь-якого елемента, з яким взаємодіє користувач. Змінна `values` містить словник Python, який відображає ключ елемента на значення. Наприклад, якщо користувач вибирає папку, тоді `"-FOLDER-"` буде відповідати шляху до папки.

```
44 # Folder name was filled in, make a list of files in the folder
45 if event == "-FOLDER-":
46     folder = values["-FOLDER-"]
47     try:
48         # Get list of files in folder
49         file_list = os.listdir(folder)
50     except:
51         file_list = []
52
53     fnames = [
54         f
55         for f in file_list
56         if os.path.isfile(os.path.join(folder, f))
57         and f.lower().endswith((".png", ".gif"))
58     ]
59     window["-FILE LIST-"].update(fnames)
```

Рис. 2.2.9. Цикл подій.

Умовні оператори використовуються для контролю того, що відбувається. Якщо подія дорівнює "Вихід" або користувач закриває вікно, тоді відбувається вихід з циклу. Тепер можна поглянути на першу частину наступного умовного оператора в циклі (див. рис. 2.2.9).

Цього разу перевіряється подія за ключем "-FOLDER-", який посилається на створений раніше елемент "In()". Якщо подія існує, відомо, що користувач вибрав папку, і використовується "os.listdir()", щоб отримати список файлів. Потім фільтрується цей список лише до файлів із розширенням «.png» або «.gif». Тепер можна поглянути на наступну частину умовного оператора (див. рис. 2.2.10):

```

60 elif event == "-FILE LIST-": # A file was chosen from the listbox
61     try:
62         filename = os.path.join(
63             values["-FOLDER-"], values["-FILE LIST-"][0]
64         )
65         window["-TOUT-"].update(filename)
66         window["-IMAGE-"].update(filename=filename)
67     except:
68         pass

```

Рис. 2.2.10. Логіка опрацювання деякої події.

Якщо подія дорівнює "-FILE LIST-", відомо, що користувач вибрав файл у "Listbox()", і необхідно оновити елементи "Image()", а також елементи "Text()", які показують назву вибраного файлу на право. Останній біт коду - це те, що завершує програму (див. рис. 2.2.11):

```

71 | window.close()

```

Рис. 2.2.11. Вихід з програми.

Коли користувач натискає кнопку Вихід, програма має закритися. Для цього можна скористатися "window.close()". Технічно можна залишити цей рядок у своєму коді, і Python все одно завершить програму, але завжди варто прибирати за програмою. Крім того, якщо використовується веб-порт PySimpleGUI і не закривається вікно належним чином, порт залишиться

відкритим. Тепер необхідно запустити код, і можна побачити такий інтерфейс (див. рис. 2.2.12):

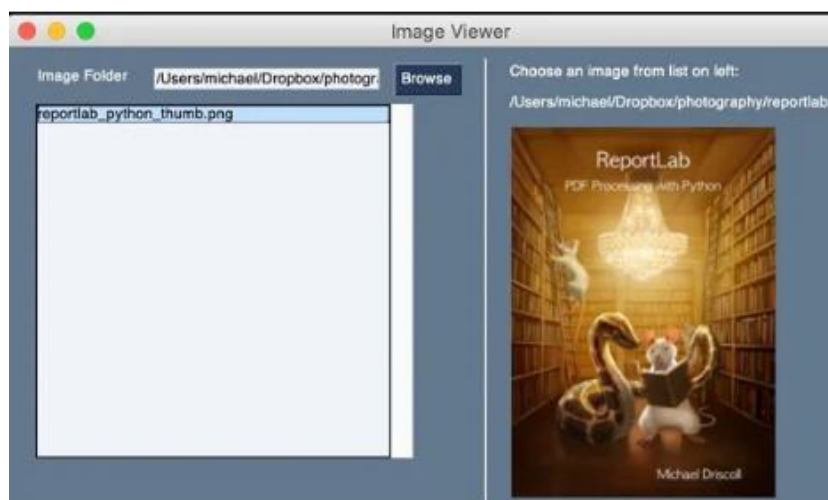


Рис. 2.2.12. Приклад віконної програми.

Можна також скористатися кнопкою «Огляд», щоб знайти на комп'ютері папку із зображеннями, аби спробувати цей код. Або ж скопіювати та вставити шлях до файлу в елемент "Text()". Зараз актуальна тема комп'ютерного зору. Python дозволяє отримати доступ до комп'ютерного зору за допомогою пакета `opencv-python`, який є оболонкою популярної програми OpenCV. Ось приклад того, як виглядатиме GUI (див. рис. 2.2.13):



Рис. 2.2.13. Віконна програма для захоплення відеопотоку з камери.

2.3. Windows API для взаємодії з операційною системою

Windows API — це велика, складна тема, за якою стоять десятиліття історії розробки та дизайну. Хоча це надто багато, щоб охопити його в одному пункті, навіть побіжних знань достатньо, щоб покращити аналіз подій і основні навички аналізу зловмисного програмного забезпечення. Розуміння того, як працює Windows, може допомогти захисникам: краще розуміти загрози та захищатися від них, знати, де можуть ховатися зловмисники, і визначати вдосконалення для обмеження можливостей зловмисників.

Це глибоке технічне занурення Windows надасть огляд того, що таке Windows API, як і чому виконувані файли використовують API, і як застосувати ці знання для покращення захисту. Простіше кажучи, Windows API — це набір стандартних бібліотек, які розробники можуть використовувати для взаємодії з операційною системою Windows. Ці бібліотеки складаються з функцій, які представлені (або експортовані) у різні бібліотеки динамічного компонування (DLL) у всій операційній системі Windows.

Надання API, як-от Windows API, має багато переваг, не останньою з яких є абстракція. Абстракція — поширене поняття в інформатиці та програмуванні, воно просте, але потужне. За словами Джона В. Гуттага, «сутність абстракцій полягає в збереженні інформації, яка є релевантною в даному контексті, і забутті інформації, яка є нерелевантною в цьому контексті». Чудовим прикладом абстракції є мережева модель OSI, у якій кожен рівень моделі не стосується роботи чи вмісту, який обробляється іншими рівнями. Інтернет-протокол функціонує однаково незалежно від того, передається він через радіосигнали WiFi або кабелі Ethernet, а TCP функціонує однаково незалежно від того, чи передається він через IPv4 чи IPv6. Ще однією перевагою Windows API є його стандартизація. Оскільки Microsoft зробила Windows API настільки надійним, розробники мають миттєвий доступ до величезної кількості функцій, але всі вони мають доступ до тих самих функцій. Це дозволяє комусь розробити програму на одному комп'ютері та мати гарантії, що вона працюватиме на іншій системі. Завдяки зворотній сумісності можна навіть очікувати, що програма буде

працювати на різних версіях операційної системи (теоретично...).

Не всі DLL створюються однаково. На додаток до стандартних DLL, які складають Windows API, існує набір DLL, які складають Native API. Нативний API працює так само, як Windows API, але є кілька ключових відмінностей. Власний API розроблено для підтримки меншого набору основних функцій до завантаження Windows API. Таким чином, будь-які компоненти Windows, завантажені до Windows API, призначені для використання Native API, включаючи сам Windows API. Нативний API відкриває ntdll.dll, але більшість функцій є недокументованими, тому нативні програми зазвичай розробляє лише Microsoft. Ntdll.dll взаємодіє з ntoskrnl.exe, де фактично реалізовано більшість функцій Native API.

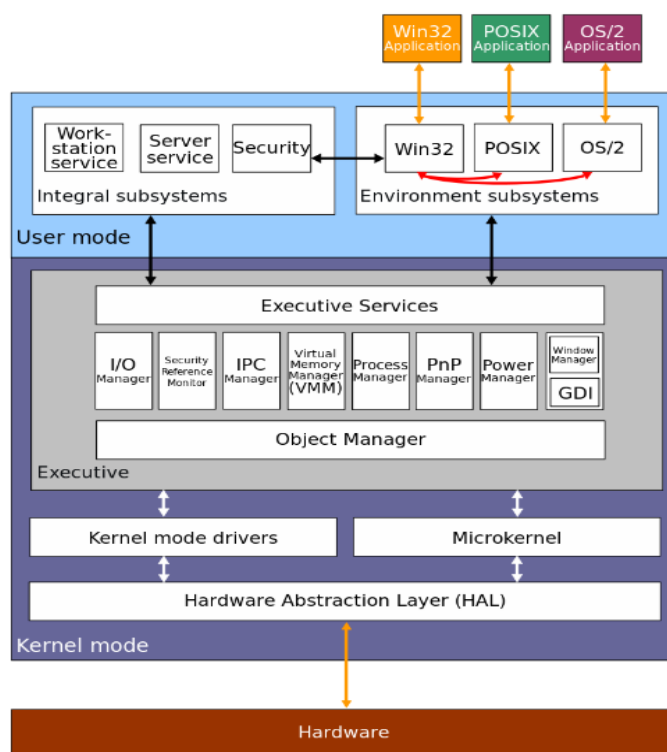


Рис. 2.3.1. Блок-схема архітектури Windows 2000.

Ядро Windows реалізовано в ntoskrnl.exe, але і ntoskrnl.exe, і Win32k.sys забезпечують інтерфейси між режимом користувача та режимом ядра. Win32k.sys — це драйвер, який містить графічні функції на рівні ядра. Під ядром

Windows знаходиться Рівень апаратної абстракції (HAL), який реалізований у hal.dll. HAL гарантує, що Windows працюватиме незалежно від основного апаратного забезпечення (зверніть увагу на «Абстракція» прямо в назві) (див. рис. 2.3.1).

Одним із найпоширеніших типів файлів у системі Windows є файл EXE. Структура цих файлів відома як формат Portable Executable (PE). Формат PE фактично використовується для багатьох різних розширень файлів (CPL, OCX і SYS, щоб назвати декілька), але основна увага приділяється розширенню файлу DLL. Хоча вони мають спільну файлову структуру, файли EXE і DLL зазвичай дещо відрізняються один від одного. Окрім позначки, яка вказує, що файл є файлом DLL, файли EXE мають дуже мало експортованих функцій, тоді як файли DLL мають багато. Різниця між імпортом і експортом досить проста, але розуміння різниці проливає світло на те, чому EXE відрізняються від DLL (див. рис. 2.3.2).

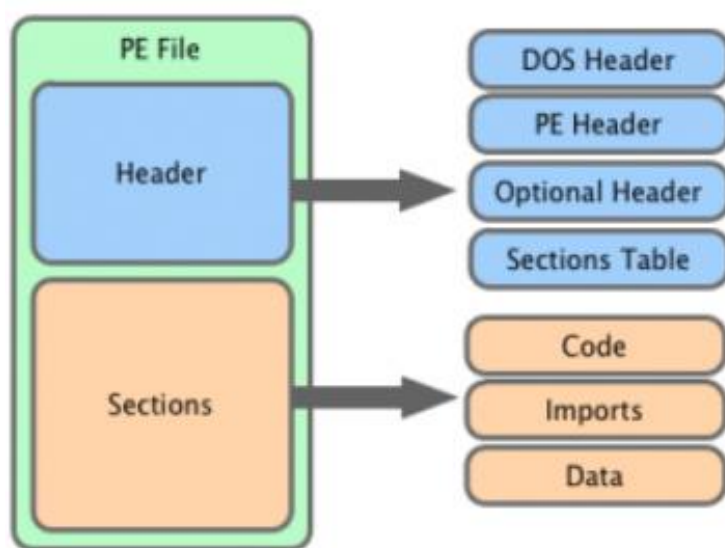


Рис. 2.3.2. Блок-схема архітектури Windows 2000.

Імпортування функції, по суті, запитує у Windows можливість використовувати певну функцію. Експорт функції надає іншим файлам доступ до можливостей експортованої функції за умови, що вони можуть знайти експортовану функцію. Наприклад, файл під назвою executable.exe може

знадобитися ініціалізувати сокет Windows за допомогою функції `WSAStartup` у бібліотеці `ws2_32.dll`. Цей файл, який бажає використовувати цю функцію, повинен пов'язати функцію `WSAStartup` із собою через процес, який називається зв'язуванням. Є кілька способів пов'язати функції з файлом; статичні, динамічні та під час виконання.

Якщо `executable.exe` використовує статичне зв'язування, фактичний код для `WSAStartup` буде скопійовано з `ws2_32.dll` у `executable.exe` під час його компіляції, додаючи функції `WSAStartup` безпосередньо до себе. За допомогою динамічного зв'язування `executable.exe` додає до заголовка PE інформацію з описом усіх функцій, які має надати операційна система, у цьому прикладі `WSAStartup`. Ця інформація називається таблицею імпорту, і вона має цікаві застосування для аналізу шкідливих програм, але про це пізніше.

Файл `executable.exe` також має опцію компонування під час виконання, у якому функції або бібліотеки завантажуються лише тоді, коли вони потрібні. Це досягається за допомогою двох додаткових функцій; `LoadLibrary` і `GetProcAddress`. Спочатку викликається `LoadLibrary` із зазначенням імені бібліотеки (`ws2_32.dll`), яку потрібно завантажити. Після завантаження `LoadLibrary` повертає дескриптор `ws2_32.dll`, який є посиланням на завантажену бібліотеку. Потім цей дескриптор разом із конкретною функцією (`WSAStartup`) можна передати в `GetProcAddress`, який повертає адресу пам'яті для вказаної функції. Оскільки функції, які будуть використовуватися через зв'язування під час виконання, вказуються під час виконання, вони не розташовані в одному зручному для читання місці. Це може бути використано авторами зловмисного програмного забезпечення, щоб приховати, які функції вони імпортують, таким чином приховуючи деякі можливості зловмисного програмного забезпечення.

Усі функції, які можна імпортувати, мають бути десь визначені; у файлах, які визначають ці експортовані функції. Експорт функцій — це просто спосіб зробити функціональні можливості цих функцій доступними для використання іншими файлами. На фундаментальному рівні Windows API — це велика колекція файлів, які екпортують більшу кількість функцій (див. рис. 2.3.3).

```

1  HANDLE OpenProcess(
2      DWORD dwDesiredAccess,
3      BOOL  bInheritHandle,
4      DWORD dwProcessId
5  );

```

Рис. 2.3.3. OpenProcess.

Функція OpenProcess повертає дескриптор процесу (HANDLE), який, по суті, є маркером, який дозволить отримати доступ до пам'яті поточного запущеного процесу. Для цього потрібні три аргументи:

- dwDesiredAccess — визначає доступ, який буде призначено повернутому дескриптору процесу. Значення цього параметра можна знайти тут;
- bInheritHandle — дозволяє всім підпроцесам, створеним цим процесом, використовувати один дескриптор;
- dwProcessId — ідентифікатор процесу (PID) запущеного процесу.

```

1  LPVOID VirtualAllocEx(
2      HANDLE hProcess,
3      LPVOID lpAddress,
4      SIZE_T dwSize,
5      DWORD  flAllocationType,
6      DWORD  flProtect
7  );

```

Рис. 2.3.4. VirtualAllocEx.

Функція VirtualAllocEx (див. рис. 2.3.4) виділить пам'ять у віртуальному адресному просторі цільового процесу. Для цього потрібно п'ять аргументів:

- hProcess — дескриптор запущеного процесу (отримано з OpenProcess);
- lpAddress — вказівник на бажану початкову адресу для виділеної пам'яті;

- `dwSize` — обсяг пам'яті, який потрібно виділити;
- `flAllocationType` — визначає тип пам'яті для виділення;
- `flProtect` — дозволи, які буде встановлено для виділеної пам'яті. Типи дозволів, які можна встановити, можна знайти тут.

```

1  BOOL WriteProcessMemory(
2      HANDLE hProcess,
3      LPVOID lpBaseAddress,
4      LPCVOID lpBuffer,
5      SIZE_T nSize,
6      SIZE_T *lpNumberOfBytesWritten
7  );

```

Рис. 2.3.5. `WriteProcessMemory`.

Функція `WriteProcessMemory` (див. рис. 2.3.5) фактично записуватиме дані в простір пам'яті, який виділяється за допомогою функції `VirtualAllocEx`. Для цього потрібно п'ять аргументів:

- `hProcess` — дескриптор процесу (отримано з `OpenProcess`);
- `lpBaseAddress` — вказівник на базову адресу виділеної пам'яті, куди потрібно записувати наші дані;
- `lpBuffer` — покажчик на дані, який записуватиметься у виділену пам'ять;
- `nSize` — кількість байтів із нашого буфера, які запишуться у виділену пам'ять;
- `lpNumberOfBytesWritten` — вказівник на змінну, яка зберігатиме кількість байтів, які були записані у виділену пам'ять.

РОЗДІЛ III. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1. Постановка задачі та архітектура проєкту

На основі раніше отриманого досвіду (щодо створення консольної програми з функцією КЗ), отримані значні результати для подальшого розвитку в дослідженні та розробці ПЗ. З отриманих результатів розробки ПЗ: автоматизації ігрового процесу, роботи з ШІ та взаємодії з грою, зрозуміло, що проєктування відіграє значну роль у реалізації вихідної ідеї, а грамотний підхід до розробки забезпечує якість ПЗ та ефективність його роботи з реальними задачами.

Консольний проєкт (бот) для автоматизації ігрового процесу, з яким у мене є досвід, мав підстави розвиватися і надалі, адже програма без взаємодії з людиною не могла бути користувацькою, тому що не було інтерфейсної частини. Ідеєю цієї роботи був розвиток вже розробленого раніше проєкту, без модифікації коду, а розробки нового, ефективнішого, як по використанню ресурсів ПК, так і по виконанні поставлених задач в грі. На основі отриманих результатів з попередніх напрацювань, зрозуміло також, що постановка задачі та архітектурне рішення відіграють чи немалу роль в розробці проєкту.

Постановка задачі:

- Розробити віконний додаток з інтеграцією таких функцій, як: обробка зображень, розпізнавання об'єктів, програмна взаємодія з операційною системою та аналіз вихідних даних;
- Створити інтерфейс для: постійного моніторингу активності та управління програмою Для прикладу, використати бота для автоматизації ігрового процесу;
- Дослідити зміни показників навантаження системи комп'ютера (в залежності від оптимізації програми), в тому числі часткове перенесення задач з центрального процесора на дискретну карту.

Архітектура проєкту буде виглядати наступним чином:

- Модулі програми:
 - Модулі, які будуть відповідати за конкретний функціонал, як от: розпізнавання, обробка, аналіз тощо;
 - Головна програма для об'єднання всіх функціональних модулів в одному (точка входу);
 - Наявність скриптів для тестування модулів.
- Модуль з даними для нейромережі (модель, дата сет, конфігурації);
- Модуль для статичних файлів;
- Модуль для скриптів запуску;
- Модуль для опрацювання вихідних даних;
- Модуль для візуального представлення даних.

Етапи розробки проєкту, можна поділити на:

- Збір даних із результату роботи попередніх програм;
- Урахування всі помилок останнього проєкту;
- Розробка макета віконного додатка;
- Розробка модулів для роботи КЗ та розпізнавання мовлення;
- Розробка модуля для програмної взаємодії з ОС Windows;
- Інтеграція функціонала з модулів в одній програмі.

Етапи дослідження проєкту, можна поділити на:

- Робота програми в штучних умовах;
- Аналіз результатів тестування для корегування програми;
- Практичне застосування програми в грі, спосіб отримання даних;
- Аналіз результатів дослідження та візуалізація даних.

3.2. Проектування дизайну віконного додатку

Графічний інтерфейс є одним із ключових елементів цієї роботи. Перед розробкою віконного додатка необхідно спроектувати дизайн вікна, який буде відповідати користувачу по всім параметрам.

Дизайн віконного додатка може бути спроектованим і в програмі Paint, як у моєму випадку. Підхід такий, що необхідно продумати усі деталі, аби користувач, в незалежності від пристрою, міг використовувати додаток повною мірою. Створення адаптивності, як вікна, так і компонентів на ньому, що до розширення дисплею користувача, на жаль, не було взято до уваги. Натомість вирішено визначити розмір вікна фіксованим значенням. Розмір віконного додатка становить 640 на 360 пікселів, зберігається в конф. файлі (див. рис. 3.2.1).

```
{
  "FPS": 60,
  "WINDOW_SIZE": [
    640,
    360
  ],
  "DISCONNECT_INFO": "disconnect_info.png",
  "WINDOW_ICON": "window_icon.ico",
  "PERMISSION_FILE": "permission.json",
  "TARGET_WINDOW": "??",
  "TARGET_FOLDER": "C:/Users/bogda/Desktop/BIGPROJECT/mobs"
}
```

Рис. 3.2.1. Конфігураційний файл програми.

Макет вікна буде охоплювати наступні компоненти:

- Кнопки для активації дій;
- Поля для вводу тексту;
- Менеджери файлового провідника;
- Текстові вікна для виводу текстової інформації;
- Вікна для виводу зображення.

Основною задачею буде грамотне компонування усіх елементів на одному полотні. Перший макет віконного додатка буде виглядати наступним чином (див. рис. 3.2.2):

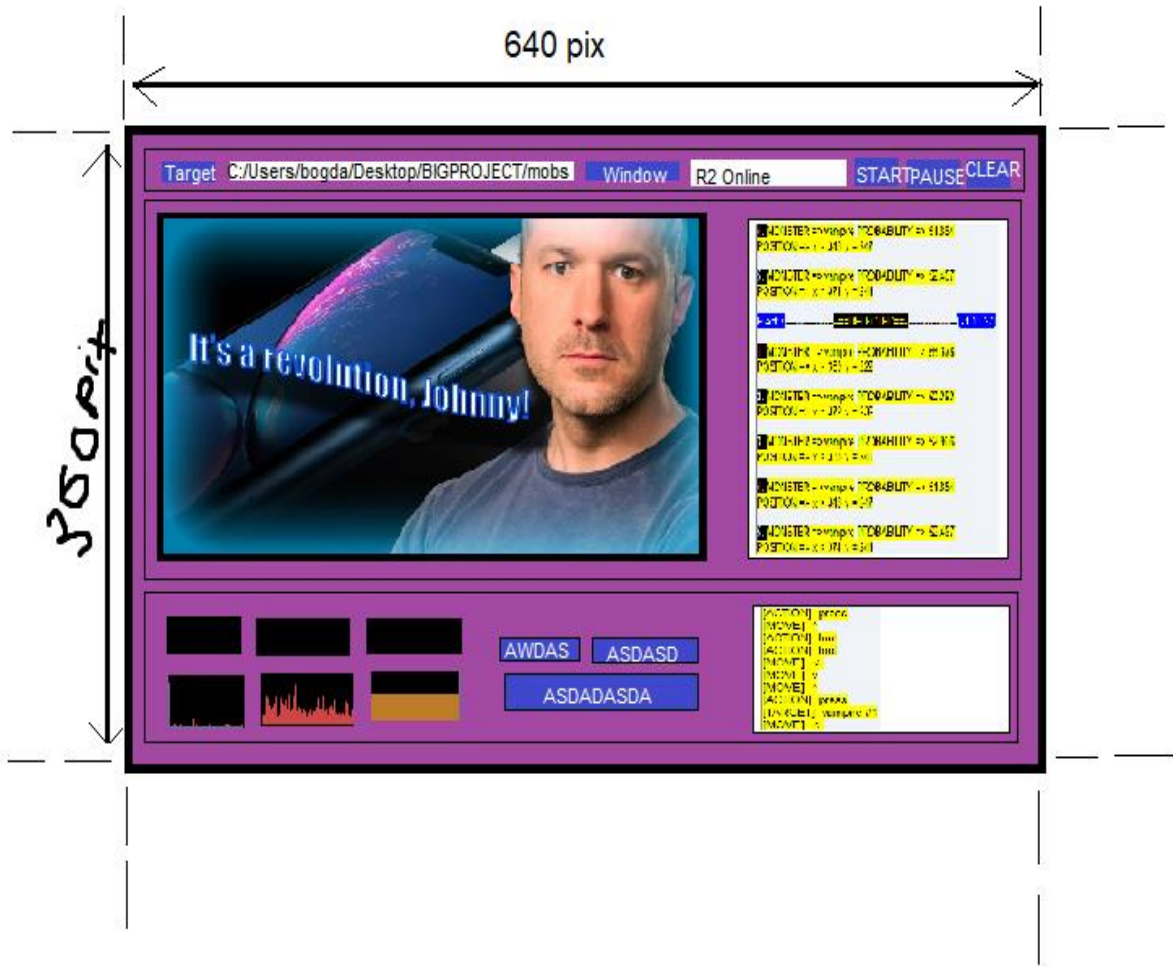


Рис. 3.2.2. Макет віконного додатка.

Вікно можна поділити на 3 окремі рамки:

- Вікно налаштувань;
- Інформаційне вікно;
- Вікно моніторингу.

На вікні налаштувань розміщені компоненти для налаштування програми перед її запуском. За допомогою кнопки “Target” можна відкрити менеджер файлового провідника з опцією вибору цільової директорії (див. рис. 3.2.3). Ця опція передбачає вибір користувачем тої директорії, у якій містяться конфігураційні файли для роботи з ШІ, як от: натренована модель, дані сет та анотації, додаткові налаштування ШІ та “логі”. У цьому випадку, програма записує шлях до директорії та виводить його на вікно.

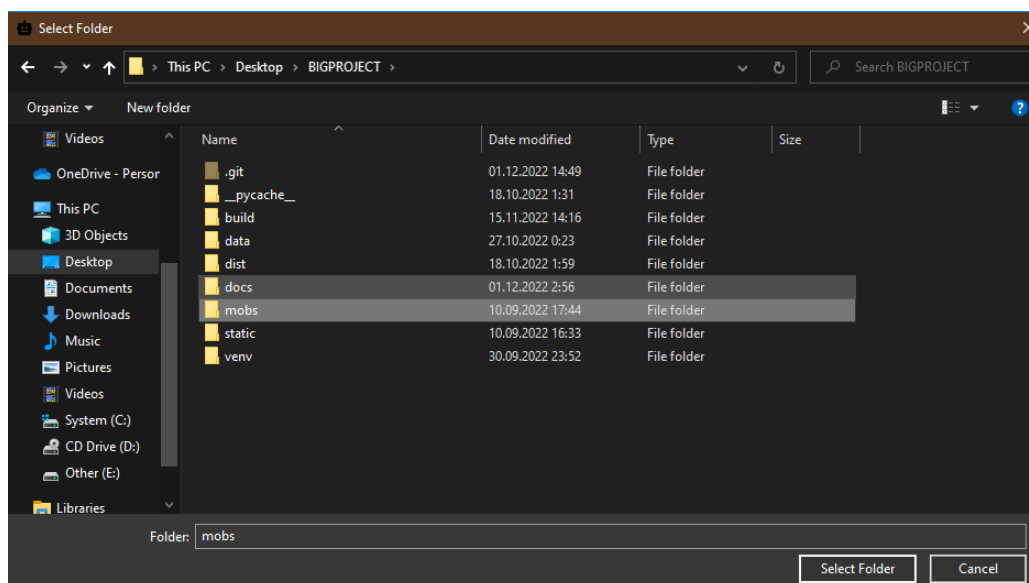


Рис. 3.2.3. Вибір директорії налаштувань ІІІ.

Наступною важливою компонентою є кнопка “Window”, вона слугує для вибору цільового вікна, яку потрібно захоплювати програмою. Вікно вибору, що появляється після натискання по кнопці, пропонує назви тих програм, які доступні на цей момент, а саме це активні вікна в ОС Windows (див. рис. 3.2.4). Після вибору цільового вікна програма записує назву та виводить його на екран. Окрім основних компонент для налаштування, додано також 3 кнопки для керування програмою:

- СТАРТ;
- ПАУЗА;
- ОЧИЩЕННЯ.

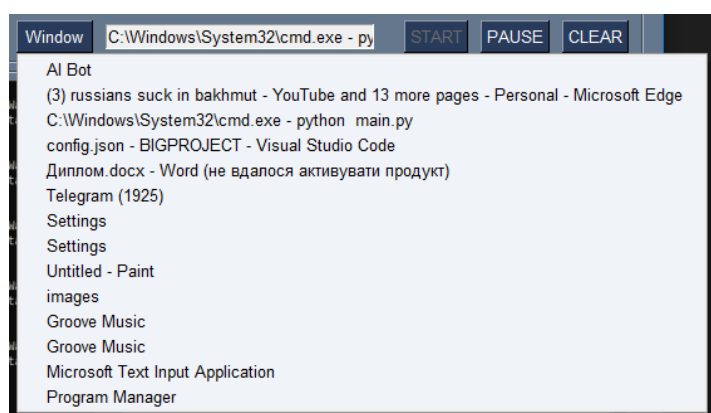
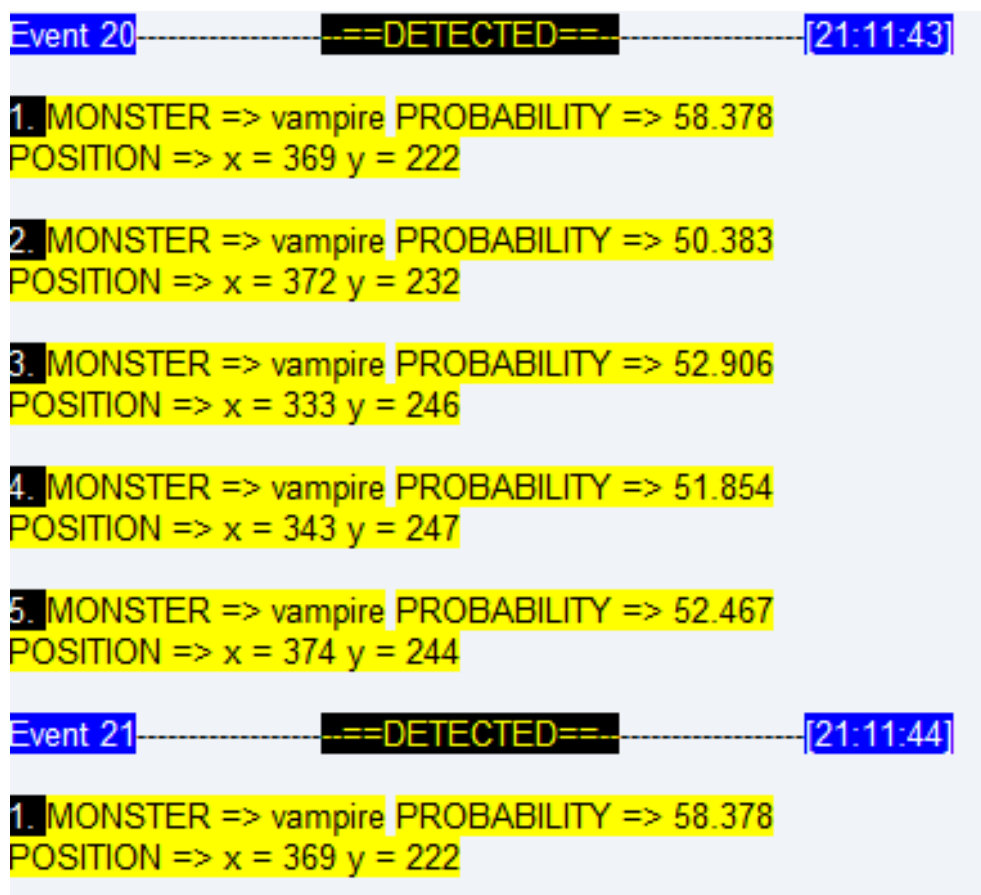


Рис. 3.2.4. Список доступних для вибору активних вікон.

Кнопка “Start” запускає аналіз потокового відео цільового вікна програми, що дозволяє запускати або зупиняти процес, а відповідно кнопка “Pause” робить те саме, але також за допомоги неї можна зупинити трансляцію вікна загалом, або ж його відновити.

Найбільше за розміром підвікно у програмі є інформаційне вікно. Воно містить фрейм з виводом потокового відео з цільового вікна програми ОС з/без попередньої обробки. Одне із таких вікон розташоване зліва, а друге справа – велике текстове поле. У текстовому полі виводяться інформаційні текстові повідомлення про наявність розпізнаних об’єктів, якщо такі є. Формат виводу виглядає наступним чином (див. рис. 3.2.5):



```

Event 20-----==DETECTED==-----[21:11:43]
1. MONSTER => vampire PROBABILITY => 58.378
   POSITION => x = 369 y = 222
2. MONSTER => vampire PROBABILITY => 50.383
   POSITION => x = 372 y = 232
3. MONSTER => vampire PROBABILITY => 52.906
   POSITION => x = 333 y = 246
4. MONSTER => vampire PROBABILITY => 51.854
   POSITION => x = 343 y = 247
5. MONSTER => vampire PROBABILITY => 52.467
   POSITION => x = 374 y = 244
Event 21-----==DETECTED==-----[21:11:44]
1. MONSTER => vampire PROBABILITY => 58.378
   POSITION => x = 369 y = 222
  
```

Рис. 3.2.5. “Логи” результатів розпізнавання.

У текстовому полі зображено подія: номер, час та інформація по зображенні. Для кожного розпізнаного об’єкта виводиться: назва, точність та позиція на цільовому вікні.

Останнє підвікно програми, це вікно моніторингу. Це підвікно охоплює інформативну дошку із відомостями про стан завантаження різних ресурсів ПК. Дані представлені у вигляді графіків (див. рис. 3.2.6): CPU, GPU, NET, RAM. Ці відомості необхідно подавати, щоб моніторити стан системи ПК, аби контролювати ефективність роботи програми, чи розподіляти ресурси ПК.

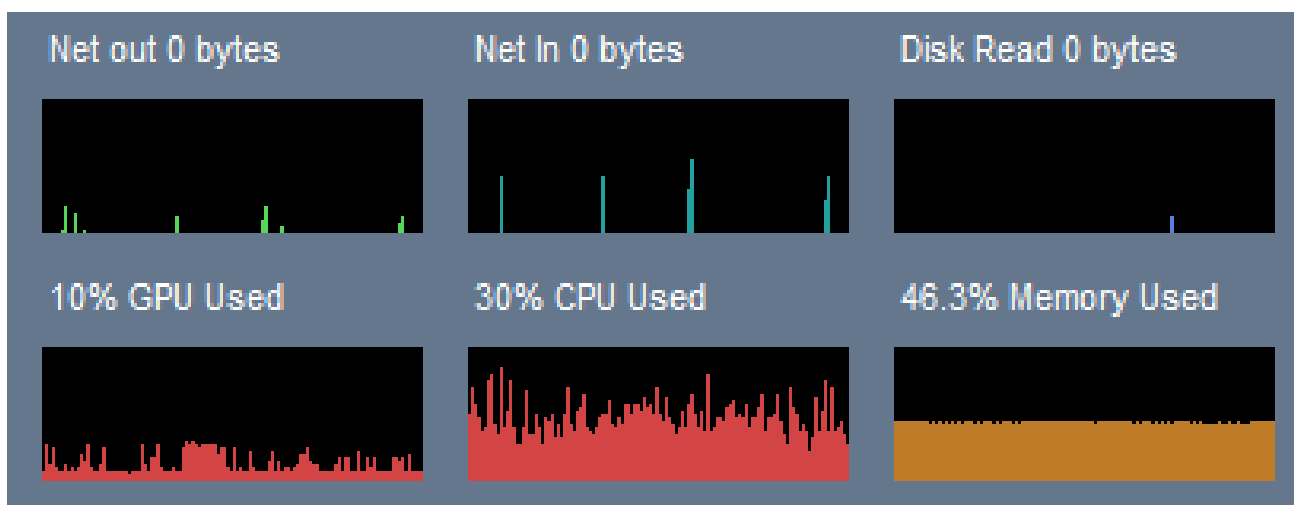


Рис. 3.2.6. “Дашборд” системи моніторингу.

Ключовими компонентами у роботі бота є кнопки його керування:

- CHECK SETTINGS;
- ACTIVATE BOT;
- SPEECH INTERVENE.

Користувач може приступити до роботи програми лише, якщо вона надасть йому такий доступ. Перевірка програмою відбувається в такому порядку: користувач надає необхідні файли для роботи ІІІ; вибирається цільове вікно, яке відповідає конфігураційним файлам, тоді користувач натискає кнопку перевірки. Якщо все вірно, то програма розблоковує кнопку “START”. Активація бота через кнопку “START” приведе до самостійної його діяльності в грі. Він втручається в ігровий процес на основі логіки його поведінки. Також чи мало важливу роль відіграє кнопка “SPEECH INTERVENE”, яка відведена для втручання людини у ігровий процес бота. Команди, які поступають для бота - інтерпретовані у “людську мову” (див. рис. 3.2.7).

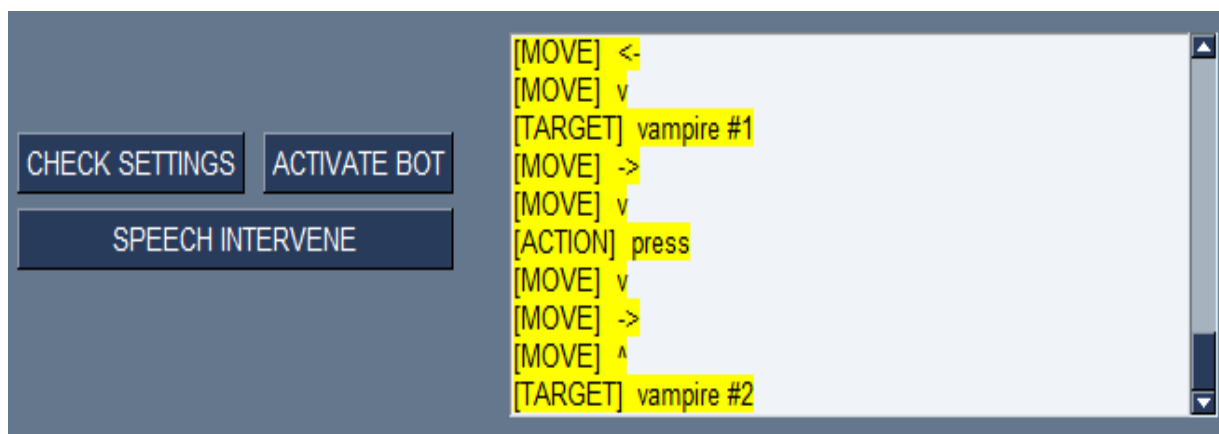


Рис. 3.2.7. Вікно виводу команд бота та кнопки для керування.

3.3. Написання модулів для роботи комп'ютерного зору та розпізнавання мовлення

Як описано вище, бібліотеки, які необхідні для роботи вже визначені, а документація - перечитана. Будемо слідувати архітектурі, яка раніше також була визначена. Отже, потрібно написати 2-3 модулі, кожен з яких буде відповідати конкретній задачі.

Один з ключових модулів відповідає за захоплення потокового відео вибраного вікна програми в ОС Windows. У ньому використовується бібліотека “win32gui”. Для роботи із цією бібліотекою створений клас із конструктором та методами для роботи з даними. Екземпляр цього класу ініціалізується всього лиш назвою цільового вікна, а, все необхідне, для отримання вихідних даних є виклик метода “get_screenshot”. При створенні екземпляра, ведеться пошук вікна за іменем та отримання спеціального номера HWND, яке ОС надає при створенні. За допомогою цього номера, програма прямо може звертатися до цього вікна, отже можна отримати його розмір та положення і налаштувати рамку для отримання саме тих пікселів, які знаходяться у визначеній нами рамці (див. рис. 3.3.1). Отже, отримуємо необхідні параметри для роботи з цільовою програмою. У класі описано також додаткові методи для отримання списку всіх вікон, а також їхнього положення. Але суть полягає в одному методі, який відповідає за видобування та обробку зображення з виводом у тому форматі, який підходить для модуля з ІШ.

```
def __init__(self, window_name):
    # find the handle for the window we want to capture
    self.hwnd = win32gui.FindWindow(None, window_name)
    if not self.hwnd:
        raise Exception('Window not found: {}'.format(window_name))

    # get the window size
    window_rect = win32gui.GetWindowRect(self.hwnd)
    self.w = window_rect[2] - window_rect[0]
    self.h = window_rect[3] - window_rect[1]

    # account for the window border and titlebar and cut them off
    border_pixels = 8
    titlebar_pixels = 30
    self.w = self.w - (border_pixels * 2)
    self.h = self.h - titlebar_pixels - border_pixels
    self.cropped_x = border_pixels
    self.cropped_y = titlebar_pixels

    # set the cropped coordinates offset so we can translate screenshot
    # images into actual screen positions
    self.offset_x = window_rect[0] + self.cropped_x
    self.offset_y = window_rect[1] + self.cropped_y
```

Рис. 3.3.1. Ініціалізація екземпляра.

Метод “get_screenshot” (див. рис. 3.2.2) при виклику вибирає вікно за визначеним HWND після цього створюється “handle”, за допомогою якого можна отримати “compatible DC”, який своєю чергою необхідний для отримати бітової карти вікна, задаючи параметри ширини та висоти. Отримання бітової карти зображення завершується, якщо копіювати його з цільового вікна. Формат зображення визначений, як “pillow image”, що дає можливість в широкому плані співпрацювати із цим типом для подальших перетворень. Після такої процедури важливо завершити роботу із вікном, також очищуються тимчасові об’єкти та повертається матриця зображення.

Окрім самого інструмента для отримання вихідного зображення цільового вікна, також необхідно підігнати зображення під певний розмір, аби вивести на екран та надати ІШ. Такі функції описані в спеціальному модулі під назвою “utils” (див. рис. 3.3.3). Бібліотеки OpenCV та PIL Image є потужними інструментами, які дозволяють з легкістю модифікувати вміст зображення. Але PIL Image має перевагу, саме для роботи у зв’язці для модулем зі ІШ.

```
def get_screenshot(self):
    # get the window image data
    wDC = win32gui.GetWindowDC(self.hwnd)
    dcObj = win32ui.CreateDCFromHandle(wDC)
    cDC = dcObj.CreateCompatibleDC()
    dataBitMap = win32ui.CreateBitmap()
    dataBitMap.CreateCompatibleBitmap(dcObj, self.w, self.h)
    cDC.SelectObject(dataBitMap)
    cDC.BitBlt((0, 0), (self.w, self.h), dcObj,
               (self.cropped_x, self.cropped_y), win32con.SRCCOPY)

    # convert the raw data into a format opencv can read
    #dataBitMap.SaveBitmapFile(cDC, 'debug.bmp')
    signedIntsArray = dataBitMap.GetBitmapBits(True)
    img = np.fromstring(signedIntsArray, dtype='uint8')
    img.shape = (self.h, self.w, 4)

    # free resources
    dcObj.DeleteDC()
    cDC.DeleteDC()
    win32gui.ReleaseDC(self.hwnd, wDC)
    win32gui.DeleteObject(dataBitMap.GetHandle())
```

Рис. 3.3.2. Метод отримання зображення.

Окрім отримання зображення в момент часу, необхідно також його опрацювати. За опрацювання зображення відповідає код з курсової роботи, який був адаптований під цей проєкт. Цей код розбитий по модулях та запускається в головній програмі.

```
def create_image(image_size):
    img = Image.new('RGB', image_size, color='black')
    buffered = BytesIO()
    img.save(buffered, format="PNG")
    img_str = base64.b64encode(buffered.getvalue())
    return img_str

def resize_image(filename, image_size):
    abs_path = get_file_path(filename)
    img = Image.open(abs_path)
    resized_img = img.resize(image_size)
    buffered = BytesIO()
    resized_img.save(buffered, format="PNG")
    img_str = base64.b64encode(buffered.getvalue())
    return img_str
```

а)

```
def output_stream(stream, image_size, detector, detecting=True):
    detections = None
    if detecting:
        stream, detections = label_detecting(stream, detector)
    resized = cv2.resize(
        stream, image_size, interpolation=cv2.INTER_AREA)
    imgbytes = cv2.imencode('.png', resized)[1].tobytes()
    return detections, imgbytes
```

б)

Рис. 3.3.3. Інструменти для перетворення зображення:

а) інструмент для вікна; б) отримання обробленого зображення від ШІ.

Уся магія в SpeechRecognition відбувається з класом Recognizer. Звичайно, основною метою екземпляра Recognizer є розпізнавання мови. Кожен екземпляр

має різноманітні налаштування та функції для розпізнавання мовлення з джерела звуку. Створити екземпляр `Recognizer` легко: `r = sr.Recognizer()`.

Функція `accept_speech_from_mic()` приймає екземпляр `Recognizer` і `Microphone`, як аргументи та повертає словник із трьома ключами. Перший ключ, «успіх», є логічним значенням, яке вказує, чи був успішним запит API. Другий ключ, «помилка», або `None`, або повідомлення про помилку, яке вказує на те, що API недоступний або мова була нерозбірливою. Нарешті, ключ «транскрипція» містить транскрипцію звуку, записаного мікрофоном.

Функція спочатку перевіряє, чи аргументи розпізнавання та мікрофона мають правильний тип, і викликає помилку `TypeError`, якщо будь-який з них недійсний (див. рис. 3.3.4):

```
Python

if not isinstance(recognizer, sr.Recognizer):
    raise TypeError("`recognizer` must be `Recognizer` instance")

if not isinstance(microphone, sr.Microphone):
    raise TypeError("`microphone` must be a `Microphone` instance")
```

Рис. 3.3.4. Перевірка мікрофона.

Потім метод `listen()` використовується для запису вхідного сигналу мікрофона (див. рис. 3.3.5):

```
Python

with microphone as source:
    recognizer.adjust_for_ambient_noise(source)
    audio = recognizer.listen(source)
```

Рис. 3.3.5. Перевірка мікрофона.

Метод `adjust_for_ambient_noise()` використовується для калібрування розпізнавача та зміни умов шуму кожного разу, коли викликається функція `recogniz_speech_from_mic()`. Далі викликається розпізнавання `google()`, щоб

транскрибувати будь-яке мовлення в записі. Блок “try...except” використовується для перехоплення винятків “RequestError” і “UnknownValueError” та відповідної обробки. Успішне виконання запиту API, будь-які повідомлення про помилки та транскрибоване мовлення зберігаються в ключах успіху, помилки та транскрипції словника відповіді, який повертає функція “recogni_speech_from_mic()”.

```
Python
response = {
    "success": True,
    "error": None,
    "transcription": None
}

try:
    response["transcription"] = recognizer.recognize_google(audio)
except sr.RequestError:
    # API was unreachable or unresponsive
    response["success"] = False
    response["error"] = "API unavailable"
except sr.UnknownValueError:
    # speech was unintelligible
    response["error"] = "Unable to recognize speech"

return response
```

Рис. 3.3.6. Функція відповіді користувачу.

Можна перевірити функцію “recogni_speech_from_mic()”, зберігши наведений вище сценарій у файлі під назвою “guessing_game.py” та запустивши наступне в сеансі перекладача (див. рис. 3.3.7):

```
Python
>>> import speech_recognition as sr
>>> from guessing_game import recognize_speech_from_mic
>>> r = sr.Recognizer()
>>> m = sr.Microphone()
>>> recognize_speech_from_mic(r, m)
{'success': True, 'error': None, 'transcription': 'hello'}
>>> # Your output will vary depending on what you say
```

Рис. 3.3.7. Тестування результату вгадування слів.

Після друку деяких інструкцій і очікування в 3 секунди, цикл “for” використовується для керування кожною спробою користувача вгадати вибране слово. Перше, що всередині циклу “for” – це ще один цикл “for”, який запитує користувача щонайбільше “PROMPT_LIMIT” разів для припущення, намагаючись щоразу розпізнавати введення за допомогою функції “recognize_speech_from_mic()” і зберігаючи словник, повернутий до локальної змінної “guess” (див. рис. 3.3.8).

```
Python

for j in range(PROMPT_LIMIT):
    print('Guess {}. Speak!'.format(i+1))
    guess = recognize_speech_from_mic(recognizer, microphone)
    if guess["transcription"]:
        break
    if not guess["success"]:
        break
    print("I didn't catch that. What did you say?\n")
```

Рис. 3.3.8. Тестування результату.

Якщо ключ «транскрипції» вгадування не є “None”, тоді мова користувача була транскрибована, а внутрішній цикл завершується перервою. Якщо мова не була транскрибована, а для ключа «успіх» встановлено значення “False”, тоді сталася помилка API, і цикл знову припиняється з розривом. В іншому випадку запит API був успішним, але мова була невпізнаною. Користувач отримує попередження, і цикл “for” повторюється, даючи користувачеві ще один шанс на поточну спробу.

3.4. Реалізація програмної взаємодії з грою

Потрібно встановити кілька додаткових бібліотек Python. Вони забезпечують гарне обгортання Python для групи низькорівневого коду C, що значно полегшує процес і швидкість написання ботом сценаріїв. Деякі коди та

бібліотеки є специфічними для Windows. Можуть існувати еквіваленти Mac або Linux. Потрібно буде завантажити та встановити такі бібліотеки:

- Бібліотека зображень Python;
- Numpy;
- PyWin.

Ймовірно, підхід, який використовується, дещо відрізняється від того, який більшість очікує, коли думає про бота. Замість того, щоб створювати програму, яка знаходиться між клієнтом і сервером і впроваджує код (наприклад, бот Quake або C/S), бот повинен сидіти виключно «зовні». Необхідно використати методи комп'ютерного зору та виклики Windows API, щоб збирати інформацію та створювати рух. Завдяки такому підходу втрачається трохи витончених деталей і контролю, але компенсується це шляхом скорочення часу розробки та простоти використання. Автоматизацію певної ігрової функції можна виконати кількома короткими рядками коду, а повноцінного бота, який від початку до кінця (для простої гри) можна запустити за кілька годин.

Переваги цього швидкого підходу такі, що як тільки ознайомитись з тим, що комп'ютер легко «бачить», можна почати дивитися на ігри дещо інакше. Хороший приклад можна знайти в головоломках. Поширена конструкція передбачає використання людських обмежень швидкості, щоб змусити прийняти не оптимальне рішення. Весело (і досить легко) «кламати» ці ігри, записуючи рухи, які ніколи не могли б виконати люди. Ці боти також дуже корисні для тестування простих ігор - на відміну від людини, боту не буде нудно грати в один і той же сценарій знову і знову. Вихідний код для всіх прикладів підручника, а також для одного із готових прикладів ботів можна знайти тут.

Робота з “win32api” спочатку може бути трохи складною. Він огортає низькорівневий код Windows C, який, на щастя, дуже добре задокументований тут, але трохи схожий на лабіринт для навігації через вашу першу пару обходів. Перш ніж почати створювати сценарії будь-яких корисних дій, необхідно уважно поглянути на деякі функції API, на які потрібно покладатися. Якщо чітко зрозуміти кожен параметр, буде легко налаштувати їх для будь-яких цілей, які

потрібні в грі. Подія “win32api.mouse_event()” (див. рис. 3.5.1):

```
1 | win32api.mouse_event(  
2 |     dwFlags,  
3 |     dx,  
4 |     dy,  
5 |     dwData  
6 | )
```

Рис. 3.5.1. Події миші.

Перший параметр “dwFlags” визначає «дію» миші. Він керує такими речами, як рух, клацання, прокручування тощо. У наведеному нижче списку показано найпоширеніші параметри, які використовуються під час руху сценаріїв, “dwFlags”:

- win32con.MOUSEEVENTF_LEFTDOWN;
- win32con.MOUSEEVENTF_LEFTUP;
- win32con.MOUSEEVENTF_MIDDLEDOWN;
- win32con.MOUSEEVENTF_MIDDLEUP;
- win32con.MOUSEEVENTF_RIGHTDOWN;
- win32con.MOUSEEVENTF_RIGHTUP;
- win32con.MOUSEEVENTF_WHEEL.

Кожна назва має пояснення. Якщо є бажання надіслати віртуальне клацання правою кнопкою миші, необхідно передати “win32con.MOUSEEVENTF_RIGHTDOWN” у параметр “dwFlags”. Наступні два параметри, “dx” і “dy”, описують абсолютне положення миші вздовж осей “x” і “y”. Хоча можна було б використовувати ці параметри для сценарію руху миші, вони використовують систему координат, відмінну від тієї, яка використовується зараз. Отже, необхідно залишити їх нульовими і покладатимемося на іншу частину API для наших потреб у переміщенні миші. Четвертий параметр — “dwData”. Ця функція використовується, якщо (і тільки якщо) “dwFlags” містить “MOUSEEVENTF_WHEEL”. В іншому випадку значення можна опустити або

встановити на нуль. “dwData” вказує обсяг руху коліщатка прокручування миші. Якщо уявити гру з системою вибору зброї, подібною до Half-Life 2, де зброєю можна вибирати обертанням коліщатка миші, придувається така функція для перегляду списку зброї (див. рис. 3.5.3):

```
1 | def browseWeapons():
2 |     weaponList = ['crowbar', 'gravity gun', 'pistol'...]
3 |     for i in weaponList:
4 |         win32api.mouse_event(win32con.MOUSEEVENTF_MOUSEEVENTF_WHEEL,
```

Рис. 3.5.3. Функція вибору.

Тут бажано імітувати прокручування коліщатка миші для навігації нашим теоретичним списком зброї, тому передалась «дія» ”MOUSEEVENTF_WHEEL” у “dwFlag”. Не потрібні позиційні дані “dx” або “dy”, тому варто залишити їх нульовими, і потрібно було б прокручувати один клацання вперед для кожної «зброї» у списку, тому необхідно передавати ціле число 120 до “dwData” (кожен клацання колеса дорівнює 120). Як зрозуміло, робота з “mouse_event” — це просто підключення правильних аргументів до потрібного місця. Тепер перейдемо до деяких інших корисних функцій.

Створімо три нові функції. Одна загальна функція клацання лівою кнопкою миші та дві, які обробляють певні стани вниз і вгору. Отже можна відкрити code.py за допомогою IDLE та додати наступне до нашого списку операторів імпорту (див. рис. 3.5.4):

```
1 | import win32api, win32con
```

Рис. 3.5.4. Імпорт необхідних залежностей.

Як і раніше, це дає нам доступ до вмісту модуля через синтаксис “module.attribute”. Далі створимо нашу першу функцію клацання миші (див. рис. 3.5.5).

```

1  def leftClick():
2      win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,0)
3      time.sleep(.1)
4      win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0)
5      print "Click."          #completely optional. But nice for debuggi

```

Рис. 3.5.5. Функція реагування на клац миші.

Згадайте, що все, що тут відбувається, це призначення «дії» першому аргументу “mouse_event”. Нам не потрібно передавати жодну позиційну інформацію, тому залишаємо параметри координат (0,0), і нам не потрібно надсилати жодну додаткову інформацію, тому “dwData” пропускається. Функція “time.sleep(.1)” повідомляє Python зупинити виконання на час, указаний у дужках. Додамо їх у наш код, як правило, на дуже короткий проміжок часу. Без цього «клацання» може випереджати себе та спрацювати до того, як меню матиме можливість оновитися. Отже, те, що виконано тут, це загальне клацання лівою кнопкою миші. Одне натискання, одне відпускання. Втрачено більшу частину нашого часу на цей, але збираємося зробити ще два варіанти. Наступні два – те саме, але тепер кожен крок розділено на окрему функцію. Вони будуть використовуватися, коли нам потрібно утримувати мишу протягом тривалого часу (для перетягування, стрільби тощо) (див. рис. 3.5.6).

```

1  def leftDown():
2      win32api.mouse_event(win32con.MOUSEEVENTF_LEFTDOWN,0,0)
3      time.sleep(.1)
4      print 'left Down'
5
6  def leftUp():
7      win32api.mouse_event(win32con.MOUSEEVENTF_LEFTUP,0,0)
8      time.sleep(.1)
9      print 'left release'

```

Рис. 3.5.6. Функції для різних подій.

Якщо клацати в сторону, все, що залишилося, це переміщати мишу по екрану. Додамо такі функції до code.py (див. рис. 3.5.7):

```

1  def mousePos(cord):
2      win32api.SetCursorPos((x_pad + cord[0], y_pad + cord[1]))
3
4  def get_cords():
5      x,y = win32api.GetCursorPos()
6      x = x - x_pad
7      y = y - y_pad
8      print x,y

```

Рис. 3.5.7. Функції для різних подій.

Ці дві функції служать абсолютно різним цілям. Перший буде використовуватися для сценарію руху в програмі. Завдяки відмінним умовам іменування тіло функції виконує саме те, що передбачає “SetCursorPos()”. Виклик цієї функції встановлює курсор миші на координати, передані їй як кортеж “x”, “y”. Зауважте, що додано поля “x” і “y”; важливо робити це скрізь, де викликається координата. Другий — простий інструмент, який необхідно буде використовувати під час інтерактивного запуску Python. Він друкує на консолі поточне положення миші як кортеж “x”, “y”. Це значно прискорює процес навігації по меню без необхідності робити знімок і розгортати лінійку. Не завжди є змога ним користуватися, оскільки деякі дії миші потребуватимуть залежно від пікселів, але коли це можливо, це фантастична економія часу.

3.5. Інтеграція роботи модулів у програмі та тестування

Дизайн інтерфейсу програми визначений вище, бібліотека для розробки PysimpleGUI - теж. Для реалізації задуму необхідно спочатку написати всі компоненти вікна. Створимо, для зручності, ще один файл під назвою “utils”, там будуть міститися другорядні функції для роботи в програмі. Перед розробкою необхідно створити конфігураційний файл для вікна, він буде містити параметри, що застосовуватимуться при запуску програми: розмір вікна, частота оновлення вікна, та файли по замовчуванню.

Модуль з компонентами, що розміщенні на вікні, підписаний, як components.py, у файлі міститься функція “layout_all”, яка повертає у вигляді списку всі компоненти. Ієрархія списку виглядає наступним чином (див. рис. 3.6.1):


```

layout = [
    [sg.Frame(layout=header_layout, title='',
              element_justification='center', expand_x=True)],
    [sg.Frame(layout=center_layout, title='',
              element_justification='center', expand_x=True)],
    [sg.Frame(layout=footer_layout, title='',
              element_justification='center', expand_x=True)]
]

```

Рис. 3.6.1. Компоненти згруповані по фреймах.

Компонента Frame містить інші “кастомні” компоненти, як от: “header”, “footer” і “content”. Для прикладу, вікно для налаштувань виглядає наступним чином (див. рис. 3.6.2). Цей шаблон охоплює: кнопки, текстові поля, менеджер провідника та меню для вибору цільового вікна зі списку. Ключовим параметром для кожної компоненти є ключ-посилання на цей елемент, яке надалі буде використовуватися для обробки подій.

```

window_menu = ['Unused', windows_list]
header_column_left = [
    [
        sg.FolderBrowse(button_text='Target',
                        target='-TARGET FOLDER-', size=(7, 1), enable_events=True),
        sg.Input(size=(70, 1), enable_events=True,
                 readonly=True, key='-TARGET FOLDER-'),
        sg.ButtonMenu('Window', window_menu, size=(
            7, 1), key='-WINDOW TITLES-'),
        sg.Input(size=(30, 1), enable_events=True,
                 readonly=True, key='-TARGET WINDOW-')
    ]
]
header_column_right = [
    [
        sg.Button(button_text='START', enable_events=True,
                  key='-START-', disabled=True),
        sg.Button(button_text='PAUSE', enable_events=True, key='-PAUSE-'),
        sg.Button(button_text='CLEAR', enable_events=True, key='-CLEAR-')
    ]
]
header_layout = [
    [
        sg.Column(header_column_left,
                  element_justification='left', expand_x=True),
        sg.Column(header_column_right,
                  element_justification='right', expand_x=True)
    ]
]
center_column_left = [
    [
        sg.Image(**get_video_banner(image_path, image_size), size=image_size,
                 key='-VIDEO STREAM-')
    ]
]

```

Рис. 3.6.2. Компоненти згруповані по фреймах.

Запуск програми починається з головного файлу `main.py`, у якій міститься функція `main`, де запускається цикл. Для реалізації вікна необхідно створити екземпляр класу `Window` та зазначити параметри запуску. Налаштування вікна перед запуском (див. рис. 3.6.3):

```
def get_window_settings():
    layout_props = [
        get_titles(),
        CONFIG['DISCONNECT_INFO'],
        CONFIG['WINDOW_SIZE']
    ]
    main_settings = dict(
        layout=layout_all(*layout_props),
        size=(1120, 630)
    )

    icon_path = get_file_path(CONFIG['WINDOW_ICON'])
    if icon_path is not None:
        main_settings.update(icon=icon_path)

    return main_settings
```

Рис. 3.6.3. Наповнення вікна компонентами.

Окрім підвантаження компонентів на вікно, необхідно задати конфігурації, які містяться у файлі (див. рис. 3.6.4) `config.json` у директорії `static`. При закритті програми, стан вікна зберігається і до наступного запуску.

```
{
    "FPS": 60,
    "WINDOW_SIZE": [
        640,
        360
    ],
    "DISCONNECT_INFO": "disconnect_info.png",
    "WINDOW_ICON": "window_icon.ico",
    "PERMISSION_FILE": "permission.json",
    "TARGET_WINDOW": "",
    "TARGET_FOLDER": ""
}
```

Рис. 3.6.4. Конфігураційний файл.

Для обробки події, як от: натискання кнопки чи отримання зображення з інших віконих програм, необхідно створити цикл (див. рис. 3.6.5). З кожною ітерацією буде відбуватися зчитування стану вікна нашої програми. У тілі циклу необхідно перевіряти назву події, як от закриття вікна програми, тоді цю подію відловлюється в умовній конструкції. З кожною ітерацією, окрім назви події можна отримати також значення із форми. Налаштування частоти оновлення вікна задається в конфігураційному файлі.

```
while True:
    event, values = window.read(timeout=timeout)

    if event == sg.WIN_CLOSED:
        program_close()
        init_config(conf_file=CONFIG)
        break
```

Рис. 3.6.5. Цикл обробки подій.

Отже, перейдемо до обробки подій, або завантаження програми (див. рис. 3.6.6). Під час запуску вікна програми, стан вікна, який збережений у конфігураційному файлі, автоматично завантажувється, якщо такі поля на формі пусті.

```
if FIRST_LOAD:
    if values['-TARGET FOLDER-'] == '':
        targetFolder_value(window)
    if values['-TARGET WINDOW-'] == '':
        targetWindow_value(window)
    if values['-ACTIONS LOGS-'] == '':
        actions_value(window)
    if values['-SPEECH LOGS-'] == '':
        speech_value(window)
```

Рис. 3.6.6. Цикл обробки подій.

Стан головних кнопок керування програмою буде залежати від стану програми (див. рис. 3.6.7), як от: доступна кнопка чи ні, в залежності від того, чи

пройшов користувач перевірку чи ні. Окрім доступності кнопок, також реалізована їхня функція. Аналогічним чином працює обробка подій при натисканні усіх кнопок на формі.

```

if values['-TARGET WINDOW-'] != '':
    window['-PAUSE-'].update(disabled=False)
else:
    window['-PAUSE-'].update(disabled=True)

if event == '-TARGET FOLDER-':
    targetFolder_event(values)
if event == '-WINDOW TITLES-':
    targetWindow_event(window, values)

if event == '-START-':
    start_event(window)
if event == '-PAUSE-':
    pause_event(window)

```

Рис. 3.6.7. Обробка подій компонентів керування, налаштування.

Ключовим є реакція програми на активацію роботи бота, як візуального, так маніпулятивного. У програмі є запис стану: розпізнавання, готовності та реалізації. При активації цих станів відбувається виклик функцій (див. рис. 3.6.8).

```

if CHECKED and DETECTOR_READY:
    detector = init_detector(values['-TARGET FOLDER-'])
    DETECTOR_READY = False

if STREAMING:
    streaming_event(window, values, detector)
    actions_logs_event(window, values)
    speeching_logs_event(window, values)

```

Рис. 3.6.8. Активація бота.

Якщо користувач пройде перевірку файлів та налаштувань, йому буде доступна кнопка “START”, при натискання якої активується режим розпізнавання. У разі, якщо це “PAUSE”, відбудеться або зупинка усіх процесів, або їхнє відновлення.

Для крос-платформності цього додатку, необхідно компілювати програму в виконуваний файл з розширенням `exe`. Хорошим інструментом для цього є бібліотека `PyInstaller`. `PyInstaller` перевірено на Windows, Mac OS X і Linux. Але, це не крос-компілятор: аби створити програму для Windows, потрібно запустити `PyInstaller` у Windows; щоб створити програму для Linux - запустити її в Linux тощо. Зазвичай називаємо один сценарій у командному рядку. Якщо назвати більше, усі вони будуть проаналізовані й включені до вихідних даних. Однак перший названий нами сценарій надає назву для файлу специфікацій і для виконуваної теки, файлу. Його код виконується першим під час виконання. Для певних цілей можна редагувати та вміст `myscript.spec` (див. рис. 3.6.9). Після того, як це зробити, назвемо файл специфікацій для `PyInstaller` замість сценарію: “`pyinstaller myscript.spec`”.

```

1  # -*- mode: python ; coding: utf-8 -*-
2
3
4  block_cipher = None
5
6
7  a = Analysis(
8      ['main.py'],
9      pathex=[],
10     binaries=[],
11     datas=[('static/config.json', '/static/.'), ('static/disconnect_info.png', '/static/.'), ('static/window_icon.ico', '/static/.')],
12     hiddenimports=['utils', 'dashboard', 'windowcapture', 'components'],
13     hookspath=[],
14     hooksconfig={},
15     runtime_hooks=[],
16     excludes=[],
17     win_no_prefer_redirects=False,
18     win_private_assemblies=False,
19     cipher=block_cipher,
20     noarchive=False,
21 )
22 pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)
23
24 exe = EXE(
25     pyz,
26     a.scripts,
27     a.binaries,
28     a.zipfiles,
29     a.datas,
30     [],
31     name='AI Bot',
32     debug=False,
33     bootloader_ignore_signals=False,
34     strip=False,
35     upx=True,
36     upx_exclude=[],
37     runtime_tmpdir=None,
38     console=False,
39     disable_windowed_traceback=False,
40     argv_emulation=False,
41     target_arch=None,
42     codesign_identity=None,
43     entitlements_file=None,
44 )
45

```

Рис. 3.6.9. Файл специфікацій для компіляції.

Файл `myscript.spec` містить більшу частину інформації, наданої параметрами, які були вказані під час запуску `pyinstaller` (або `pyi-makespec`) із файлом сценарію, як аргументом. Зазвичай не потрібно вказувати параметри під час запуску `pyinstaller` з файлом специфікацій. Лише кілька параметрів командного рядка мають ефект під час створення з файлу специфікацій. Альтернативним рішенням замість використання специфікаційного файлу є спосіб задання параметрів у командному рядку (див. рис. 3.6.10), що і було, зокрема, використано у цьому проєкті.

```
$ convert_exe.sh
1 pyinstaller \
2   --onefile --windowed \
3   --add-data "static/config.json;/static/" --add-data "static/disconnect_info.png;/static/" --add-data "static/window_icon.ico;/static/" \
4   --hidden-import="utils" --hidden-import="dashboard" --hidden-import="windowcapture" --hidden-import="components" \
5   --name "AI Bot" \
6   main.py
```

Рис. 3.6.10. Скрипт для запуску компіляції.

Отже, у цьому проєкті можна скомпілювати код у виконавчий файл, але буде проблемою його великий розмір, як у моєму випадку це понад 500 мегабайтів. Після запуску виконавчого файлу усі статичні файли будуть зберігатися у спеціальній тимчасовій “директорії”, до таких файлів відносяться “логи” роботи бота, які користувач може зберегти під час виконання програми.

РОЗДІЛ IV. ТЕСТУВАННЯ РОЗРОБКИ. ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1. Запуск програми в тестовому середовищі

Почнімо із запуску програми в режимі розробника. Необхідно активувати віртуальне середовище, де мають бути інсталювані всі залежності, після чого запускаємо на виконання скрипт `main.py`. Відкривається вікно (див. рис. 4.1.1) програми, але пусте. Спершу необхідно базово налаштувати роботу бота, а саме приєднати цільову “директорію” (див. рис. 4.1.2) та вказати цільове вікно (див. рис. 4.1.3).

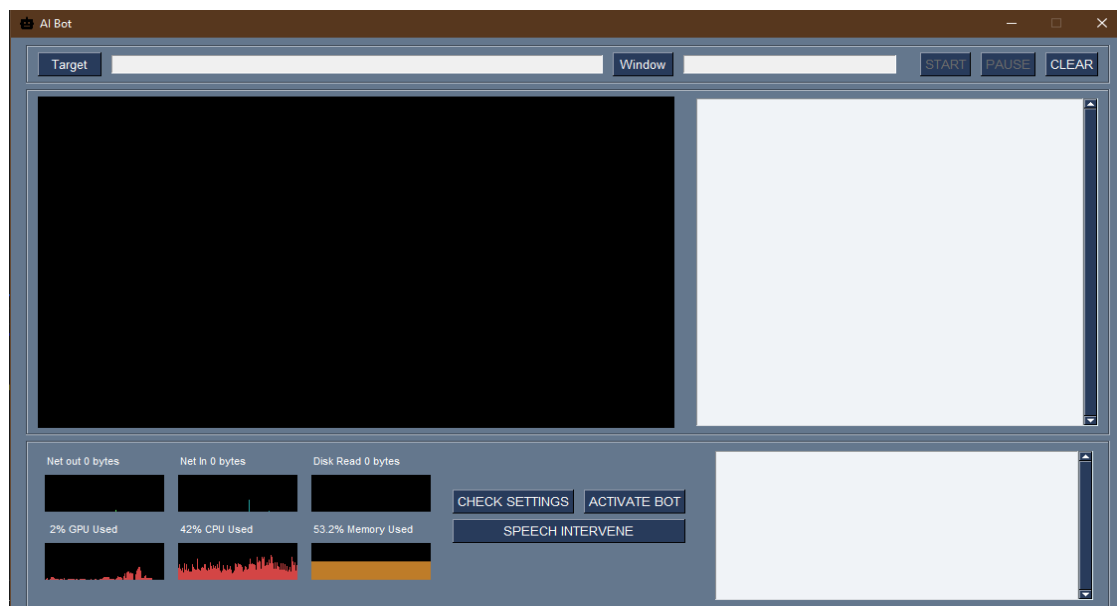


Рис. 4.1.1. Вікно програми.

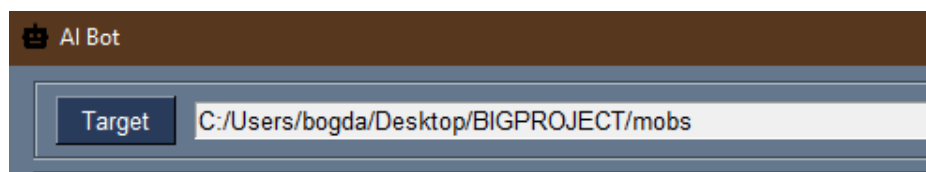


Рис. 4.1.2. Налаштування цільової директорії.

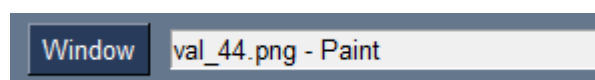


Рис. 4.1.3. Налаштування цільового вікна.

Після натискання кнопки “CHECK SETTINGS” відбувається підготовка (див. рис. 4.1.4) до роботи з технологіями розпізнавання, налаштування роботи з відеокартою. Відеокарта ідентифікується в системі, як NVIDIA GeForce GTX 1050 Ti, саме вона буде під навантаженням під час активації функції комп’ютерного зору.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER COMMENTS
2022-12-07 01:04:44.602331: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusolver64_10.dll
2022-12-07 01:04:44.602472: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cusparse64_11.dll
2022-12-07 01:04:44.602617: I tensorflow/stream_executor/platform/default/dso_loader.cc:49] Successfully opened dynamic library cudnn64_8.dll
2022-12-07 01:04:44.602792: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1862] Adding visible gpu devices: 0
2022-12-07 01:04:45.312604: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1261] Device interconnect StreamExecutor with strength 1 edge matrix:
2022-12-07 01:04:45.312958: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1267] 0
2022-12-07 01:04:45.313258: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1280] 0: N
2022-12-07 01:04:45.313583: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1406] Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 2976 MB mem
ory) -> physical CPU (device: 0, name: NVIDIA GeForce GTX 1050 Ti, pci bus id: 0000:01:00.0, compute capability: 6.1)
2022-12-07 01:04:45.317972: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x259f9d2b530 initialized for platform CUDA (this does not guarantee that XLA will
be used). Devices:
2022-12-07 01:04:45.318404: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): NVIDIA GeForce GTX 1050 Ti, Compute Capability 6.1

```

Рис. 4.1.4. Ініціалізація роботи з бібліотекою Tensorflow.

Тестове середовище буде виглядати наступним чином (див. рис. 4.1.5): відкриємо фотоекзмпляр гри у програмі Paint та проводимо досліди над різними ситуаціями.

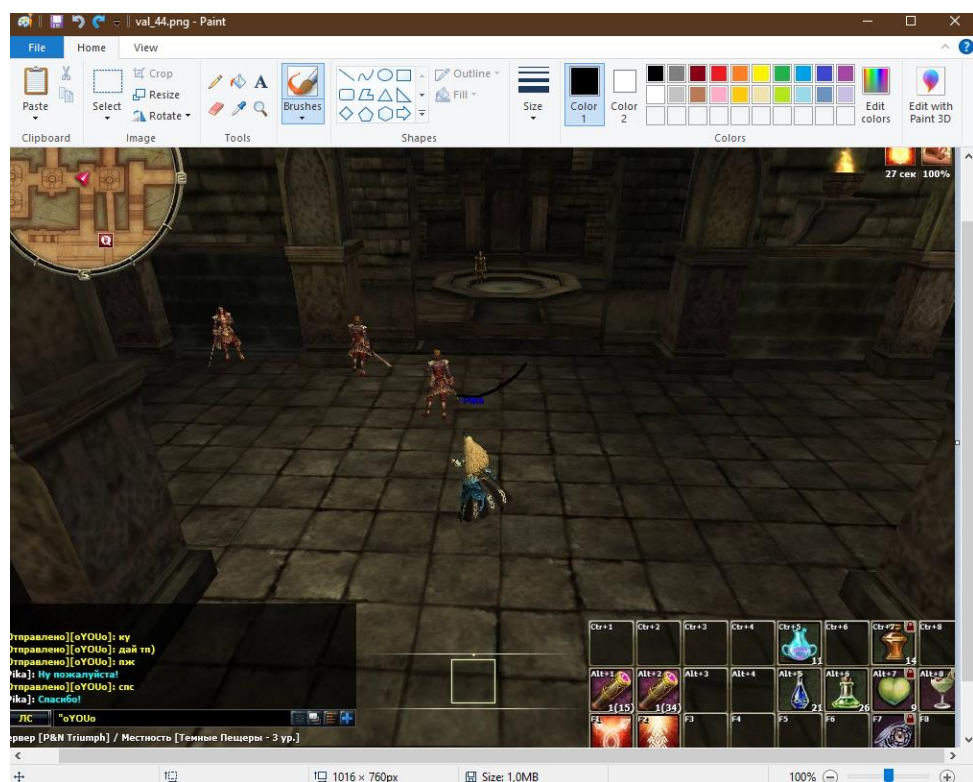


Рис. 4.1.5. Тестове вікно.

Сенс запускати саму гру поки що немає, адже потрібно оптимізувати основні процеси нашої програми. Окрім визначеного наперед середовища, яке імітує гру, потрібно перевірити також роботоздатність скомпільованого виконавчого файлу. Для компіляції необхідно запустити скрипт `convert_exe.sh`. Компіляція проходить успішно (див. рис. 4.1.6), можемо тепер поглянути на файл (див. рис. 4.1.7) та його розмір. Тепер вихідний файл можна запускати в незалежності чи налаштоване середовище в ОС Windows, чи ні. Наш бот запуститься у кожного, але є проблема з розміром файлу, це понад 400 мегабайтів, що проявляється у його довготривалому запуску та повільній передачі в мережі інтернет.

```
128679 INFO: Updating resource type 24 name 1 language 0
128684 INFO: Appending PKG archive to EXE
129110 INFO: Fixing EXE headers
134928 INFO: Building EXE from EXE-00.toc completed successfully.
```

Рис. 4.1.6. Компіляція виконуваного файлу.

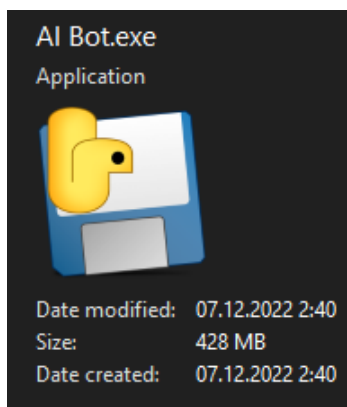


Рис. 4.1.7. Деталі виконавчого файлу.

4.2. Аналіз результатів тестування та корекція програми

Першим етапом тестування було перевірка роботи інтерфейсу, його відклик та робота з навантаженням системи ПК (без активації функцій розпізнавання). Проблема з взаємодію користувача і програми на етапі підготовки до роботи бота не виявлено. Спробуємо запустити трансляцію вікна програми Paint та після отримання змін на графіках – зупинити, порівняємо

отримані результати. Візуально отримані графіки (див. рис. 4.2.1 а, б) відрізняються, при трансляції цільового вікна програми й без. Отже, якщо порівняти обидва рисунки (див. рис. 4.2.1 а, б), тоді стає очевидним, що, в середньому, трансляція у нашу програму навантажує CPU ПК майже у 2 рази більше, аніж при її відсутності.

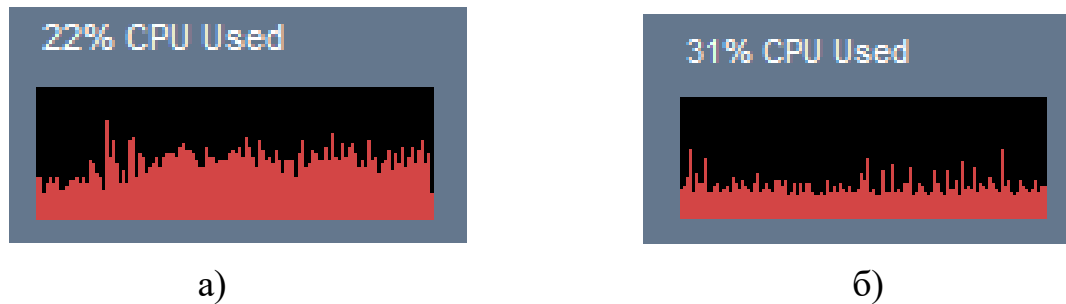


Рис. 4.2.1. Тестування навантаження CPU при трансляції вікна:

а) вікно транслюється; б) вікно не транслюється.

Спробуємо запустити режим розпізнавання, натиснувши на кнопку “START”, але при цьому попередньо ввімкнувши трансляцію. Програма відповідає через 1-2 секунди затримки, адже відбувається додаткова ініціалізація модуля. На вікні нашої програми отримуємо наступне (див. рис. 4.2.2):

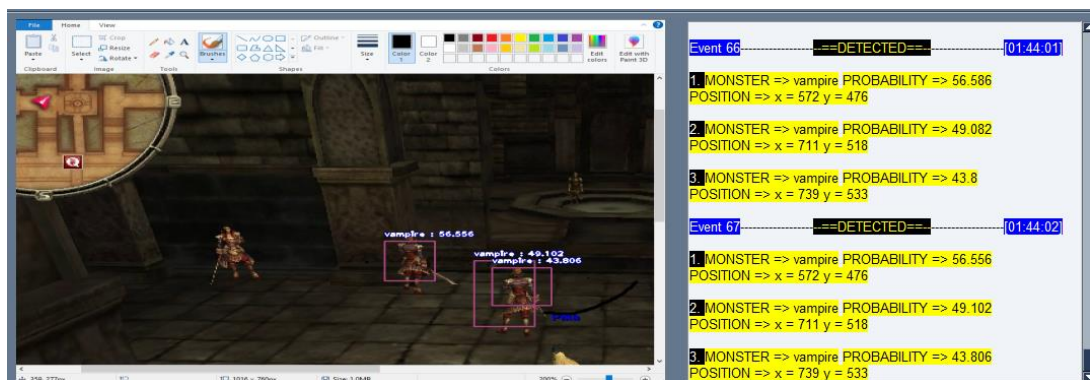


Рис. 4.2.2. Результати роботи технології розпізнавання та відомості.

Щодо результатів роботи програми: бот розпізнає цільові об'єкти із точністю близько 40%, але цього не достатньо, адже лівий об'єкт не був

захоплений у поле зору. По отриманим даним, зрозуміло, що в більшості випадків технологія дає змогу нашому боту зорієнтуватися, але для кращої успішності необхідно доопрацювати дата сет зображень, а саме збільшити його об'єм хоча б у 2 рази.

Погляньмо на систему моніторингу, а саме на показники споживання GPU (див. рис. 4.2.3). З графіка стає зрозуміло, на скільки зросло споживання відеокарти після активації функції розпізнавання, а саме, значення виросло на 35%.

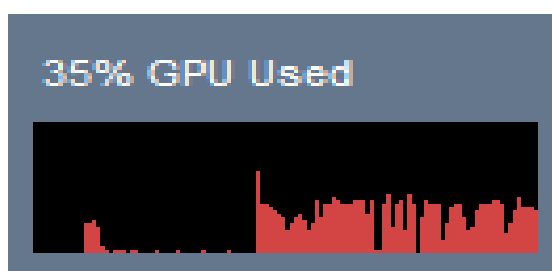


Рис. 4.2.3. Графік використання GPU.

Після штучного випробовування у тестовому середовищі, де були враховані помилки, внесено ряд змін на основі результатів із попередніх тестів. Розглянуто ряд проблем, опрацьовано інформацію, яка допоможе оптимізувати нашу програму. виправлено затримки під час активації функцій та навантаження CPU.

4.3. Збір даних із довготривалого користування в “бойовому” режимі

Отже, програма, яка була модернізована, може бути випробуваною в реальних умовах гри, а саме в R2 Online. Розпочнімо з запуску гри, де вже створений ігровий акаунт, на якому раніше проводились випробування з розробленими прототипами.

Краще запускати гру у віконному режимі з найменшим розширенням, аби наш бот працював ефективніше. З рисунка (див. рис. 4.3.1) можна помітити (у лівому верхньому куті), що вікно гри підписується, як “?”. Задум запуску гри у

віконному режимі, полягає в тому, аби мати можливість, на одному екрані, користуватися, як інтерфейсом гри, так і бота.



Рис. 4.3.1. Вікно гри.

Отже, маємо запущене вікно гри, тепер можна звернути увагу на ключові елементи:

- Персонаж (маніпулятивний об'єкт);
- Середовище (область взаємодії);
- Монстри (цільові об'єкти).

Людина керує персонажем через комп'ютерну перефрмію, як от: клавіатура та миша. Ігрок сприймає інформацію через монітор, звуки гри для бота в цьому проєкті не мають сенсу. Наша програма взаємодії з грою на нижчому рівні: отримує інформацію швидше, безпосередньо через програмну взаємодію.

Розгляньмо ключові моменти ігрового процесу, для яких розроблений бот. Програмі потрібно відтворювати дії, за які раніше відповідав ігрок, як от: пошук

(монстрів, див. рис. 4.3.2), атака цілей (див. рис. 4.3.2) та отримання нагород (предмети, які випадають з монстрів після їхньої смерті, див. рис. 4.3.3). Це не всі аспекти гри, бот відповідатиме за прийняття рішень, подібним чином, як це робить людина. Алгоритм взаємодії бота в ігровому середовищі виглядає наступним чином: аналіз області (виявлення цілей), пошук маршруту/атака цілі, отримання нагороди й продовження циклу. Програма вміє працювати із казусними ситуаціями, якщо такі трапляються.



Рис. 4.3.2. Атака цілі.



Рис. 4.3.3. Отримання нагороди.

Отже, спробуємо зафіксувати роботу цього бота. У ОС Windows запущена гра та програма, система моніторингу бота демонструє наступні графіки (див. рис. 4.3.4) використання ресурсів ПК.

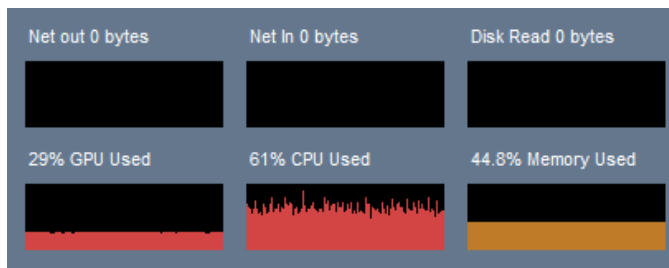


Рис. 4.3.4. Система моніторингу ПК.

Програма знаходиться в режимі простою, тоді ввімкнемо трансляцію вікна гри. Можна помітити незначне (+15%) зростання навантаження на CPU (див. рис. 4.3.5).

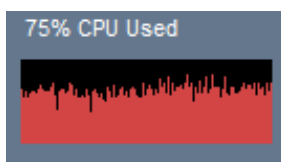


Рис. 4.3.5. Графік завантаження CPU.

Тепер необхідно увімкнути режим розпізнавання і поглянути на стан ПК. На графіках (див. рис. 4.3.6) спостерігається зміна лише у вікні GPU та RAM, стає зрозуміло, що наш ІІІ починає використовувати апаратний прискорювач для виконання роботи по розпізнанню об'єктів на кожному кадрі з трансляції.

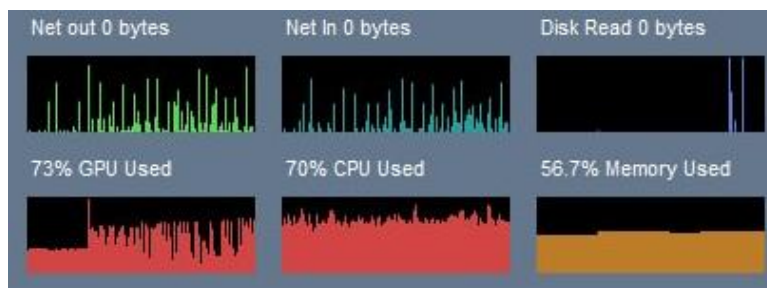


Рис. 4.3.6. Моніторинг завантаження ПК після активації функції розпізнавання.

Можна помітити, як різко збільшується навантаження GPU з 30% до 70%, а RAM з 45% до 55%. Гра завантажує ПК менше, ніж це робить бот. Результати інтегрованої функції розпізнавання можна оцінити на рисунку (див. рис. 4.3.7):



Рис. 4.3.7. Віконний інтерфейс бота.

Отже, з рисунка (див. рис. 4.3.7) зрозуміло, що бот працює самостійно: визначає ціль та атакує її. Для збирання даних розроблено скрипти, які записують дані з системи моніторингу. Все, що необхідно для дослідження – це запустити ботів на довготривалий час. Наші дослідження будуть проводитися з останніми прототипами ботів та людини, окрім цього потрібно порівняти ефективність виконання задач та навантаження системи ПК. Для збору даних, необхідно запустити скрипти Python, які записуватимуть стан системи ПК і зберігатимуть у файл з розширенням json для подальшого аналізу.

4.4. Аналіз результатів дослідження

Після довготривалого збору даних, потрібно опрацювати їх за допомогою Python скриптів. Отримані графіки можна порівняти й прийти до певних висновків, таких як: ефективність виконання задач, відповідність поставлених

задачі та оптимізація програми для роботи з ПК.

Спочатку порівняємо використання ресурсів ПК програмою та людиною. На рисунку (див. рис. 4.4.1) зображено графік навантаження CPU ПК за участю лише людини. Людина відтворює свою звичну поведінку в грі, подекуди спостерігається різкий стрибок до 90%, при тому середнє значення близьке до 55%. Для наочного прикладу, порівняймо з графіком за участю бота. На рисунку (див. рис. 4.4.2) спостерігається нестабільне використання CPU. Окрім цього помітно, що наша програма навантажує ПК в середньому на 80%.

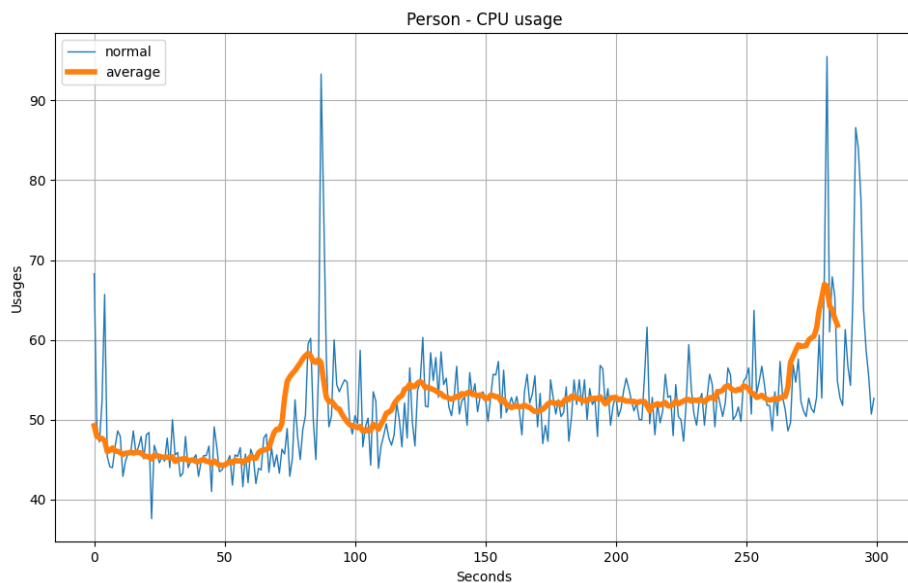


Рис. 4.4.1. Використання CPU при участі людини.

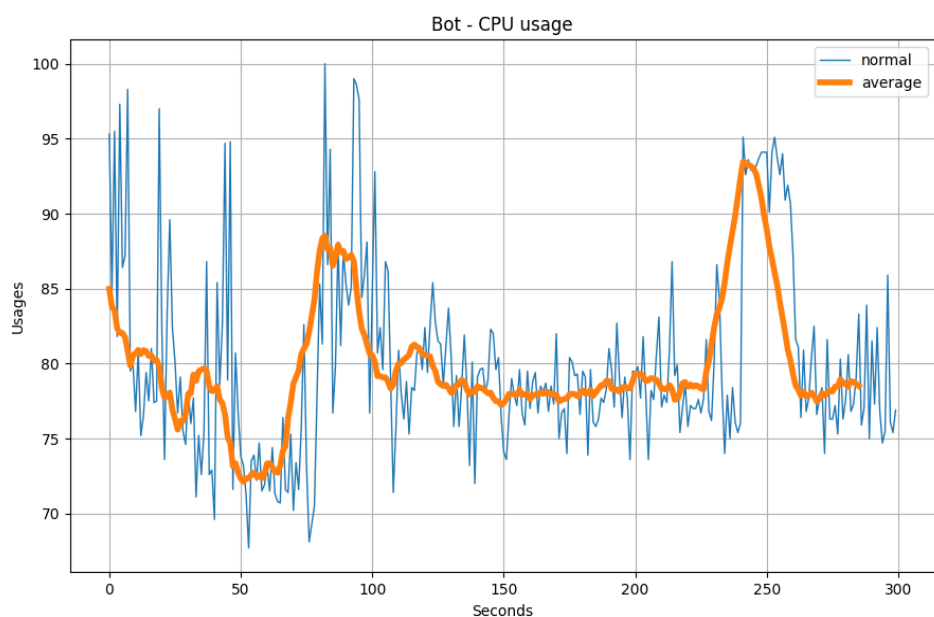


Рис. 4.4.2. Використання CPU при участі бота.

Найбільш інформативним, з результатів спостереження, є завантаження саме GPU, адже у попередньому прототипі ШІ використовував саме ресурс CPU, що довело низьку ефективність роботи програми. Як видно з графіків (див. рис. 4.4.3, рис. 4.4.4) навантаження на GPU більше у випадку з ботом, аніж з людиною; використання ресурсів GPU в середньому на 10% більше для другого випадку. Розмах завантаження GPU: людиною — |20% - 40%|, а ботом — |20%-80%|.

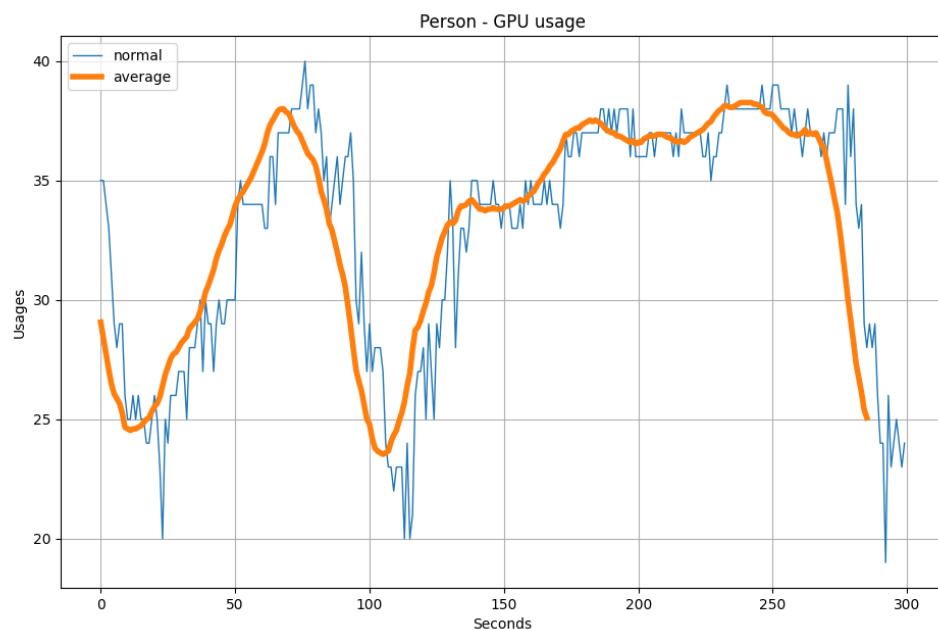


Рис. 4.4.3. Використання GPU при участі людини.

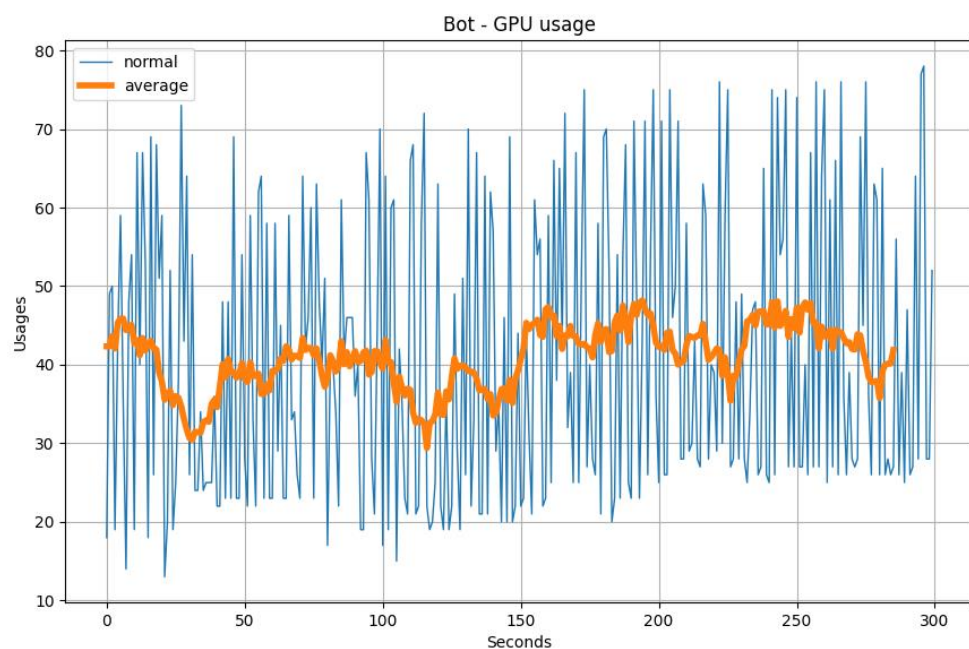


Рис. 4.4.4. Використання GPU при участі бота.

На графіках (див. рис. 4.4.5, рис. 4.4.6) зображено завантаження RAM. Можна зрозуміти, що бот не впливає на показники оперативної пам'яті ПК. Для різниці між людиною і ботом: графіки RAM показують в середньому значення у 5%. До підсумку проведених спостережень (див. табл. 4.4.7), можна сказати, що бот, використовує ресурси ПК куди більше, аніж це робить людина.

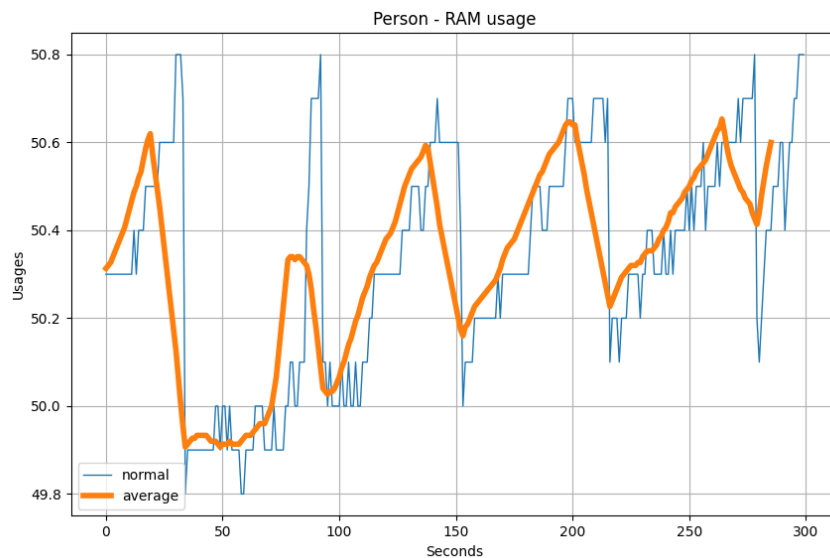


Рис. 4.4.5. Використання RAM при участі людини.

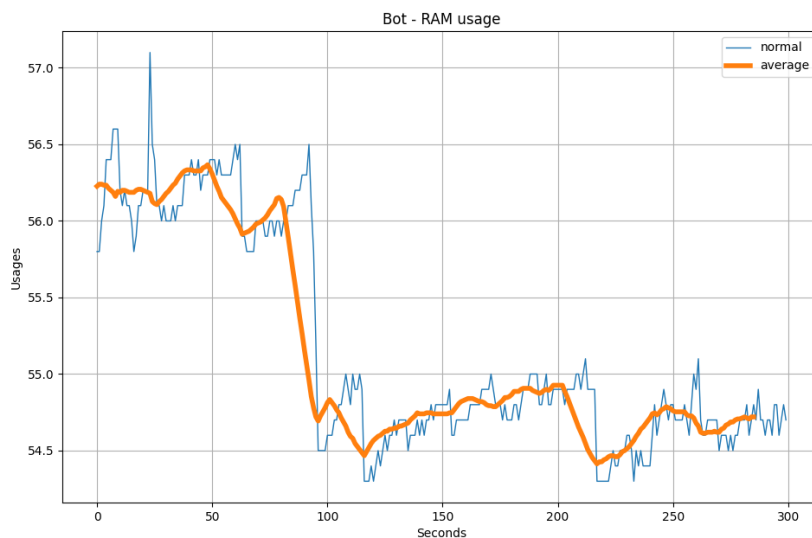


Рис. 4.4.6. Використання RAM при участі бота.

CPU		GPU		RAM	
Human	Bot	Human	Bot	Human	Bot
51.7 %	80.0 %	33.0 %	40.9 %	50.3 %	55.2 %
на 28.3 %		на 7.9		на 1.9 %	

Табл. 4.4.7. Підсумок проведених спостережень завантаження ПК.

Для розуміння повної картини, можна провести ще апроксимацію наших даних, використовуючи поліноми: 1, 2 та 3 порядку. Проведемо лінії тренду та поглянемо на графіки. На графіку завантаження CPU (див. рис. 4.4.8) спостерігається тенденція росту CPU (за участю людини), яку найкраще відображає поліном 1-го порядку, якщо орієнтуватися на точність розміщення майбутніх точок – з цим справляється апроксимація 2-го порядку. З графіку (див. рис. 4.4.9) завантаження CPU (за участю бота) випливає, що не все так однозначно, адже використання апаратного прискорювача було нестабільним, гадаю апроксимація поліномом 2-го порядку буде доречною.

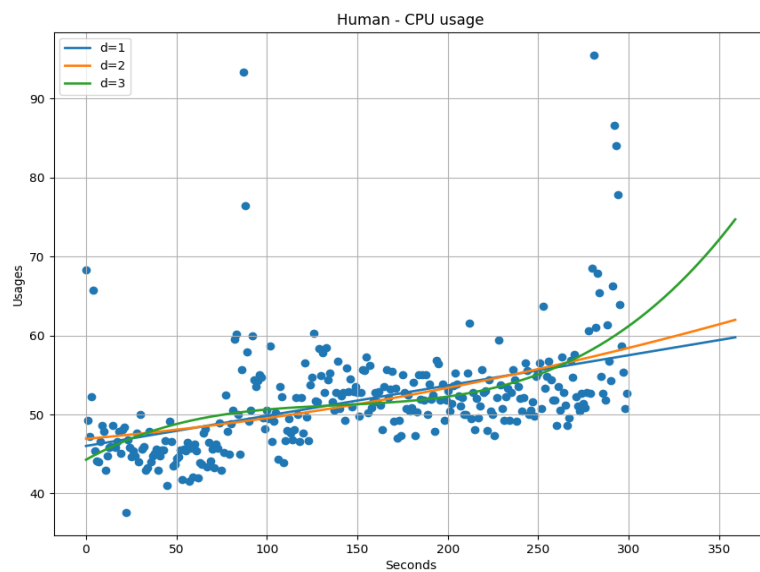


Рис. 4.4.8. Апроксимація даних навантаження CPU при участі людини.

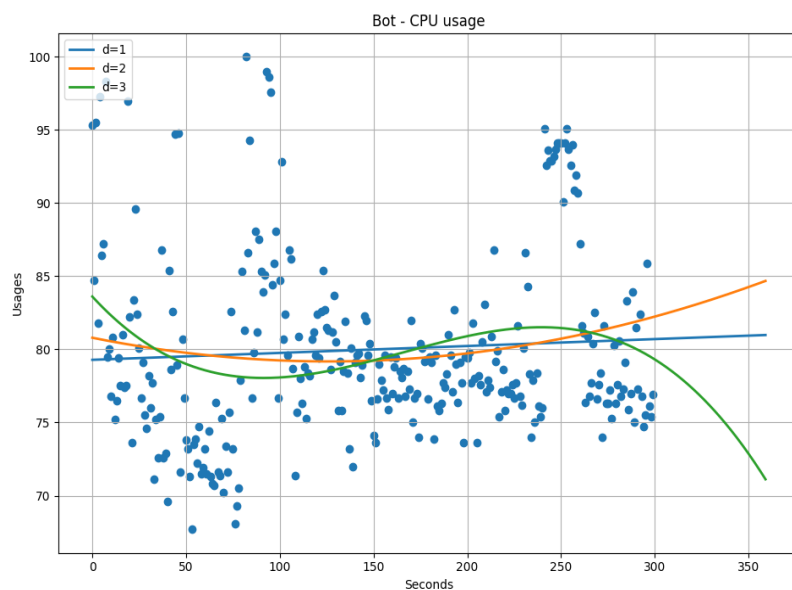


Рис. 4.4.9. Апроксимація даних навантаження CPU при участі бота.

Щодо використання GPU (див. рис. 4.4.10, рис. 4.4.11), важко сказати про тенденцію росту чи спаду завантаження ресурсу, але за участю бота, це значення буде рости.

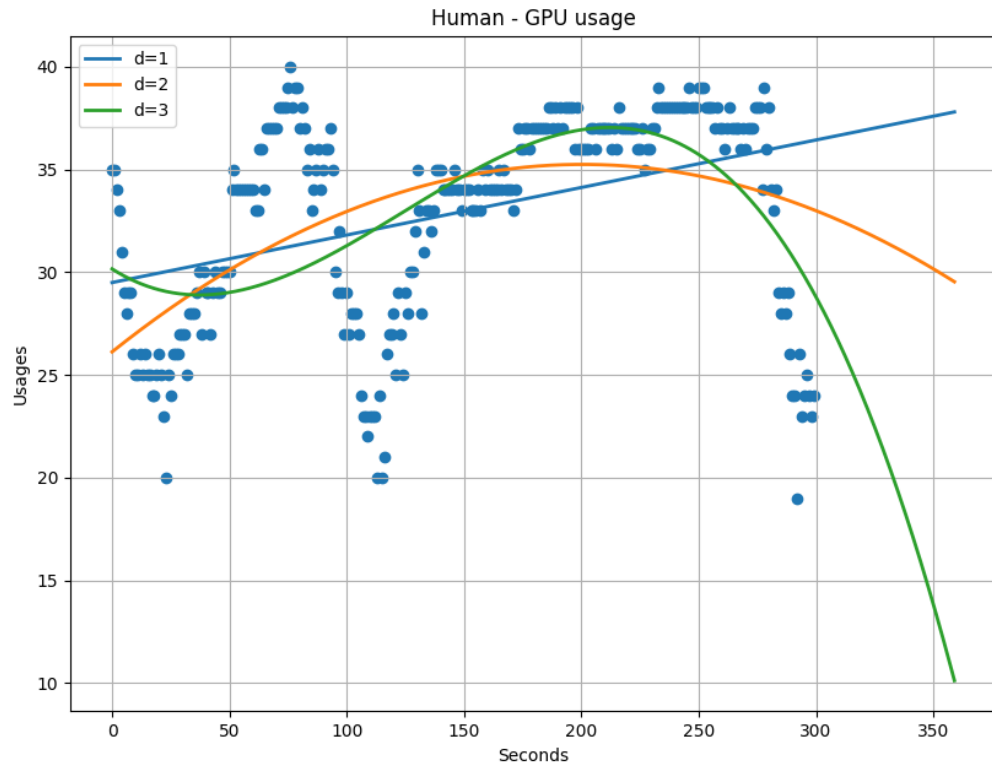


Рис. 4.4.10. Апроксимація даних навантаження GPU при участі людини.

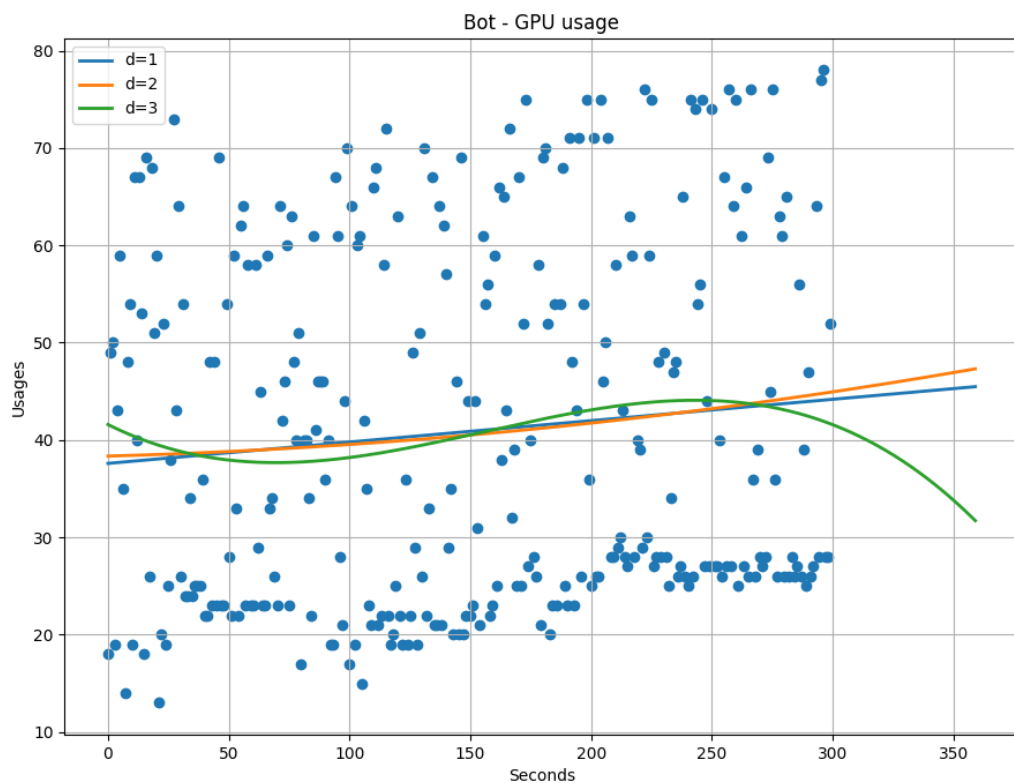


Рис. 4.4.11. Апроксимація даних навантаження GPU при участі бота.

Ось і графіки RAM (див. рис. 4.4.12, рис. 4.4.13). Тенденцію росту, за участю людини, найкраще відображає апроксимація поліномом 1-го порядку, за участю бота – 2-го порядку.

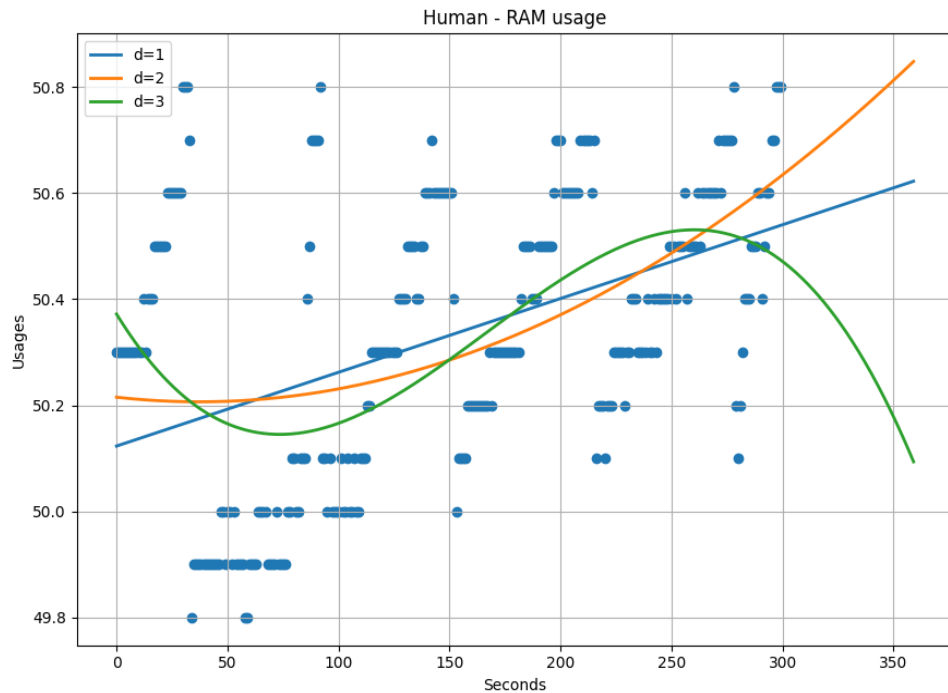


Рис. 4.4.12. Апроксимація даних навантаження RAM при участі людини.

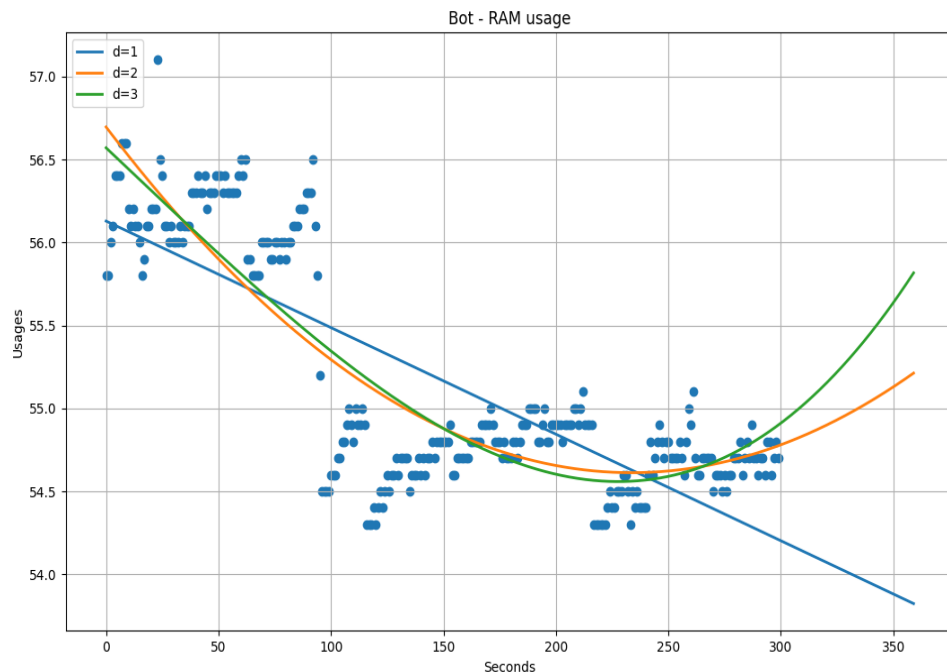


Рис. 4.4.13. Апроксимація даних навантаження RAM при участі бота.

З порівняннями використання ресурсів ПК. у випадках людини чи бота, все зрозуміло. Але як справляється бот із поставленою задачею? Отже, за попереднім

алгоритмом, який був вже описаний вище, можна схематично, на основі практичного досвіду, описати взаємодію бота з грою (див. рис 4.4.14). На цьому рисунку зрозуміло, що бот окрім успішних (виконаних), дій має і не успішні (не виконані, або недоконані). У середньому, бот справляється з однією повною ітерацією за 50 секунд, поки людина це виконує в 5 раз швидше. Основними затримками в роботі програми є довгий аналіз, наприклад, людина із розпізнаванням та фокусуванням цілі справляється в десятки раз швидше.

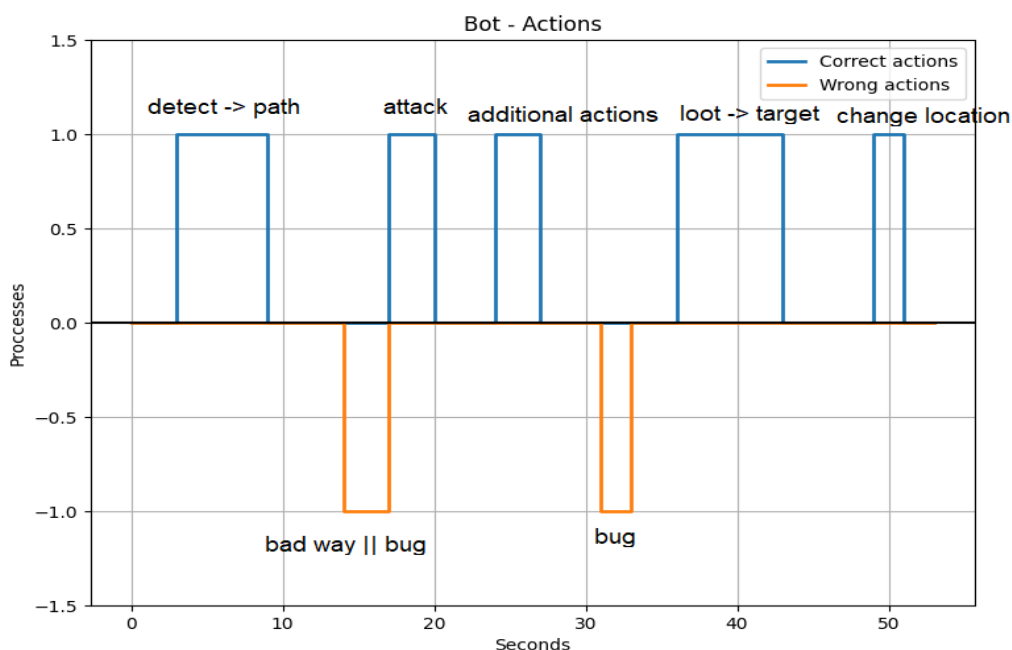


Рис. 4.4.14. Схематичне представлення поведінки бота в грі.

BOT 2			BOT 1		
CPU	GPU	RAM	CPU	GPU	RAM
80%	50%	55%	99%	25%	40%

Табл. 4.4.15. Порівняння нашого бота з прототипом, попередньої роботи.

Як зображено у таблиці (див. табл. 4.4.15), бот з цього проєкту (BOT 2) навантажує GPU на 25% більше, аніж це робить бот з попередньої розробки (BOT 1). Можна прийти до висновку, що оптимізація, яка була проведена у цьому проєкті - грамотно завантажує ресурси ПК, адже робота ШІ була перенесена з CPU на GPU, що дає змогу підняти швидкість розпізнавання, зменшити затримку програми та розвантажити CPU для виконання інших задач.

ВИСНОВКИ

У роботі розглянуто тенденції побудови віконних додатків для взаємодії користувача з програмою через графічний інтерфейс. Описано підхід до розробки архітектури програми, інтеграції функцій у віконний додаток. Оцінено шляхи оптимізації роботи ШІ, способи перенесення навантажень з одного апаратного забезпечення на інше, у системі ПК, та критерії ефективності виконання, поставлених програмі, задач.

Опрацьовано методи обробки інформації, задачі автоматизації процесів та комп'ютерну взаємодію. Оцінено технології та бібліотеки для розробки графічних інтерфейсів та міжпрограмної взаємодії. Реалізовано віконний додаток з інтеграцією таких функцій, як: обробка зображень, розпізнавання зображення та мовлення, програмна взаємодія з ОС Windows та аналіз вихідних даних програми. Створено інтерфейс для: постійного моніторингу активності та управління програмою користувачем. Для прикладу, використано бота для автоматизації ігрового процесу. Забезпечено крос-платформність застосунку, протестовано на роботоздатність функціонала.

Досліджено зміни показників навантаження системи ПК (в залежності від оптимізації), в тому числі часткове перенесення задач з CPU на GPU. Опрацьовано та проаналізовано вихідні дані, забезпечено візуальне представлення.

У підсумку, ідея роботи, протягом написання дипломної, частково не реалізована: функція розпізнавання мовлення, як засіб для втручання людини у сценарій бота, відведена в окремий модуль та не залучена. Причиною проблеми реалізації функції — нездатність програми до масштабування.

З отриманого досвіду розробки проєкту дипломної, можливо створити новий прототип, з врахуванням помилок попередніх напрацювань, що може бути залучений з метою віддаленого управління собакою-роботом, та моніторингом його стану. Перспектива комерційної розробки, для прикладу, як технологія GeForce NOW: гра запускається на сервері замість ПК користувача, так і бот:

буде запускати гру та грати у неї віддалено (на сервері), поки користувач, у режимі адміністратора, моніторитиме статистику зібраних ресурсів.

У заключенні, на основі отриманих даних, можна прийти до висновку, що програма успішно оптимізована — навантаження ефективно розподілено між ресурсами ПК (відклик/затримка програми зменшені). Отримано досвід в реалізації складних віконних додатків з інтеграцією функцій: обробки та аналізу відео/аудіоматеріалів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Detection Classes [Електронний ресурс] / Режим доступу: [www/URL:https://imageai.readthedocs.io/en/latest/](https://imageai.readthedocs.io/en/latest/) – 09.05.2019 р.
2. Prediction Classes [Електронний ресурс] / Режим доступу: [www/URL:https://imageai.readthedocs.io/en/latest/prediction/index.html](https://imageai.readthedocs.io/en/latest/prediction/index.html) - 09.05.2019 р.
3. Video and Live-Feed Detection and Analysis [Електронний ресурс] / Режим доступу URL: <https://imageai.readthedocs.io/en/latest/video/index.html> – 09.05.2019р.
4. Python [Електронний ресурс] / Режим доступу: [www/URL:https://www.python.org/](https://www.python.org/) - 09.05.2019 р.
5. Prediction Classes [Електронний ресурс] / Режим доступу: [www/URL:https://imageai.readthedocs.io/en/latest/prediction/index.html](https://imageai.readthedocs.io/en/latest/prediction/index.html) - 09.05.2019 р.
6. Kivy [Електронний ресурс] / Режим доступу URL: <https://kivy.org/#home> – 09.05.2019р.
7. Рихтер Д. Head First / Рихтер Д. – Вашингтон: 2017. – 816с.
8. Fluent Python / Лучано Рамалью. – Вашингтон: 2015. - 792с.
9. Р. Динеш. Всі патерни проектування, 2019. – 320 с.
- 10.Фрейдзон И.Р. Автоматизовані системи / И.Р. Фрейдзон. – Л.: Судостроение, 1988. – 365 с.
- 11.Крисолов В.А. представлення вихідних даних в задачах нейронного програмування / Одеса: ОНПУ. 2003. С. 7.
- 12.Леван Д.Н., Феоктистов Н.А. Особливості використання многослойного парцептрона/ Науковведення. віп. 2. 2014. С. 8.
- 13.Шитиков В.К., Розенберг Г.С., Зинченко Т.Д. Методи системної ідентифікації / Тольятти: ИЭВБ РАН. 2003. 463 с.
- 14.Шахнов В.А., Власов А.И., Кузнецов А.С. Нейрокомпюттери: схемотехніка / М.: Изд-во Машинобудування . 2000. 64 с.

15. McMillan C. The Connectionist Scientist Game: Rule Extraction and Refinement in a Neural Network / C. McMillan, M.C. Mozer, P. Smolensky // Proc. XIII Annual Conf of the Cognitive Science Society, Hillsdale, NJ, USA. – 2001
16. Область застосування штучних нейронних мереж [Електронний ресурс]//Основні напрямки використання / URL: <http://www.neuropro.ru/> (дата звернення 20.11.2020).
17. Philipp Lottes, Cyrill Stachniss; Semi-Supervised Online Visual Crop and Weed Classification in Precision Farming Exploiting Plant Arrangement. [Електронний ресурс] — Режим доступу: http://flourishproject.eu/fileadmin/user_upload/publications/lottes17iros.pdf.
18. Yao Wang ; Yisong Chen ; Peng Lu ; Heng Wang Sobel Heuristic Kernel for Aerial Semantic Segmentation// 2018 25th IEEE International Conference on Image Processing (ICIP), Electronic ISSN: 2381-8549.
19. The 2016 Sugar Beets Dataset Recorded at Campus Klein Altendorf in Bonn, Germany [Електронний ресурс] — Режим доступу: <http://www.ipb.unibonn.de/data/sugabeets2016>.
20. McMillan C. The Connectionist Scientist Game: Rule Extraction and Refinement in a Neural Network / C. McMillan, M.C. Mozer, P. Smolensky // Proc. XIII Annual Conf of the Cognitive Science Society, Hillsdale, NJ, USA. – 2001.
21. Область застосування штучних нейронних мереж [Електронний ресурс]//Основні напрямки використання / URL: <http://www.neuropro.ru/> (дата звернення 20.11.2020).