**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

**АНОТАЦІЯ**

Розроблено віконний додаток з інтеграцією таких функцій, як: обробка зображень, розпізнавання об`єктів, програмна взаємодія з операційною системою, та аналіз вихідних даних. Створено інтерфейс для: постійного моніторингу активності та управління програмою Для прикладу, 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 output data. An interface was created for: constant activity monitoring and program management. For example, a bot was used to automate the game process.

Researched the сhanges 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  
ПК Персональний комп’ютер  
АП Автоматизація процесу  
ТЗ Технічні засоби  
ПЗ Програмне забезпечення  
ОС Операційна система  
CPU Центральний процесор  
GPU Дискретний графічний акселиратор  
RAM Оперативна пам`ять  
NET Інтернет мережа

**ЗМІСТ**

[ВСТУП 5](#_Toc121495426)

[РОЗДІЛ I. ОБРОБКА ІНФОРМАЦІЇ. АВТОМАТИЗАЦІЯ ПРОЦЕСІВ. РОЗРОБКА ПРОГРАМНИХ РІШЕНЬ 7](#_Toc121495427)

[1.1. Методи обробки інформації 7](#_Toc121495428)

[1.2. Задачі автоматизації 11](#_Toc121495429)

[1.3. Комп’ютерна система взаємодій 14](#_Toc121495430)

[РОЗДІЛ II. ТЕХНОЛОГІЇ ПРОГРАМНОЇ РОЗРОБКИ ТА АНАЛІЗУ. СИСТЕМА ПРИЙНЯТТЯ РІШЕНЬ 20](#_Toc121495431)

[2.1. Постановка задачі та архітектура рішень 20](#_Toc121495432)

[2.2. Розробка віконних програм 22](#_Toc121495433)

[2.3. Програмна взаємодія з операційної системою 32](#_Toc121495434)

[РОЗДІЛ III. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. РЕАЛІЗАЦІЯ ПРОЄКТУ 39](#_Toc121495435)

[3.1. Створення прототипів та тестування 39](#_Toc121495436)

[3.2. Реалізація віконної програми з майбутнім функціоналом 40](#_Toc121495437)

[3.3. Написання модулів для роботи комп’ютерного зору та розпізнавання мовлення 45](#_Toc121495438)

[3.4. Реалізація програмної взаємодії з грою 50](#_Toc121495439)

[3.5. Інтеграція роботи модулів у програмі та тестування 55](#_Toc121495440)

[РОЗДІЛ IV. ТЕСТУВАННЯ РОЗРОБКИ. ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ 62](#_Toc121495441)

[4.1. Запуск програми в тестовому середовищі 62](#_Toc121495442)

[4.2. Аналіз результатів тестування та корекція програми 64](#_Toc121495443)

[4.3. Збір даних із довготривалого користування в “бойовому” режимі 66](#_Toc121495444)

[4.4. Аналіз результатів дослідження 70](#_Toc121495445)

[ВИСНОВКИ 78](#_Toc121495446)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 79](#_Toc121495447)

[ДОДАТОК А 81](#_Toc121495448)

[ДОДАТОК Б 104](#_Toc121495449)

**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 слідують одному статичному сценарію.

Створення систем людино-машинного інтерфейсу тісно пов'язане з ергономікою, але не тотожно їй. Проєктування 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:** розрозбка і дослідження оптимізованого, ефективного бота для спрощення взаємодії людини і програми з перспиктивами його модернізації.

**Мета:** реалізація віконного додатку 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 зв 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. РОЗРОБКА ПРОГРАМНИХ РІШЕНЬ

* 1. **Методи обробки інформації**

Універсальність сучасних інформаційних систем випливає 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 популярним з кінця 1970-х років. Її основною перевагою перед ієрархічною та мережевою структурами є здатність обробляти непередбачені зв’язки даних без покажчиків. 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 покажчиків. 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.

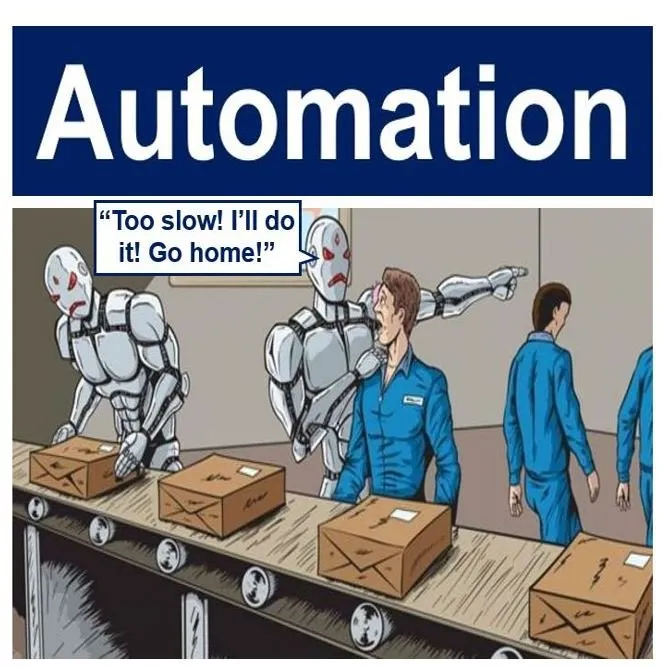
Щоб люди могли сприймати та розуміти інформацію, вона має бути представлена у вигляді друку та зображення на папері; як друк і зображення на плівці або на відеотерміналі; як звук через радіо чи телефонний зв'язок; як друк, звук і відео в кінофільмах, телевізійних передачах або на лекціях і конференціях; 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. **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 визначать 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;
* Гіперавтоматізація — 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 a 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. **Комп’ютерна система взаємодій**

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. Перше відоме використання було в 1975 році Карлайлом.[1] 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 символьними 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 HCI є задоволеність користувачів (або задоволеність кінцевого користувача комп’ютером).

Завдяки міждисциплінарному характеру HCI 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 (HCI);
* На основі аудіо: Взаємодія на основі аудіо між комп’ютером і людиною є ще однією важливою областю систем HCI. 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]

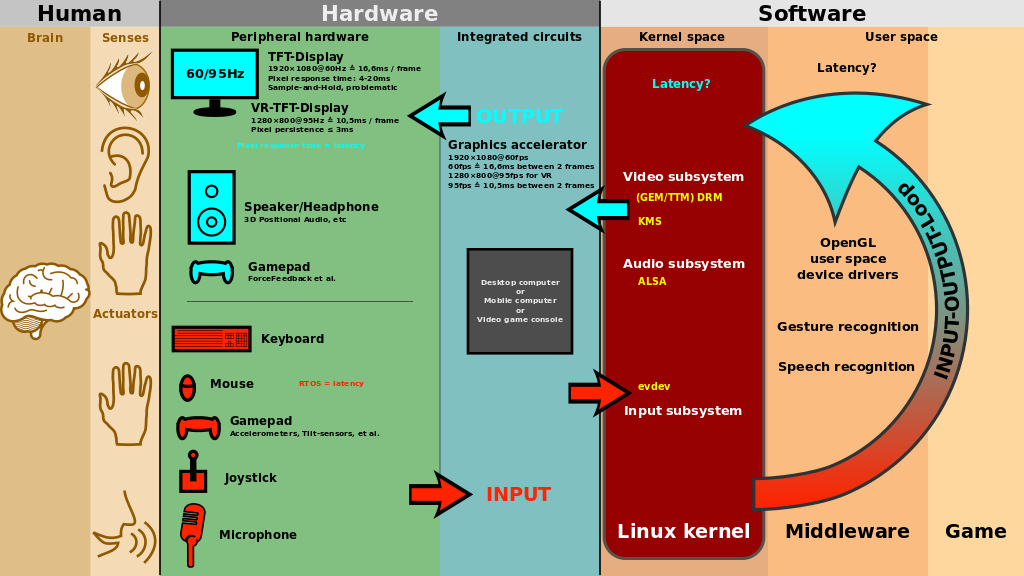
Значна частина досліджень у сфері взаємодії людини та комп’ютера спрямована на:

* 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 відрізнятися. 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 відрізнятися 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 користувача із обладненням.

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

# РОЗДІЛ II. ТЕХНОЛОГІЇ ПРОГРАМНОЇ РОЗРОБКИ ТА АНАЛІЗУ. СИСТЕМА ПРИЙНЯТТЯ РІШЕНЬ

* 1. **Постановка задачі та архітектура рішень**

З попередніх розробок були отримані значні результати для подальшого розвитку в дослідженні та розробці ПЗ в області комп’ютерного зору та автоматизації. Із отриманих результатів попередньої роботи, зрозуміло, що проектування відіграє значну роль у реалізації вихідної ідеї, грамотний підхід до розробки забузпечує якість ПЗ та ефективність його роботи з реальними задачами.

Ідея полягала у тому, аби оптимізувати ігровий процес, покращити ефективність виконаної роботи за той самий відрізок часу. Оцінювались різні підходи до реалізації, все залежало від знань, та інстурментів для розробки. Проект являв собою еволюцію ПЗ та результатів, на кожному етапі оціюнювалась відповідність поставленої задачі та отриманих результатів, були проведений аналіз та враховувались помилки, що і мотивувало займатися розробкою і дальше.

І так перехід від примітивного автоклікера, програмии, яка являла собою звичаний скрипт, за допомогою якого можна було автоматизувати якісь маленькі рутинні задачі гри., до повноцінної програми, яка в достатній мірі взаємозаміняла людинув ігровому процес Але цього було недостатньо, таким чином відбувався перехід із одного прототипу до іншого, цей процес розробки розвивався на основі попередньо отриманих результатів.

Проект мав підстави розвиватися і надалі, адже програма без взаємодії з людиною не могла бути користувацькою, так як не було інтерфейсної частини. Ідеєю цієї роботи було розвиток з вже розробленого проекту, без дописування (“латання дир”) поперенього коду, а розробки нового, ефективнішого, як по використанням ресурсів ПК так і по виконанні поставлених задач. Із отриманих результатів попередньої роботи, зрозуміло, що проектування відіграє значну роль.

Поставновка задачі виглядає наступним чином:

* Зібрати дані із результатів розробки попереднього проекту;
* Врахувати всі помилки останнього проекту;
* Віднайти в інтернеті кращі інструменти, або їхні аналоги;
* Поглиблене вивчення бібліотек та фреймворків;
* Дізнатися, як працювати з WinAPI для взаємодії з ОС Windows;
* Робота із відеопотоками та вікнами в ОС Windows;
* Розробити прототипи для отримання перших результатів;
* Обєднати весь функціонал та протестувати роботу програми;
* Розробити архітектуру програми для ефективного використання ресурсів ПК;
* Провести перші дослідження на основі результатів із гри;
* Дійти до висновку, який випливатиме із роботи алгоритму.

Люди у світі програмного забезпечення давно сперечалися щодо визначення архітектури. Для деяких це щось на зразок фундаментальної організації системи або способу з’єднання компонентів найвищого рівня. Моя думка з цього приводу була сформована обміном електронною поштою з Ральфом Джонсоном, який поставив під сумнів цю фразу, стверджуючи, що не існує об’єктивного способу визначити, що є фундаментальним або високим рівнем, і що кращий погляд на архітектуру є спільним розумінням того, що експертні розробники мати дизайн системи. Архітектура є складною темою для клієнтів і користувачів програмних продуктів, оскільки це не те, що вони сприймають відразу. Але погана архітектура є основним фактором зростання крафту – елементів програмного забезпечення, які заважають розробникам зрозуміти програмне забезпечення.

Архітектура ПЗ це не місце, це потрібно писати красиво, головною задачею, є рішення, яка зробить програму ефективною в плані використанянм ресурсів ПК, та/або зробить код маштабованим. Архітектурне рішення також можна розвивати, для цього слід, можливо і написати нову наукову роботу. У цьому проекті архітектурне рішення може примітивним та не дуже корисним. Головною ідеєю є розробка та реалізація.

Архітектура прокту буде виглядати наступним чином:

* Модуль програми:
  + Розробка моделів, які будуть відповідати за конкретний функціонал;
  + Обєднання всіх функціональних модулів в одному (точка входу);
  + Наявність скриптів для тестування модулів.
* Модуль з даними для нейромережі (модель, датасет, конфігурації);
* Модуль для статичних файлів;
* Модуль для скриптів запуску.
  1. **Розробка віконних програм**

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

Ваш код PySimpleGUI є простішим і коротшим, ніж написання безпосередньо з використанням основного фреймворку, оскільки PySimpleGUI реалізує для вас 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 ви працюєте поверх, за потреби ви можете отримати прямий доступ до залежних від фреймворків віджетів і вікон. Якщо параметр або функція ще не представлені або доступні за допомогою API PySimpleGUI, ви не відгороджені від фреймворку. Ви можете 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. Ви можете використовувати Python і пакет PySimpleGUI для створення привабливих інтерфейсів користувача, які сподобаються вам і вашим 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

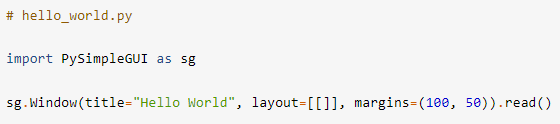
PySimpleGUI обгортає частини кожного з цих пакетів і полегшує їх використання. Однак кожен із портів потрібно встановлювати окремо.

PySimpleGUI огортає весь Tkinter, який постачається з Python. PySimpleGUI містить більшу частину PySide2, але лише невелику частину wxPython. Коли ви встановлюєте PySimpleGUI, ви отримуєте варіант Tkinter за замовчуванням. Щоб дізнатися більше про Tkinter, ознайомтеся з програмуванням Python GUI за допомогою Tkinter.

Залежно від того, який варіант PySimpleGUI ви використовуєте, програми, які ви створюєте 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 ви коли-небудь раніше користувалися набором інструментів GUI, можливо, ви чули 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(), ви можете зробити наступне (див. рис. 2.2.1):



**Рис. 2.2.1.** Старт першої віконної програми.

Window() приймає багато різних аргументів — занадто багато, щоб їх тут перелічувати. Однак для цього прикладу ви можете 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. Ви дізнаєтеся більше про це в наступних розділах цього посібника. Коли ви запускаєте цей код, ви повинні 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. Демоверсії, які входять до пакету, є великими. Ви можете 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. Ви можете використовувати цей код для перегляду власних фотографій або включити його для перегляду фотографій, які ви завантажуєте або читаєте з бази даних. Щоб спростити все, ви скористаєтеся вбудованим елементом Image() PySimpleGUI для перегляду зображень. На жаль, елемент Image() може відображати лише формати PNG і GIF у звичайній версії PySimpleGUI. Якщо ви хочете мати можливість відкривати файли зображень інших типів, ви можете завантажити Pillow, який підтримує формати TIFF, JPG і BMP. Перегляньте папку демонстрації PySimpleGUI на GitHub, щоб отримати приклад, який показує, як це зробити.

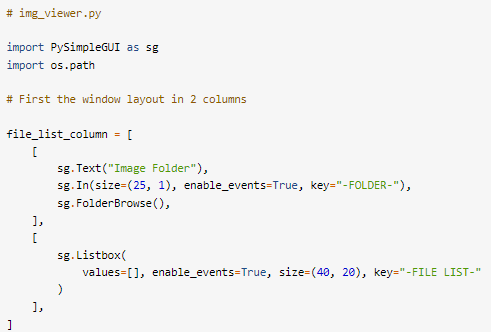
З іншого боку, якщо ви встановите порт PySimpleGUIQt, ви побачите, що Qt підтримує більше форматів зображень із коробки, ніж Tkinter. Ось макет того, як має виглядати переглядач зображень у кінці (див. рис. 2.2.3):



**Рис. 2.2.3.** Макет вікна.

Для цього прикладу буде багато коду, але не хвилюйтеся. Пізніше ви будете розглядати це меншими шматками.

Ви можете створити файл з іменем img\_viewer.py у своєму редакторі Python. Потім додайте такий код (див. рис. 2.2.4):



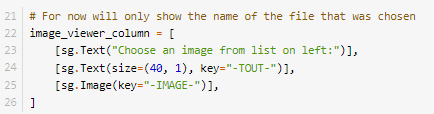
**Рис. 2.2.4.** Компоненти вікна.

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

Ключовий параметр важливий. Це те, що ви використовуєте для ідентифікації конкретного елемента у вашому GUI. Для елемента керування введенням тексту In() ви надаєте йому ідентифікатор "-FOLDER-". Ви використаєте це пізніше для доступу до вмісту елемента. Ви можете вмикати або вимикати події для кожного елемента за допомогою параметра enable\_events.

Елемент Listbox() відобразить список шляхів до зображень, які потім можна вибрати для відображення. Ви можете попередньо заповнити Listbox() значеннями, передавши список рядків.

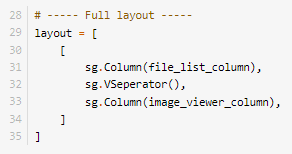
Коли ви вперше завантажуєте свій інтерфейс користувача, ви хочете, щоб Listbox() був порожнім, тому ви передаєте йому порожній список. Ви вмикаєте події для цього елемента, установлюєте його розмір і надаєте йому унікальний ідентифікатор, як ви робили з елементом input. Тепер ви можете подивитися на праву колонку елементів (див. рис. 2.2.5):



**Рис. 2.2.5.** Макет вікна.

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

Для отримання додаткової інформації про елемент Image() перегляньте документацію. Наступний фрагмент коду визначає ваш макет (див. рис. 2.2.6):



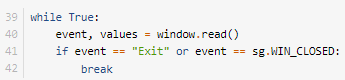
**Рис. 2.2.6.** Приклад компонування.

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(). Ви можете дізнатися більше про те, як працюють Column() і VSeperator(), прочитавши відповідні сторінки документації. Щоб додати макет у своє вікно, ви можете зробити це (див. рис. 2.1.7):



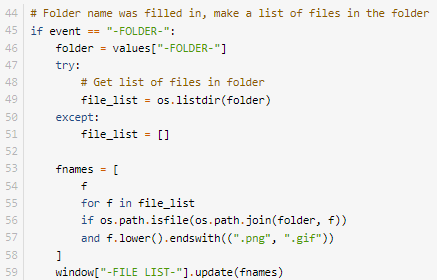
**Рис. 2.2.7.** Компонування для вікна.

Тепер, коли ви розібралися з інтерфейсом користувача, ви можете подивитися на код циклу подій. Ось перший твір (див. рис. 2.2.8):



**Рис. 2.2.8.** Цикл подій.

Цикл подій містить логіку вашої програми. Тут ви витягуєте 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.



**Рис. 2.2.9.** Цикл подій.

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

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



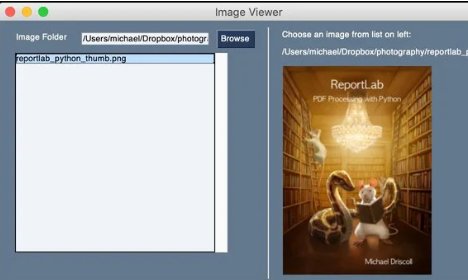
**Рис. 2.2.10.** Умовний вираз.

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



**Рис. 2.2.11.** Умовний вираз.

Коли користувач натискає кнопку Вихід, програма має закритися. Для цього можна скористатися window.close(). Технічно ви можете залишити цей рядок у своєму коді, і Python все одно завершить програму, але завжди варто прибирати за собою. Крім того, якщо ви використовуєте веб-порт PySimpleGUI і не закриваєте вікно належним чином, ви залишите порт відкритим. Тепер запустіть код, і ви повинні побачити такий інтерфейс (див. рис. 2.2.12):



**Рис. 2.2.12.** Результат віконної програми.

Ви можете скористатися кнопкою «Огляд», щоб знайти на своєму комп’ютері папку із зображеннями, щоб спробувати цей код. Або ви можете 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):



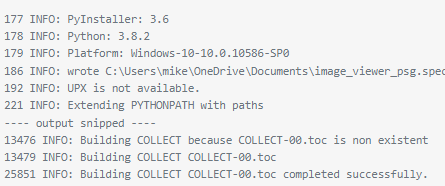
**Рис. 2.2.13.** Віконна програма для захоплення відеопотоку з камери.

Ось багато різних пакетів Python, які можна використовувати для перетворення коду Python у виконуваний файл для Windows. Одним з найпопулярніших є PyInstaller. Ви використовуватимете PyInstaller, щоб перетворити програму перегляду зображень, яку ви створили раніше, у виконуваний файл (див. рис. 2.2.14).



**Рис. 2.2.14.** Запуск компіляції.

Коли ви запустите цю команду, ви побачите багато результатів, подібних до наступного:



**Рис. 2.2.15.** Інформація про виконання команди.

Цей висновок є скороченим, оскільки PyInstaller досить багатослівний. Коли це буде зроблено, ви матимете підпапку під назвою dist у тій же папці, яка містить img\_viewer.py. Ви можете зайти в папку dist, щоб знайти img\_viewer.exe і спробувати запустити його. У папці dist буде багато інших файлів, які використовує виконуваний файл. Якщо ви бажаєте мати лише один виконуваний файл, ви можете повторно запустити команду за допомогою прапорця –onefile (див. рис. 2.2.16):



**Рис. 2.2.16.** Опціональний запуск.

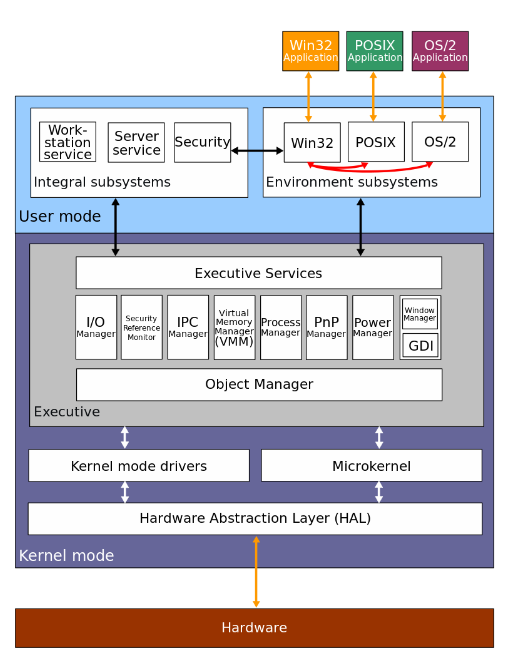
* 1. **Програмна взаємодія з операційної системою**

Windows API — це велика, складна тема, за якою стоять десятиліття історії розробки та дизайну. Хоча це надто багато, щоб охопити його в одному розділі, 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». Прекрасним 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…).

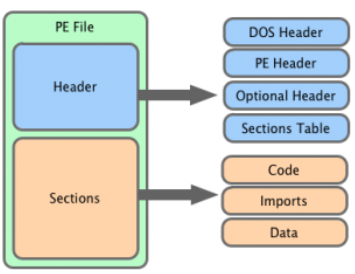
Не всі DLL створюються однаково. На додаток до стандартних DLL, які складають Windows API, існує набір DLL, які складають Native API. Нативний API працює так само, як Windows API, але є кілька ключових відмінностей. Власний API розроблено для підтримки меншого набору основних функцій до завантаження 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).



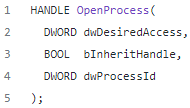
**Рис. 2.3.2.** Блок-схема архітектури Windows 2000.

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.

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.

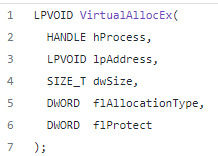
Усі функції, які можна імпортувати, мають бути десь визначені; у файлах, які визначають ці експортовані функції. Експорт функцій — це просто спосіб зробити функціональні можливості цих функцій доступними для використання іншими файлами. На фундаментальному рівні Windows API — це велика колекція файлів,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



**Рис. 2.3.3.** OpenProcess.

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

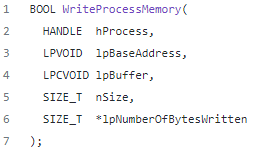
* dwDesiredAccess — визначає доступ, який буде призначено повернутому дескриптору процесу. Значення цього параметра можна знайти тут;
* bInheritHandle — дозволяє всім підпроцесам, створеним цим процесом, використовувати один дескриптор;
* dwProcessId — ідентифікатор процесу (PID) запущеного процесу.



**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:

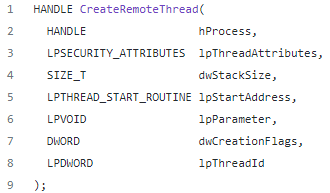
* hProcess — дескриптор запущеного процесу (отримано з OpenProcess);
* lpAddress — вказівник на бажану початкову адресу для виділеної пам’яті;
* dwSize — обсяг пам'яті, який ми хочемо виділити;
* flAllocationType — визначає тип пам’яті, яку ми будемо виділяти;
* flProtect — дозволи, які буде встановлено для пам’яті, яку ми виділяємо. Типи дозволів, які можна встановити, можна знайти тут.



**Рис. 2.3.5.** WriteProcessMemory.

Функція WriteProcessMemory (див. рис. 2.3.5) фактично записуватиме дані в простір пам’яті, який ми виділили 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 ми будемо записувати наші дані;
* lpBuffer — покажчик на дані, які ми будемо записувати у виділену пам’ять;
* nSize — кількість байтів із нашого буфера, які ми запишемо у виділену пам’ять;
* lpNumberofBytesWritten — вказівник на змінну, яка зберігатиме кількість байтів, які були записані у виділену пам’ять.



**Рис. 2.3.6.** CreateRemoteThread.

Функція CreateRemoteThread (див. рис. 2.3.6) дає можливість запускати потік у пам’яті запущеного процесу. Для цього потрібно сім аргументів:

* hProcess — дескриптор процесу (отримано з OpenProcess);
* lpThreadAttributes — вказівник на структуру, що визначає дескриптор безпеки та визначає, чи будуть дочірні процеси успадковувати дескриптор;
* dwStackSize — початковий розмір стека в байтах;
* lpStartAddress — покажчик на функцію зворотного виклику під назвою ThreadProc, яка існує у віддаленому процесі та служить початковою адресою створеного потоку;
* lpParameter — дані, що передаються до функції ThreadProc, визначеної у віддаленому процесі;
* dwCreationFlags — визначає спосіб створення потоку;
* lpThreadId — покажчик на змінну, яка зберігатиме значення, що ідентифікує створений потік.

# РОЗДІЛ III. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. РЕАЛІЗАЦІЯ ПРОЄКТУ

* 1. **Створення прототипів та тестування**

У попередній роботі було декілька етапів у розробці, підходах. Все розвивалось від примітивного скрипта, який відповідав конкретній задачі, але завершилось складною програмою із достатньою ефективністю виконанню поставлених задач із застосуванням модулів для розпізнавання.

Таким чином отримуємо модуль, який підходитиме нашому проекту. Перша програма справляється із потоковим відео, та успішно аналізує вміст на кожному із кадрів. Наш модуль відведений для робота з ШІ містить датасет, який сформований ще раніше. Цей датасет зображень містить і опис самих зображень (анотації), які атоматично створюється за допомогою програми LabelImg. Модель натренована розпізнавати лише один клас об’єктів, тому необхідно в подальшому навчити наш ШІ також і певному наборі інших об’єктів.

Такий підхід доволі не користувацький, отже необхідно задіяти, як графічний інтерфейс так і побудувати архітектуру проєкту, таким чином, аби програми не завантажували ПК та ефективно (з малою затримкою) аналізували дані, які їм подаються на вхід.

Для автоматизації усього процесу необхідно буде поєднати в одному усі модулі, які відповідають за конкретну функцію, але до їхньої повноцінної розробки ще далеко. Натомість можна віднайти в інтернет туторіали, для швидкого старту. Таким чином можна написати та випробувати програму, яка включатиме у себе: роботу з вікнами ОС Windows, пошуком та захоплення відео потоку вікна, аналізом вмісту зображень та виводом інформації.

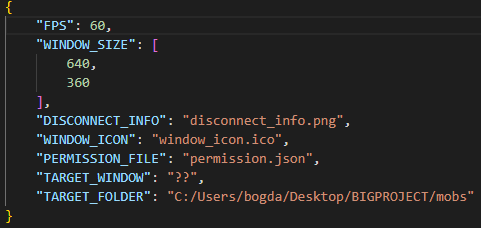
Для написання графічного інтерфейсу було використано бібліотеку PysimpleGUI вона дає можливість взаємодіяти з програмою через вікно. У циклі подій, ми будемо перехоплювати ті події, які трапляються при, як от: натисканні кнопки, вводу тексту, вибору файла та іншого, що відбувається у вікні програми. Але натомість реалізуємо кнопку вибору вікна та його захоплення.

Наступним етапом буде підключення якогось функціоналу, якщо користувач буде взаємодіяти із компонентами вікна. При натисканні на кнопку, вибору файла для завантаження модуля з конфігураціями ШІ, записується шлях до директорії із всіма налаштуваннями. Користувач запускає відео трансляцію вибраного вікна у Windows та може переглядати її вміст у вікні нашої програми. Натомість перший етап у створенні такого “переглядача” завершений, програма захоплює вікно іншої, активної, програми у ОС та транслює це потокове відео у наше вікно. Наступний функціонал дадається таким самим чином, як і попередній, але для повноцінної розробки такої програми потрібно розбити кожен функціонал на модулі та адаптувати для виконання наших задач.

* 1. **Реалізація віконної програми з майбутнім функціоналом**

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

Дизайн віконного додатку 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 в повній мірі. Створення адаптивності вікна та компонентів на ньому, що до розширення дисплею користувача, на жаль, не було взято до уваги, натомість вирішено визначити розмір вікна фіксованим значенням. Розмір віконного додатку становить 640 на 360 пікселів (див. рис. 3.2.1).

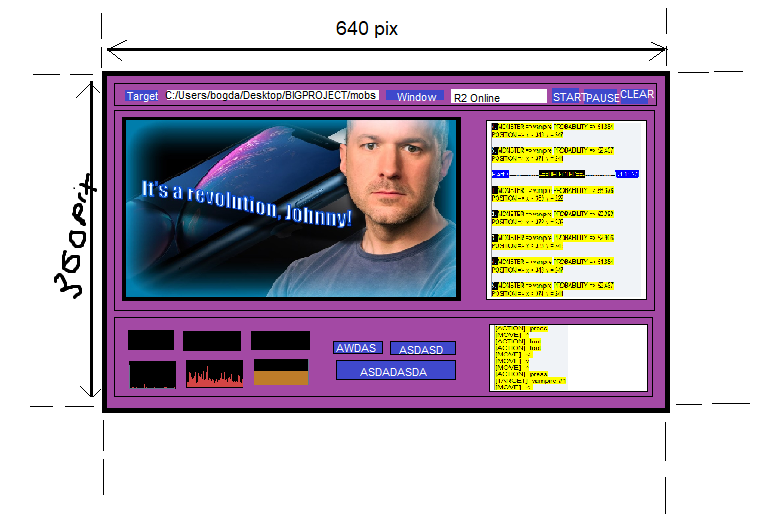


**Рис. 3.2.1.** Конфігураційний файл програми.

Макет вікна буде включати у себе 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 додатку буде виглядати наступним чином (див. рис. 3.2.2):

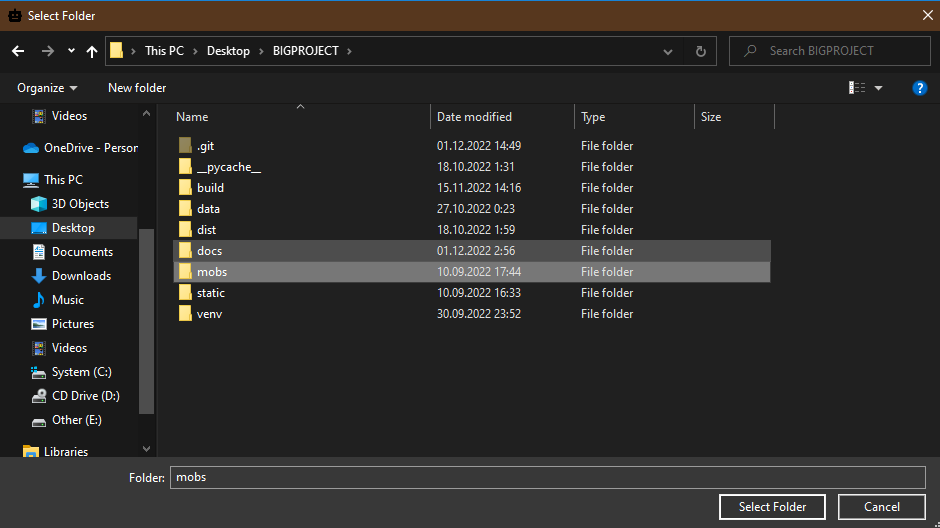


**Рис. 3.2.2.** Макет віконного додатка.

Вікно можна поділити на 3 окремі рамки:

* Вікно налаштувань;
* Інформаційне вікно;
* Вікно моніторингу.

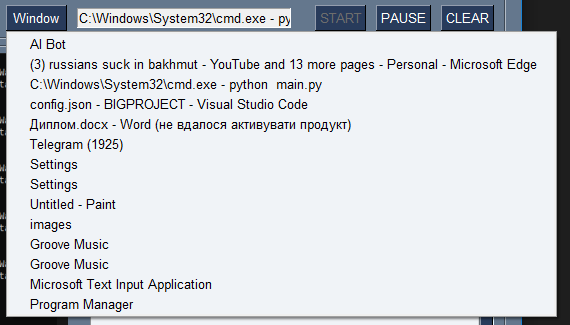
На вікні налаштувань розміщенні компоненти для налаштування програми перед її запуском. За допмогою кнопки “Target” можна відкрити менеджер файлового провідника із опцією вибору цільової директорії (див. рис. 3.2.3). Ця опція передбачає вибір користувачем тої директорії, у якій містяться конфігураційні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



**Рис. 3.2.3.** Вибір директорії налаштувань Ші.

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

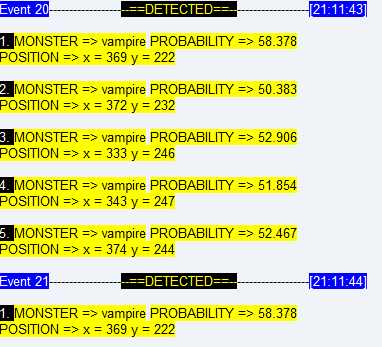
* СТАРТ;
* ПАУЗА;
* ОЧИЩЕННЯ;



**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 взагалом, або ж його відновити.

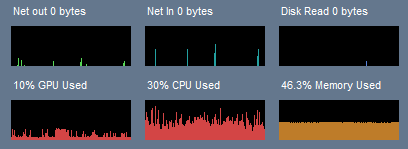
Найбільше за розміром підвікно у програмі є інформаційне вікно. Воно містить фрейм з виводом потокового відео з цільового вікна програми ОС з/без поперденьої обробки. Одне із таких вікон розташоване зліва, а друге справа – велике текстове поле. У текстовому полі виводяться інформаційні текстові повідомлення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



**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 включає у себе інформативну дошку із відомостями про стан завантаження різних компонентів 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.

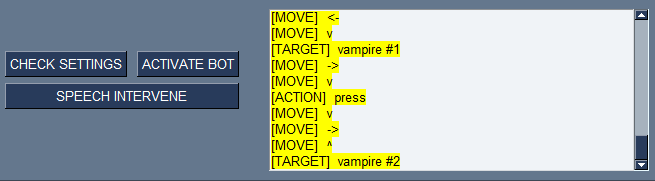


**Рис. 3.2.6.** Дашборд системи моніторингу.

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 ШІ; вибирається цільове вікно, яке відповідає конфігураційним файлам; користувач натискає кнопку перевірки. Якщо все вірно, програма розблоковує кнопку “START”. Активація бота кнопкою приведе до самостійної діяльності бота 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 процес бота. Команди, які поступають для бота інтерпритовані на “людську мову” (див. рис. 3.2.7).

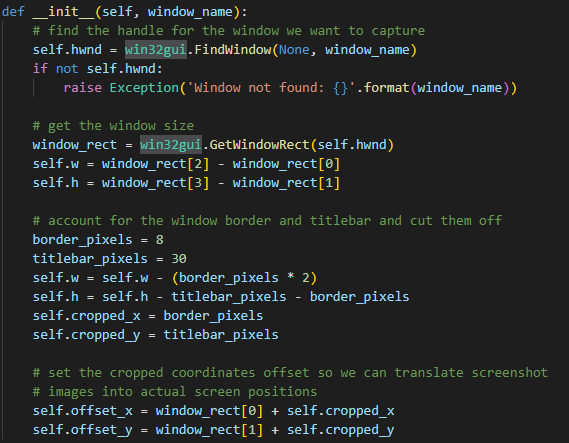


**Рис. 3.2.7.** Вікно виводу та кнопки для керування.

* 1. **Написання модулів для роботи комп’ютерного зору та розпізнавання мовлення**

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

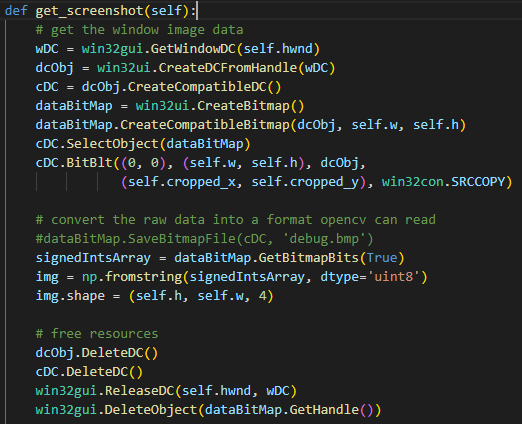
Один з ключових модулів, відповідає за захоплення потокового відео вибраного вікна програми в ОС Windows. У ньому використовується бібліотека “win32gui”. Для роботии із цією бібліотекою створений клас із конструктором та методами для роботи з даними. Екземпляр цього класу ініціалізується всього лиш назвою цільового вікна, а все, необхідне для отримання вихідних даних є виклик метода “get\_screenshot”. При створенні екземпляру, ведеться пошук вікна за іменем та отримання спеціального номера HWND, яке віндовс 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 находитимуться 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 зрозумілий.



**Рис. 3.3.1.** Інніціалізація екземпляра.

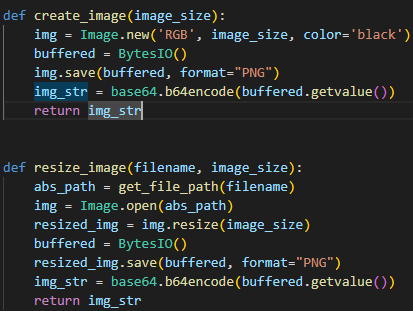
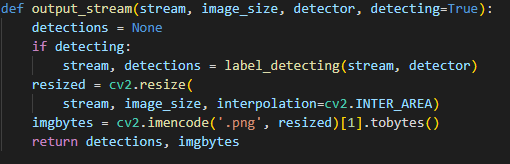
Метод “get\_screenshot” (див. рис. 3.2.2) при виклику вибирає вікно за визначеним HWND після цього створюється hadle, за допомогою якого можна отримати compatible DC, який в свою чергу необхідний для оримати бітової карти вікна задаючи параметри ширини та висоти. Отримання бітової карти зображення завершується, коли ми копіюємо його з цільового вікна. Формат зображення визначений pillow image, що дає можливість в широкому плані співпрацювати із цим типом для подальших перетворень. Після такої процедури важливо завершити роботу із вікном, очищується тимчасові обєкти та повертається матриця зображення.

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



**Рис. 3.3.2.** Метод отримання зображення.

Окрім отримання зображення в моменит часу, необхідно також його опрацювати. За опрацювання зображення відповідає код з попередньої роботи, який був адаптований під цей проект. Цей код розбитий по модулях та вмонтований у головному модулі.

а) б)

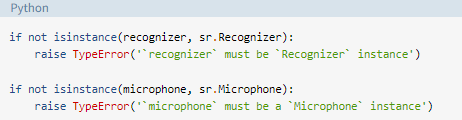
**Рис. 3.3.3.** Інструменти для перетворення ображення:

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().

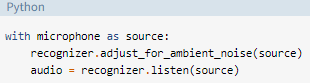
Функція accept\_speech\_from\_mic() приймає екземпляр Recognizer і Microphone як аргументи та повертає словник із трьома ключами. Перший ключ, «успіх», є логічним значенням, яке вказує, чи був успішним запит API. Другий ключ, «помилка», або None, або повідомлення про помилку, яке вказує на те, що API 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):



**Рис. 3.3.4.** Перевірка мікрофону.

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



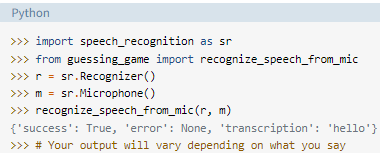
**Рис. 3.3.5.** Перевірка мікрофону.

Метод adjust\_for\_ambient\_noise() використовується для калібрування розпізнавача для 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 і 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().



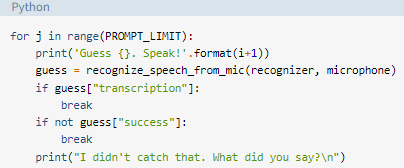
**Рис. 3.3.6.** Функція відповіді.

Ви можете 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):



**Рис. 3.3.7.** Тестування результату.

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 recogni\_speech\_from\_mic() і зберігаючи словник, повернутий до локальної змінної guess (див. рис. 3.3.8).



**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.

* 1. **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:

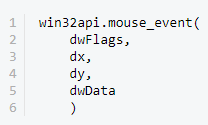
* Бібліотека зображень Python;
* Numpy;
* PyWin.

Ймовірно, підхід, який ми застосуємо, 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), наш бот буде сидіти виключно «зовні». Ми покладатимемося на методи комп’ютерного зору та виклики Windows API, щоб збирати необхідну інформацію та створювати рух. Завдяки такому підходу ми втрачаємо трохи витончених деталей і контролю, але компенсуємо це за рахунок 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 ви ознайомитеся з тим, що комп’ютер легко «бачить», ви почнете дивитися на ігри дещо інакше. Хороший приклад можна знайти в головоломках. Поширена конструкція передбачає використання людських обмежень швидкості, щоб змусити вас прийняти не оптимальне рішення. Весело (і досить легко) «ламати» 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 ми почнемо створювати сценарії будь-яких корисних дій, ми збираємося уважно поглянути на деякі функції API, на які ми будемо покладатися. Щойно ми чітко зрозуміємо кожен параметр, буде легко налаштувати їх для будь-яких цілей, які нам потрібні в грі. Подія win32api.mouse\_event() (див. рис. 3.5.1):

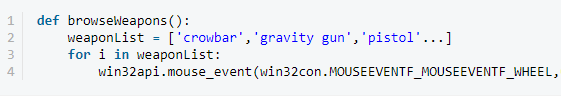


**Рис. 3.5.1.** Події миші.

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 ви хочете надіслати віртуальне клацання правою кнопкою миші, ви повинні передати win32con.MOUSEEVENTF\_RIGHTDOWN у параметр dwFlags. Наступні два параметри, dx і dy, описують абсолютне положення миші вздовж осей x і y. Хоча ми могли б використовувати ці параметри для сценарію руху миші, вони використовують систему координат, відмінну від тієї, яку ми використовували. Отже, ми залишимо 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 ми уявимо гру з системою вибору зброї, подібною до Half-Life 2, де зброю можна вибирати обертанням коліщатка миші, ми придумаємо таку функцію для перегляду списку зброї (див. рис. 3.5.3):



**Рис. 3.5.3.** Функція вибору.

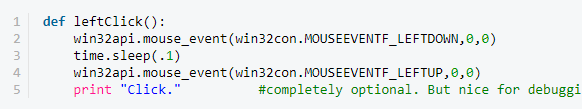
Тут ми хочемо імітувати прокручування коліщатка миші для навігації нашим теоретичним списком зброї, тому ми передали «дію» ...MOUSEEVENTF\_WHEEL у dwFlag. Нам не потрібні позиційні дані dx або dy, тому ми залишили їх нульовими, і ми хотіли прокручувати один клацання вперед для кожної «зброї» у списку, тому ми передали ціле число 120 до dwData (кожен клацання колеса дорівнює 120). Як бачите, робота з mouse\_event — це просто підключення правильних аргументів до потрібного місця. Тепер перейдемо до деяких інших корисних функцій.

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



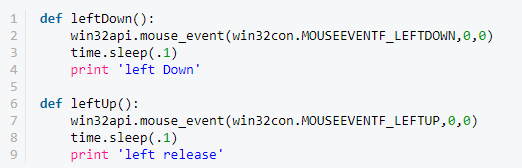
**Рис. 3.5.4.** Імпорт необхідних залежностей.

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



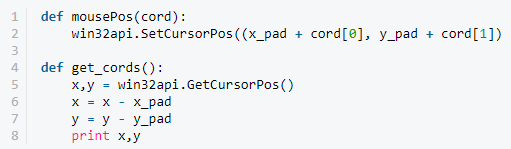
**Рис. 3.5.5.** Функція реагування на клік миші.

Згадайте, що все, що ми тут робимо, це призначаємо «дію» першому аргументу mouse\_event. Нам не потрібно передавати жодну позиційну інформацію, тому ми 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. Ми 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 ми зробили тут, це загальне клацання лівою кнопкою миші. Одне натискання, одне відпускання. Ми витратимо більшу частину нашого часу на цей, але ми 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).



**Рис. 3.5.6.** Функції для різних подій.

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



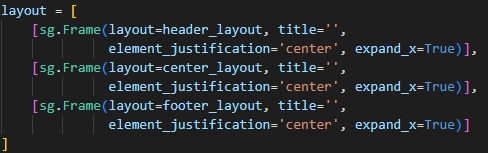
**Рис. 3.5.7.** Функції для різних подій.

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 ми додали поля x і y; важливо робити це скрізь, де викликається координата. Другий — простий інструмент, який ми будемо 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. Ми не завжди зможемо ним користуватися, оскільки деякі дії миші потребуватимуть залежно від пікселів, але коли це можливо, це фантастична економія часу.

* 1. **Інтеграція роботи модулів у програмі та тестування**

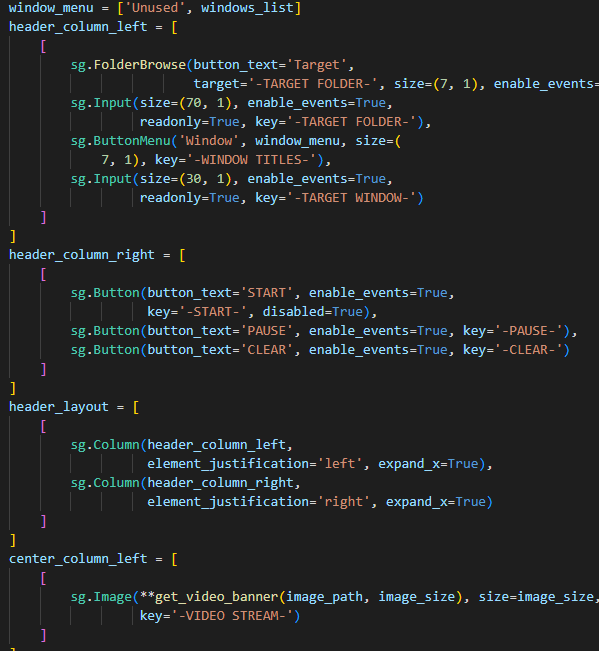
Дизайн інтерфейса програми визначений вище, бібліотека для розробки PysimpleGUI теж. Для реалізації задумки необхідно спочатку написати всі компоненти вікна. Створимо, для зручності, ще один файл під назвою utils, там будуть міститися другорядні функції для роботи в програмі. Перед розробкою необхідно 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):



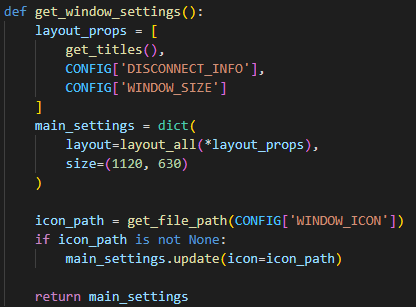
**Рис. 3.6.1.** Компоненти згруповані по фреймах.

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



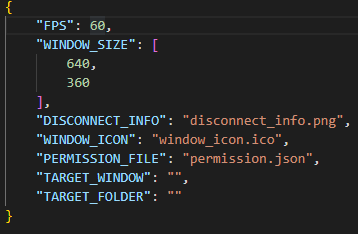
**Рис. 3.6.2.** Компоненти згруповані по фреймах.

Запуск програми починається з головного файла 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):



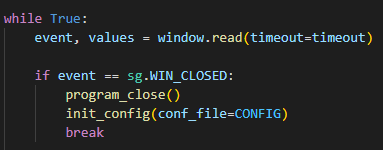
**Рис. 3.6.3.** Наповнення вікна компонентами.

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.



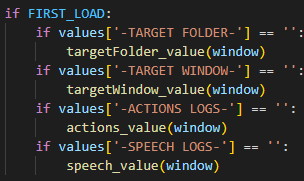
**Рис. 3.6.4.** Конфігураційний файл.

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 ітерацію, окрім назви події можна отримати також значення із форми. Налаштування частоти оновлення вікна задається в конфігураційному файлі.



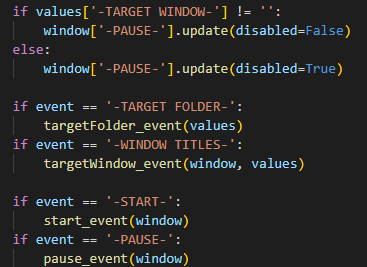
**Рис. 3.6.5.** Цикл обробки подій.

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



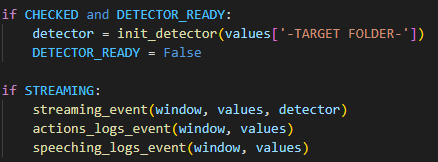
**Рис. 3.6.6.** Цикл обробки подій.

Стан головних кнопок керування програмою буде залежити 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.



**Рис. 3.6.7.** Обробка подій компонентів керування, налаштування.

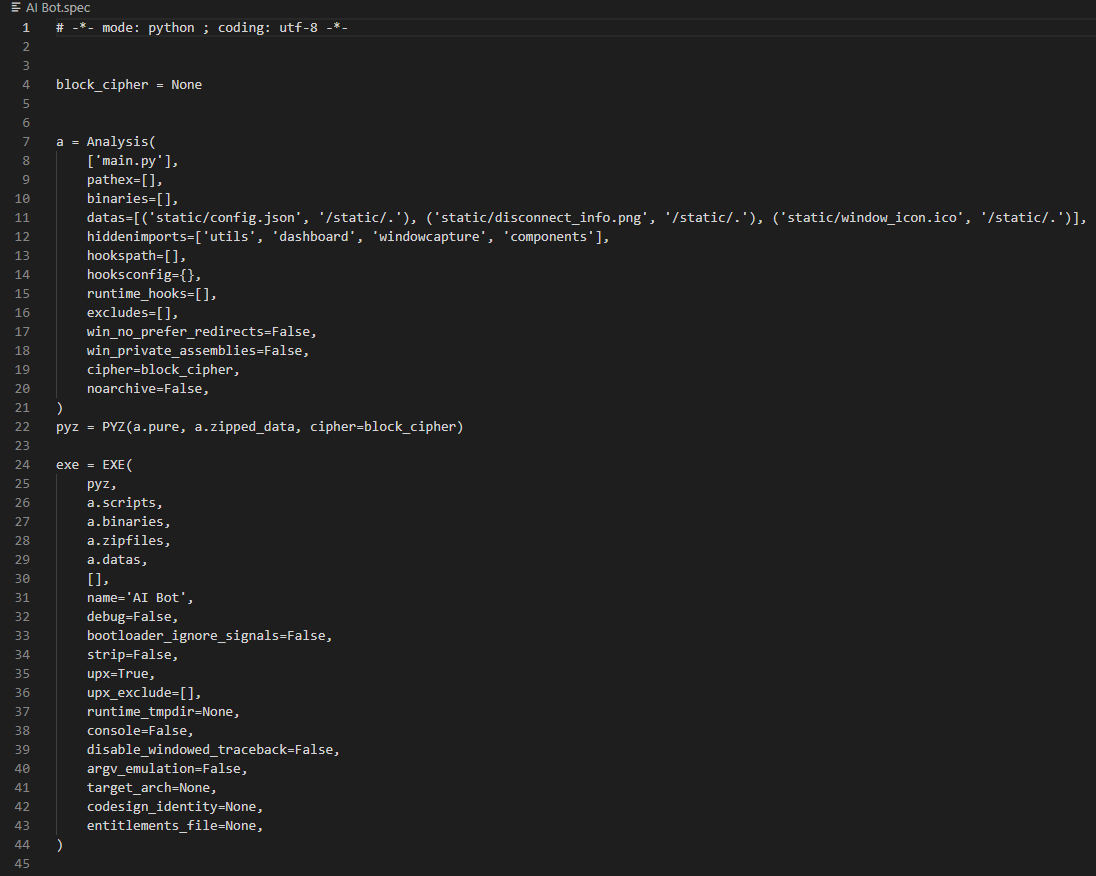
Ключовим є реакція програми на активацію роботи бота, як візуального так маніпулятивного. У програмі є запис стану: розпізнавання, готовності та реалізації. При активації цих станів відбувається виклик функцій (див. рис. 3.6.8).



**Рис. 3.6.8.** Активація бота.

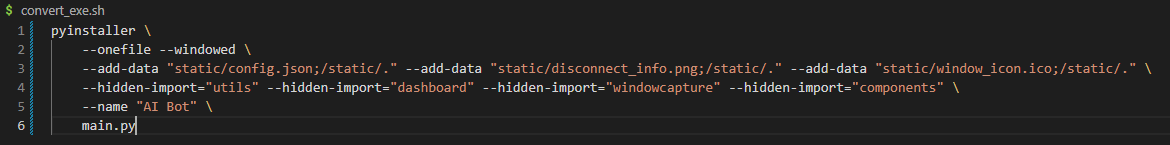
Якщо користувач пройде перевірку файлів та шалаштувань, йому буде доступна кнопка START, при нажатті якої активується режим розпізнавання. У разі, якщо це PAUSE, відбудеться або зупинка усіх процесів, або їхнє відновлення.

Для кросплатформності цього додатку, необхідно компілювати програму в виконуваний файл з розширенням exe. Хорошим інстурментом 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 і включені до вихідних даних. Однак перший названий нами сценарій надає назву для специфікаційного файлу і для виконуваної папки, файлу. Його код виконується першим під час виконання. Для певних цілей можна редагувати і вміст myscript.spec (див. рис. 3.6.9). Після того, як це зробити, назвемо специфікаційний файл для PyInstaller замість сценарію: pyinstaller myscript.spec.



**Рис. 3.6.9.** Файл специфікації 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 із специфікаційним файлом. Лише кілька параметрів командного рядка мають ефект під час створення зі специфікаційного файлу. 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 проекті.



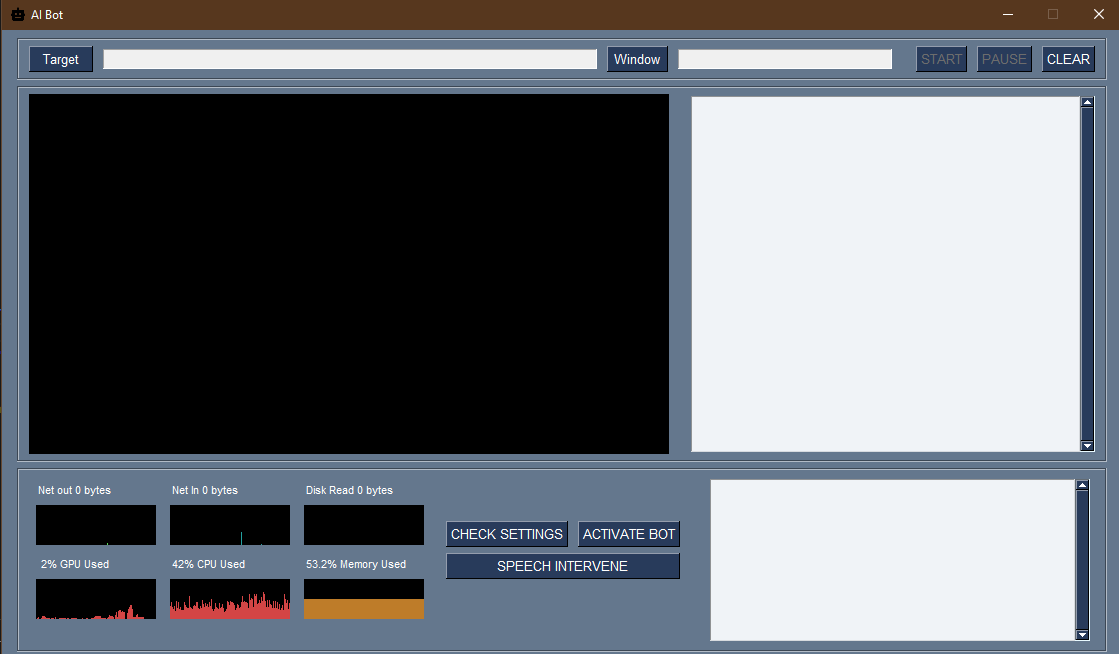
**Рис. 3.6.10.** Скрипт для запуску компіляції.

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

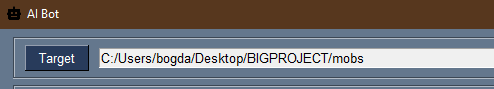
# РОЗДІЛ IV. ТЕСТУВАННЯ РОЗРОБКИ. ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

* 1. **Запуск програми в тестовому середовищі**

Почнімо із запуску програми в режимі розробника. Необхідно активувати віртуальне середовище, де мають бути інстальовані 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 підключити цільову директорію (див. рис. 4.1.2) та вказати цільове вікно (див. рис. 4.1.3).



**Рис. 4.1.1.** Вікно програми.

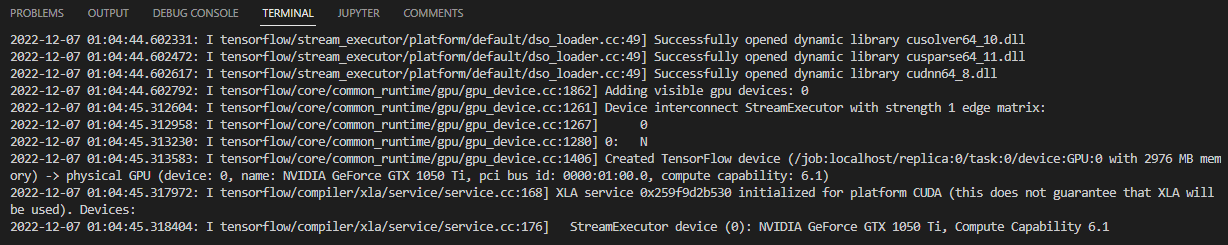


**Рис. 4.1.2.** Налаштування цільової директорії.



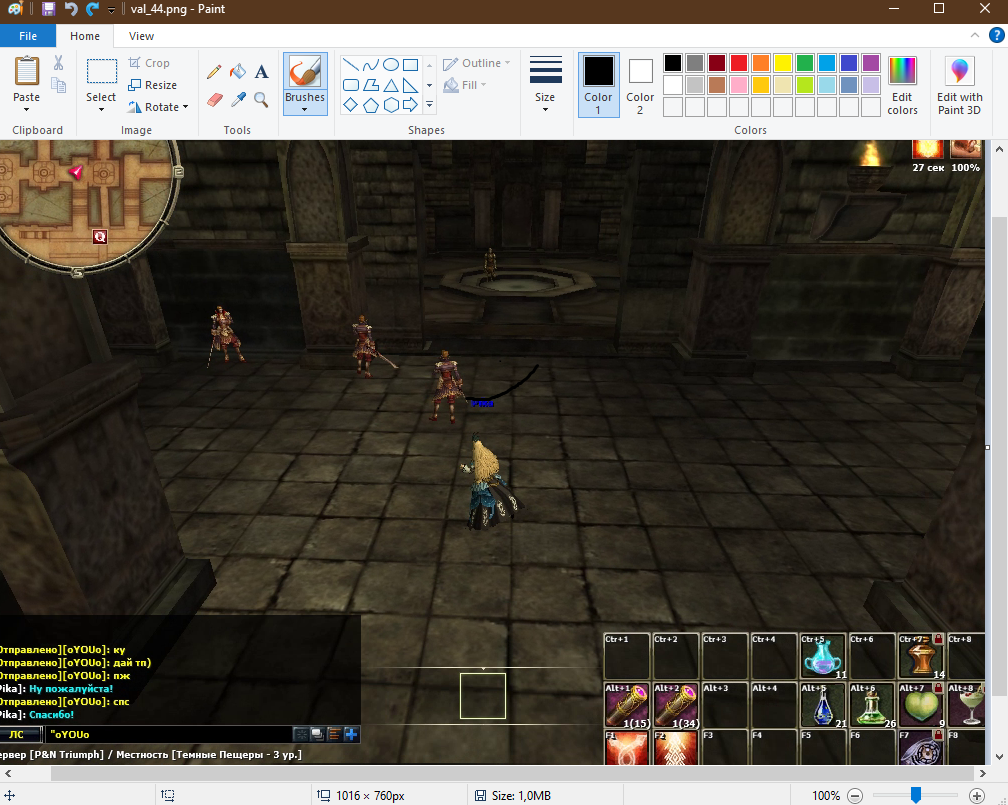
**Рис. 4.1.3.** Налаштування цільового вікна.

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



**Рис. 4.1.4.** Інніціалізація роботи з бібліотекою Tensorflow.

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

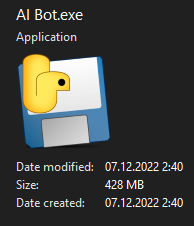


**Рис. 4.1.5.** Тестове вікно.

Сенс запускати саму гру поки що немає, адже потрібно оптимізувати основні процеси нашої програми. Окрім визначеного наперед середовща, яке імітує гру, потрібно перевірити також роботоздатність зкомпільованого 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 у ОС Windows чи ні. Наш бот запуститься у кожного, але є проблема з розміром файлу понад 400 мб, що відображається у його довготривалому запуску та повільній передачі в мережі інтернету.



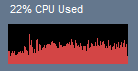
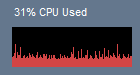
**Рис. 4.1.6.** Компіляція виконуваного файлу.



**Рис. 4.1.7.** Деталі виконавчого файлу.

* 1. **Аналіз результатів тестування та корекція програми**

Першим етапом тестування буде перевірка робота 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 і без. Отже, якщо порівняти обидва рисунки (див. рис. 4.2.1 а, б), тоді стає очевидним, що, всередньому, трансляція у нашу програму навантажує CPU ПК майже у 2 рази більше, аніж при її відсутності.

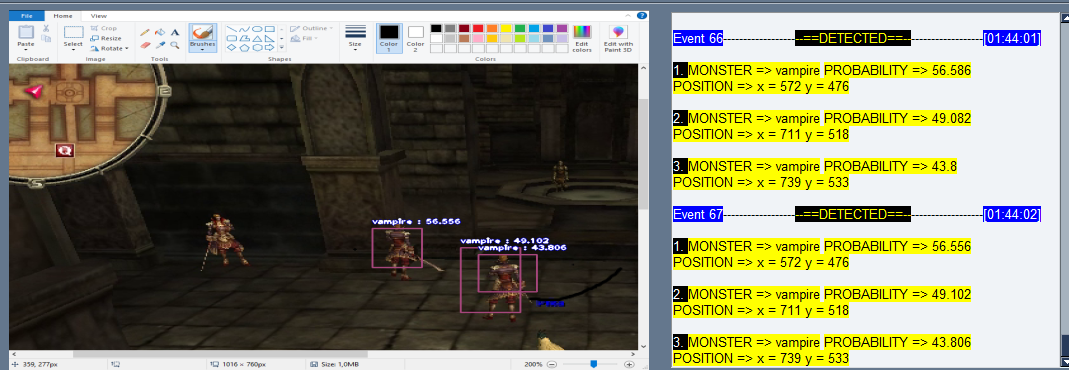
 

а) б)

**Рис. 4.2.1.** Тестування навантаження CPU при трансляції вікна:

а) вікно транслюється; б) вікно не транслюється.

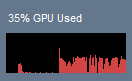
Спробуємо запустити режим розпізнавання натиснувши на кнопку “START”, але при цьому попередньо ввімкнувши трансляцію. Програма відповідає через 1-2 секунди затримки, адже відбувається додаткова інніціалізація модуля. На вікні нашої програми отримуємо наступне (див. рис. 4.2.2):



**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 монстр 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 датасет зображень, а саме збільшити його об`єм хоча би у 2 рази.

Давайте поглянемо 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 рад проблем, що виникали раніше, опрацьовано інформацію, яка допоможе оптимізувати нашу програму. Виправлено сильні затримки під час активації функцій, та сильне навантаження на CPU.

* 1. **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 у мене вже є створений ігровий акаунт, на якому я раніше проводив випробування з прототипами, розробленими у попередній роботі.

Краще запускати гру у віконному режимі з найменшим розширенням, аби наш бота працював ефективніше. З рисунку (див. рис. 4.3.1) можна помітити (у лівому верхньому куті), що вікно гри підписується, як “??”. Задумка в 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, тож можна звернути увагу на ключові елементи:

* Персонаж (маніпулятивний об`єкт);
* Середовище (область взаємодії);
* Монстри (цільові об`єкти).

Наш персонаж, взаємодії із середовищем на основі фізичних маніпуляторів таких як: клавіатура та миша. Ігрок сприймає інформацію через монітор, звуки гри для бота в цьому проекті немають сенсу. Тоді, як наша програма взаємодії з грою на нижчому рівні: отримує інформацію швидше (на рівні абстракцій), безпосередньо через програмну взаємодію.

Давайте розглянемо ключові моменти ігрового процесу, для яких розроблений наш бот. Програмі потрібно відтворювати дії, за які раніше відповідав ігрок, як от пошук цілей (монстрів, див. рис. 4.3.2), атака цілей (див. рис. 4.3.2) та отримання нагород (лут, який випадає з монстрів після їхньої смерті, див. рис. 4.3.3). Це не всі аспекти гри, бот віддповідатиме за прийняття рішень, подібним чином, як це робить людина. Алгоритм взаємодії бота у ігровому середовищі виглядає наступним чином: аналіз області (виявлення цілей), пошук маршруту/атака цілі, отримання нагороди і продовження циклу. Програма вміє працювати із казусними ситуаціями, якщо такі траплятимуться.

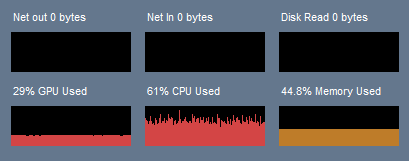


**Рис. 4.3.2.** Атака цілі.



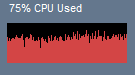
**Рис. 4.3.3.** Отримання нагороди.

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



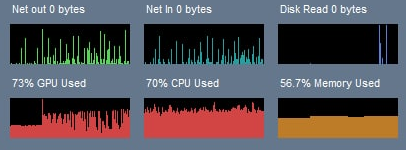
**Рис. 4.3.4.** Система моніторингу ПК.

Програма в режимі простою, давайте ввімкнемо трансляцію вікна гри. Можна помітити незначене (+15%) зростання навантаження 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 ШІ починає використовувати апаратний прискорювач для виконання роботи по розпізнавнню 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, аніж це робить наша програма. Результати інтегрованої технології розпізнавання можна побачити на рисунку (див. рис. 4.3.7):



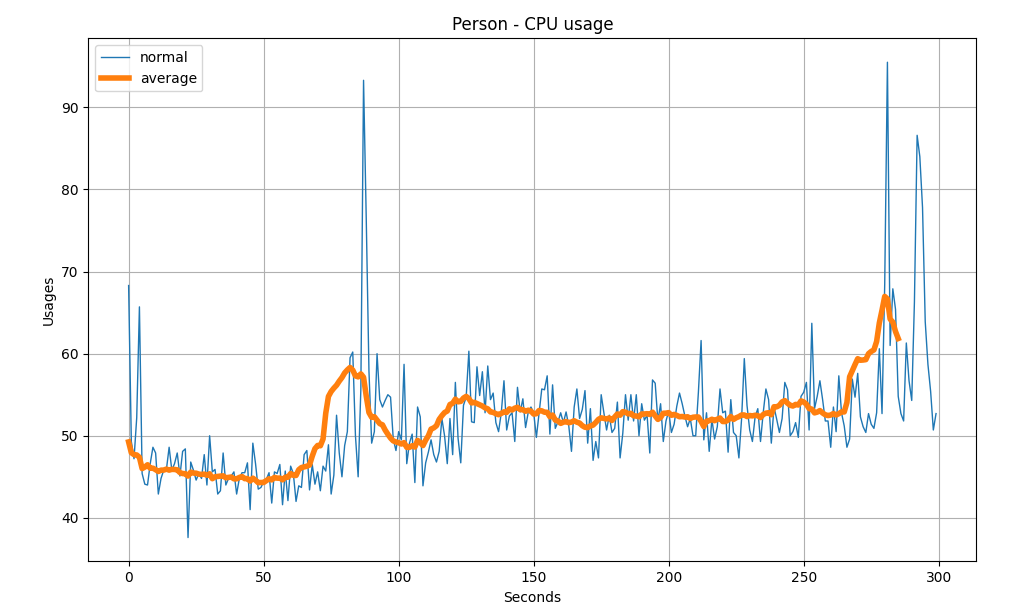
**Рис. 4.3.7.** Вікно бота.

Отже, з рисунку (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 початку дослідження – це запустити ботів на довготривалий час. Наші дослідження будуть проводитися із останнім прототипом бота та людини, окрім цього потрібно порівняти ефективність виконання задач та навантаження системи ПК.

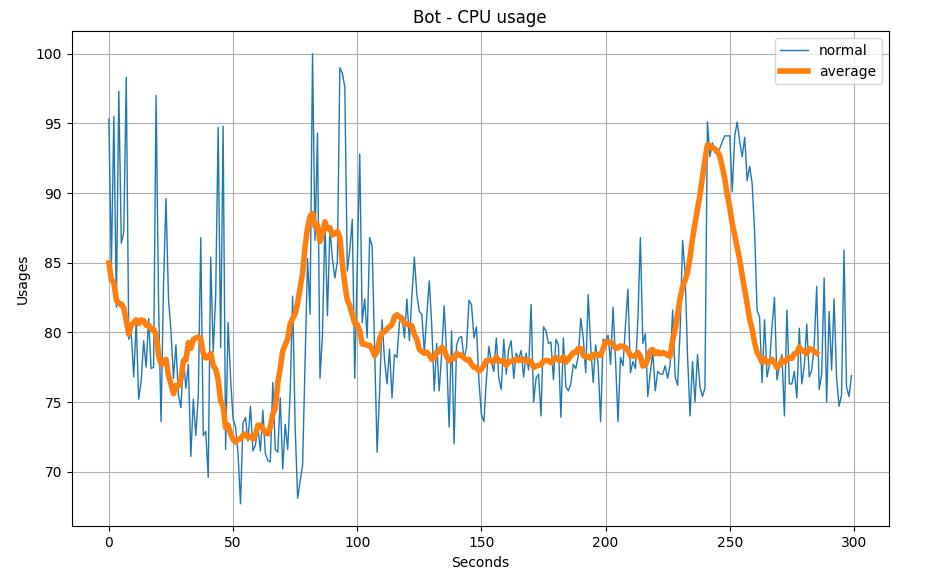
* 1. **Аналіз результатів дослідження**

Після довготривалого збору даних, потрібно опрацювати їх за допомогою Python скриптів. Отримані графіки можна порівняти і прийти де певних висновків, таких як: ефективність виконання задача, відповідність поставленій 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 наглядно прикладу, давайте порівняємо з графіком за участю бота. На рисунку (див. рис. 4.4.2) спостерігається нестабільне використання CPU. Окрім цього помітно, що наша програма навантажує ПК середньому близько 80%.



**Рис. 4.4.1.** Використання CPU при участі людини.

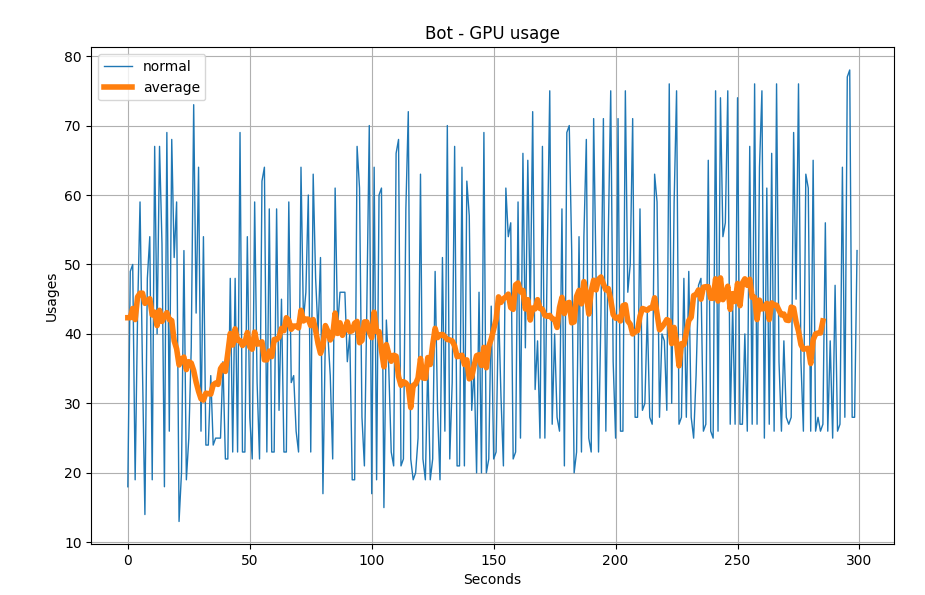


**Рис. 4.4.2.** Використання CPU при участі бота.

Найбільш інформативним, з результатів спостереження, є завантаження саме GPU, адже у попередній роботі ШІ використовував саме ресурс CPU, що довело низьку ефективність роботи попередньої 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 у другому випадку. Розмах завантаження GPU: людиною від |20% - 40%|, а ботом від |20%-80%|.

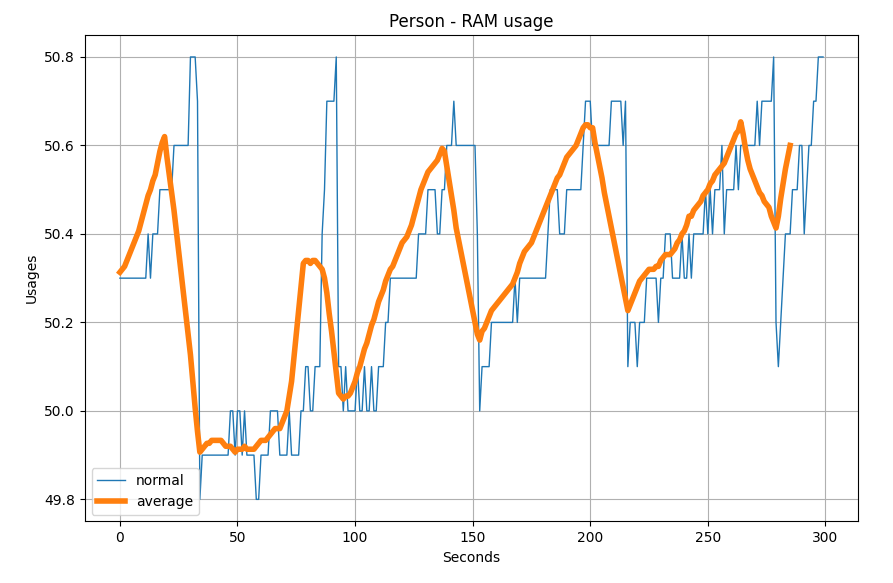


**Рис. 4.4.3.** Використання GPU при участі людини.

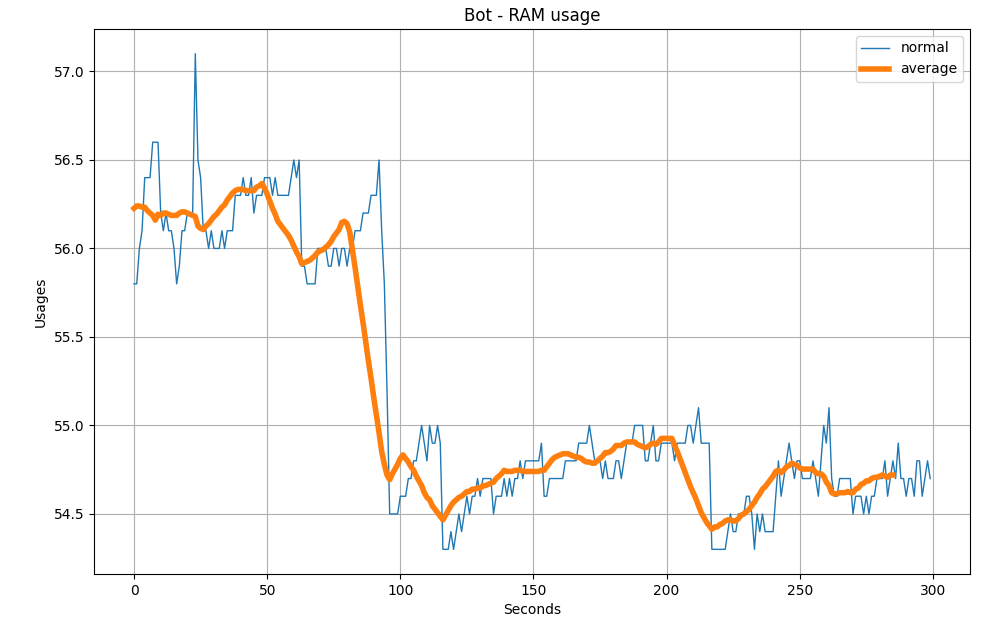


**Рис. 4.4.4.** Використання GPU при участі бота.

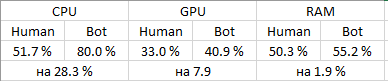
На графіках (див. рис. 4.4.5, рис.4.4.6) зображено завантаження RAM. Можна зрозуміти, що бот не впливає на завантаження оперативної памятті ПК, різниця між людиною і ботом, графіки показують лише у 5% в середньому. До підсумку проведених спостережень (див. табл. 4.4.7), можна сказати, що бот, використовує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



**Рис. 4.4.5.** Використання RAM при участі людини.

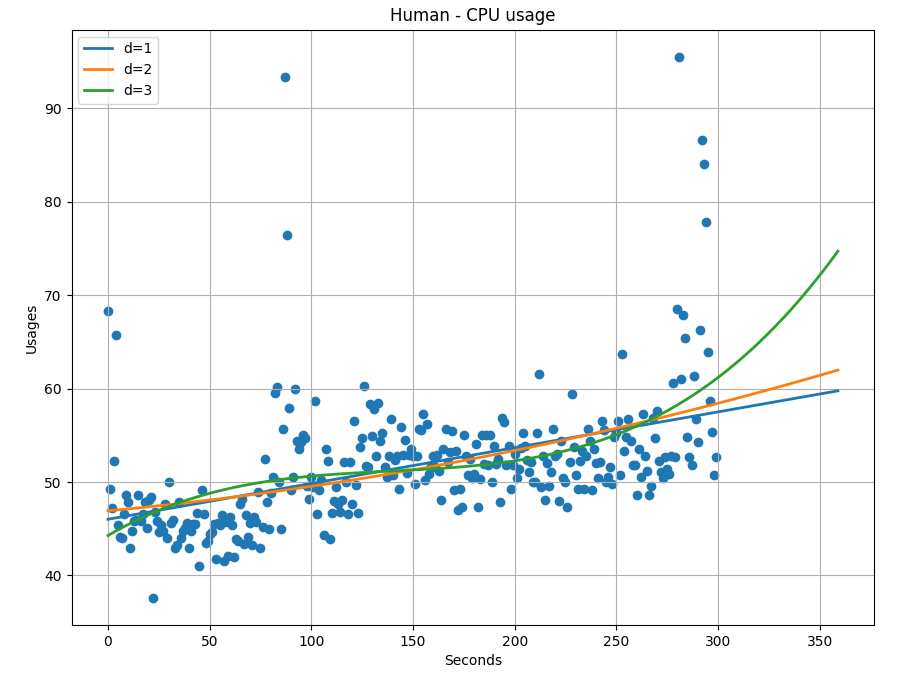


**Рис. 4.4.6.** Використання RAM при участі бота.



**Табл. 4.4.7.** Підсумок проведений 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.

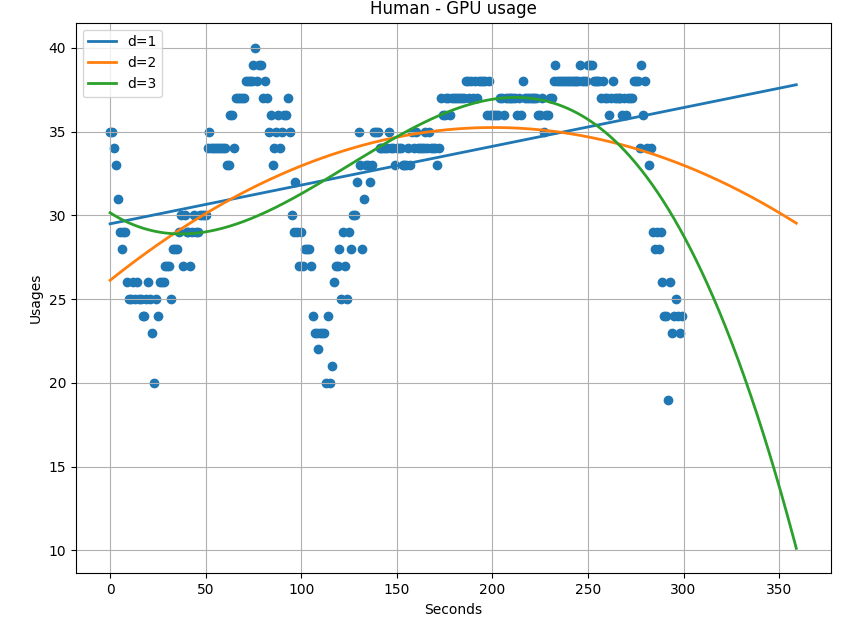


**Рис. 4.4.8.** Апроксимація даних навантаження CPU при участі людини.

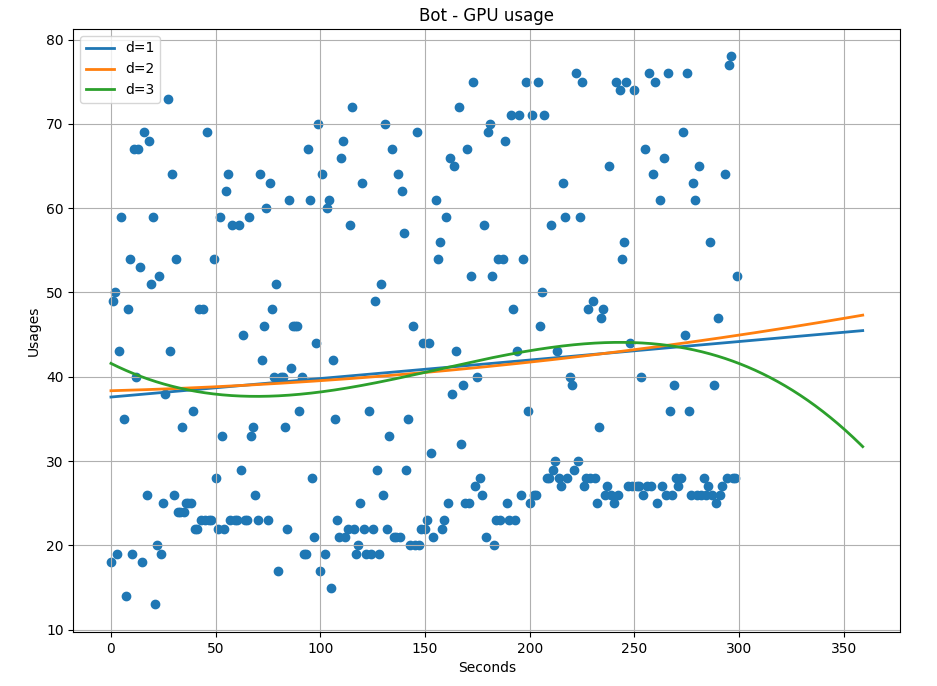


**Рис. 4.4.9.** Апроксимація даних навантаження CPU при участі бота.

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

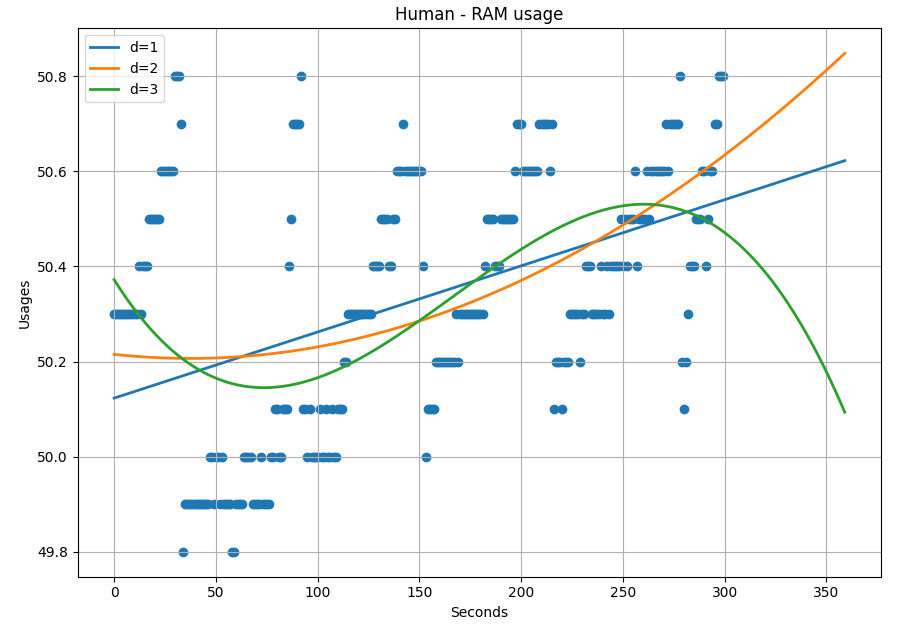


**Рис. 4.4.10.** Апроксимація даних навантаження GPU при участі людини.

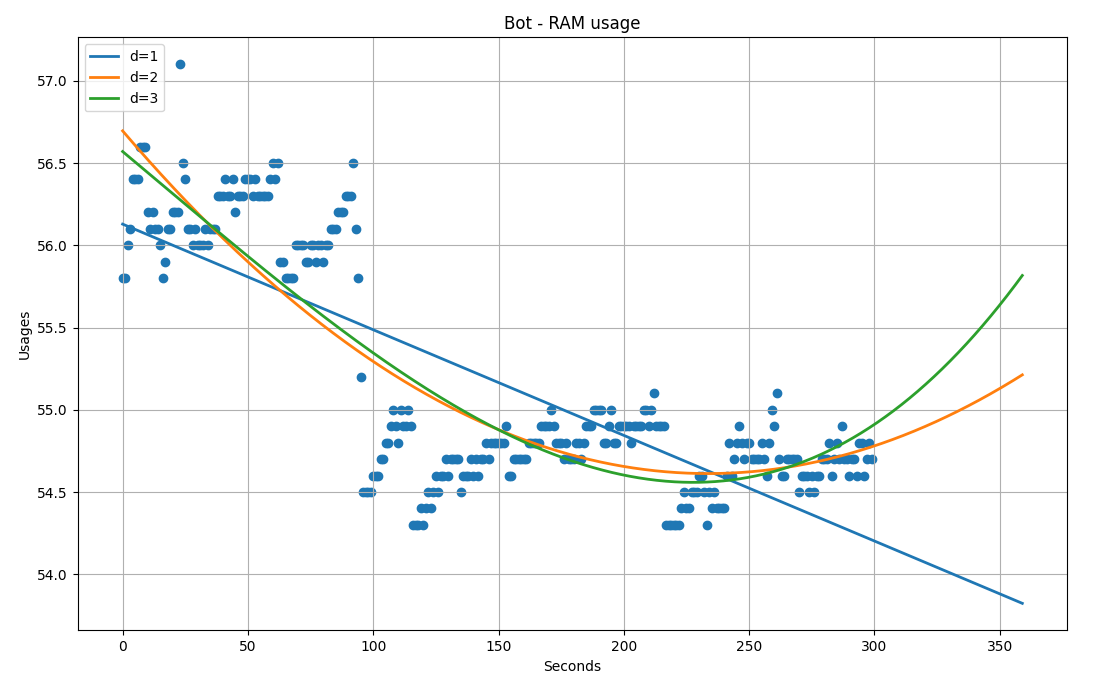


**Рис. 4.4.11.** Апроксимація даних навантаження GPU при участі бота.

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.

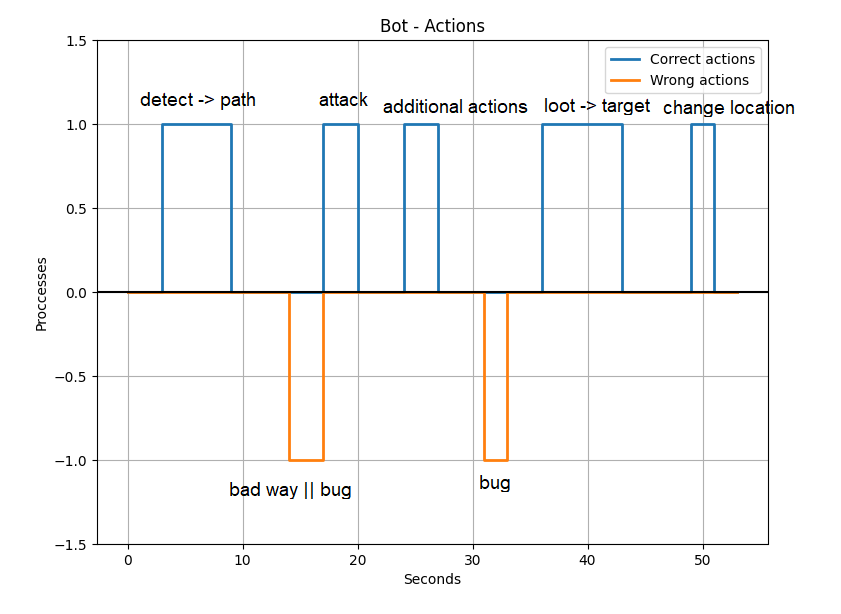


**Рис. 4.4.12.** Апроксимація даних навантаження RAM при участі людини.



**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 алгоритом, який був вже опасаний 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 справляєтсья із 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 фокосуванням цілі справляється в десятки раз швидше.



**Рис. 4.4.14.** Схематичне представлення поведінки бота в грі.



**Табл. 4.4.15.** Порівняння нашого бота з прототипом, попредньої роботи.

Як зображено на таблиці (див. табл. 4.4.15) бот з цього проекту (BOT 2) навантажує GPU на 25% більше, аніж це робить бот з попердньої роботи (BOT 1). Можна прийти до висновку, що оптимізація, яка була проведена у цьому проекті, грамотно завантажує ресурси ПК, адже робота ШІ була перенесена з CPU на GPU.

# ВИСНОВКИ

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Detection Classes [Електронний ресурс] / Режим доступу: www/URL: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 - 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/ - 09.05.2019 р.
5. Prediction Classes [Електронний ресурс] / Режим доступу: www/URL: 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. Крисилов В.А. представлення вихідних даних в задачах нейронного програмування / Одесса: 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. Нейрокомпютери: 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 [Електроний 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 [Електроний ресурс//Основні напрямки використання / URL: http://www.neuropro.ru/ ( дата звернення 20.11.2020).

# ДОДАТОК А

**main.py**

***import PySimpleGUI as sg***

***import tensorflow as tf***

***import os***

***import random***

***from imageai.Detection.Custom import CustomObjectDetection***

***from wincapture import WindowCapture***

***from components import layout\_all***

***# from speech import recognition\_speech***

***from datetime import datetime***

***from dashboard import \****

***from utils import (***

***resize\_image,***

***init\_config,***

***output\_stream,***

***get\_titles,***

***get\_file\_path,***

***get\_perm***

***)***

***os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'***

***os.environ['TF\_XLA\_FLAGS'] = '--tf\_xla\_enable\_xla\_devices'***

***tf.config.experimental.enable\_mlir\_graph\_optimization()***

***tf.config.run\_functions\_eagerly(True)***

***def start\_event(window):***

***global STREAMING, DETECTING, CHECKED, DETECTOR\_READY***

***STREAMING = True***

***DETECTING = not DETECTING***

***CHECKED = not CHECKED***

***def pause\_event(window):***

***global STREAMING, DETECTING, DETECTOR\_READY***

***STREAMING = not STREAMING***

***DETECTING = not DETECTING***

***def targetWindow\_value(window):***

***try:***

***target\_folder = CONFIG['TARGET\_WINDOW']***

***window['-TARGET WINDOW-'].update(target\_folder)***

***except:***

***pass***

***def targetWindow\_event(window, values):***

***global CONFIG***

***window\_title = values['-WINDOW TITLES-']***

***CONFIG['TARGET\_WINDOW'] = window\_title***

***window['-TARGET WINDOW-'].update(window\_title)***

***window['-WINDOW TITLES-'].update(['Unused', get\_titles()])***

***def targetFolder\_value(window):***

***try:***

***target\_folder = CONFIG['TARGET\_FOLDER']***

***window['-TARGET FOLDER-'].update(target\_folder)***

***except:***

***pass***

***def targetFolder\_event(values):***

***global CONFIG***

***CONFIG['TARGET\_FOLDER'] = values['-TARGET FOLDER-']***

***def actions\_value(window):***

***try:***

***actions = CONFIG['ACTIONS\_LOGS']***

***window['-ACTIONS LOGS-'].update(actions)***

***except:***

***pass***

***def speech\_value(window):***

***try:***

***speech = CONFIG['SPEECH\_LOGS']***

***window['-SPEECH LOGS-'].update(speech)***

***except:***

***pass***

***def activate\_game\_bot(values):***

***if 'R2' not in values['-TARGET WINDOW-'] or '??' not in values['-TARGET WINDOW-']:***

***sg.Popup('WARNING!\ngame is not ready',***

***title='Activate ERROR')***

***def streaming\_event(window, values, detector):***

***global DETECTIONS***

***try:***

***wincap = WindowCapture(values['-TARGET WINDOW-'])***

***stream = wincap.get\_screenshot()***

***except:***

***return***

***DETECTIONS, stream\_data = output\_stream(***

***stream, CONFIG['WINDOW\_SIZE'], detector, detecting=DETECTING)***

***window['-VIDEO STREAM-'].update(data=stream\_data)***

***def actions\_logs\_event(window, values):***

***global DETECTOR\_COUNT***

***if DETECTING and DETECTIONS is not None:***

***from datetime import datetime***

***now = datetime.now()***

***time = now.strftime("%H:%M:%S")***

***if DETECTIONS != []:***

***DETECTOR\_COUNT += 1***

***window["-ACTIONS LOGS-"].update('\n\n', append=True)***

***window["-ACTIONS LOGS-"].update('Event ' + str(DETECTOR\_COUNT), text\_color\_for\_value='white',***

***background\_color\_for\_value='blue', append=True)***

***window["-ACTIONS LOGS-"].update('-'\*18, append=True)***

***window["-ACTIONS LOGS-"].update(***

***'--==DETECTED==--', text\_color\_for\_value='yellow', background\_color\_for\_value='black', append=True)***

***window["-ACTIONS LOGS-"].update('-'\*18, append=True)***

***window["-ACTIONS LOGS-"].update(***

***'[' + time + ']', text\_color\_for\_value='white', background\_color\_for\_value='blue', append=True)***

***for idx, detect in enumerate(DETECTIONS):***

***window["-ACTIONS LOGS-"].update('\n\n', append=True)***

***window["-ACTIONS LOGS-"].update(f'{idx+1}. ', text\_color\_for\_value='white',***

***background\_color\_for\_value='black', append=True)***

***window["-ACTIONS LOGS-"].update('MONSTER => ' + detect['name'],***

***text\_color\_for\_value='black', background\_color\_for\_value='yellow', append=True)***

***window["-ACTIONS LOGS-"].update('\t', append=True)***

***window["-ACTIONS LOGS-"].update('PROBABILITY => ' + str(round(detect['percentage\_probability'], 3)),***

***text\_color\_for\_value='black', background\_color\_for\_value='yellow', append=True)***

***window["-ACTIONS LOGS-"].update('\n', append=True)***

***window["-ACTIONS LOGS-"].update('POSITION => x = ' + str(detect['box\_points'][0]) + ' y = ' + str(detect['box\_points'][1]),***

***text\_color\_for\_value='black', background\_color\_for\_value='yellow', append=True)***

***def speech\_window():***

***window = sg.Window('Speech Recognition', layout=[***

***[sg.Button('Hello World')]])***

***while True:***

***event, values = window.read()***

***if event == sg.WIN\_CLOSED:***

***break***

***window.close()***

***def speeching\_logs\_event(window, values):***

***global DETECTOR\_COUNT***

***if DETECTING and DETECTIONS is not None:***

***now = datetime.now()***

***time = now.strftime("%H:%M:%S")***

***commands = random.choices(***

***SLAVE\_MESSAGE['commands'], weights=SLAVE\_MESSAGE['weights'], k=random.randint(1, 3))***

***# time = '-=== DETECTED ===-'***

***if DETECTIONS != []:***

***for command in commands:***

***window["-SPEECH LOGS-"].update('\n', append=True)***

***window["-SPEECH LOGS-"].update(command,***

***text\_color\_for\_value='black', background\_color\_for\_value='yellow', append=True)***

***def clear\_event(window):***

***global CONFIG, FIRST\_LOAD***

***FIRST\_LOAD = False***

***window['-START-'].update(disabled=True)***

***window['-TARGET FOLDER-'].update('')***

***window['-TARGET WINDOW-'].update('')***

***window['-VIDEO STREAM-'].update(***

***data=resize\_image(CONFIG['DISCONNECT\_INFO'], CONFIG['WINDOW\_SIZE']))***

***window['-ACTIONS LOGS-'].update('')***

***window['-SPEECH LOGS-'].update('')***

***def program\_close():***

***global CONFIG***

***CONFIG['TARGET\_WINDOW'] = VALUES['-TARGET WINDOW-']***

***CONFIG['TARGET\_FOLDER'] = VALUES['-TARGET FOLDER-']***

***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***

***def save\_actions\_logs(values):***

***with open('actions\_logs.txt', 'w') as file:***

***file.write(values['-ACTIONS LOGS-'])***

***def check\_event(window, values):***

***global DETECTING, CHECKED***

***target\_folder = values['-TARGET FOLDER-']***

***perm\_file = CONFIG['PERMISSION\_FILE']***

***if not get\_perm(perm\_file, target\_folder, window, values):***

***sg.Popup('WARNING!\n"target" or "window" is not ready',***

***title='CHECK ERROR')***

***else:***

***CHECKED = True***

***def init\_detector(folder\_path):***

***model\_path = os.path.join(***

***folder\_path, "models/detection\_model-ex-049--loss-0012.515.h5")***

***json\_path = os.path.join(folder\_path, "json/detection\_config.json")***

***detector = CustomObjectDetection()***

***detector.setModelTypeAsYOLOv3()***

***detector.setModelPath(model\_path)***

***detector.setJsonPath(json\_path)***

***detector.loadModel()***

***return detector***

def psutility(window, net\_graph\_in, net\_graph\_out, disk\_graph\_read, gpu\_usage\_graph, cpu\_usage\_graph, mem\_usage\_graph):

netio = psutil.net\_io\_counters()

write\_bytes = net\_graph\_out.graph\_value(netio.bytes\_sent)

read\_bytes = net\_graph\_in.graph\_value(netio.bytes\_recv)

window['\_NET\_OUT\_TXT\_'].update(

'Net out {}'.format(human\_size(write\_bytes)))

window['\_NET\_IN\_TXT\_'].update(

'Net In {}'.format(human\_size(read\_bytes)))

# ----- Disk Graphs -----

diskio = psutil.disk\_io\_counters()

read\_bytes = disk\_graph\_read.graph\_value(diskio.read\_bytes)

window['\_DISK\_READ\_TXT\_'].update(

'Disk Read {}'.format(human\_size(read\_bytes)))

# ----- GPU Graph -----

gpu = GPUtil.getGPUs()[-1].load\*100

gpu\_usage\_graph.graph\_percentage\_abs(gpu)

window['\_GPU\_TXT\_'].update('{0:2.0f}% GPU Used'.format(gpu))

# ----- CPU Graph -----

cpu = psutil.cpu\_percent(0)

cpu\_usage\_graph.graph\_percentage\_abs(cpu)

window['\_CPU\_TXT\_'].update('{0:2.0f}% CPU Used'.format(cpu))

# ----- Memory Graph -----

mem\_used = psutil.virtual\_memory().percent

mem\_usage\_graph.graph\_percentage\_abs(mem\_used)

window['\_MEM\_TXT\_'].update('{}% Memory Used'.format(mem\_used))

***CONFIG = init\_config()***

***STREAMING = True***

***DETECTING = False***

***FIRST\_LOAD = True***

***VALUES = None***

***CHECKED = False***

***DETECTIONS = None***

***DETECTOR\_COUNT = 0***

***DETECTOR\_READY = True***

***def main():***

***global VALUES, DETECTOR\_READY***

***detector = None***

***main\_settings = get\_window\_settings()***

***window = sg.Window('AI Bot', \*\*main\_settings)***

***timeout = int(1000/CONFIG['FPS'])***

netio = psutil.net\_io\_counters()

net\_in = window['\_NET\_IN\_GRAPH\_']

net\_graph\_in = DashGraph(net\_in, netio.bytes\_recv, '#23a0a0')

net\_out = window['\_NET\_OUT\_GRAPH\_']

net\_graph\_out = DashGraph(net\_out, netio.bytes\_sent, '#56d856')

diskio = psutil.disk\_io\_counters()

disk\_graph\_read = DashGraph(

window['\_DISK\_READ\_GRAPH\_'], diskio.read\_bytes, '#5681d8')

gpu\_usage\_graph = DashGraph(window['\_GPU\_GRAPH\_'], 0, '#d34545')

cpu\_usage\_graph = DashGraph(window['\_CPU\_GRAPH\_'], 0, '#d34545')

mem\_usage\_graph = DashGraph(window['\_MEM\_GRAPH\_'], 0, '#BE7C29')

***while True:***

***event, values = window.read(timeout=timeout)***

***if event == sg.WIN\_CLOSED:***

***program\_close()***

***init\_config(conf\_file=CONFIG)***

***break***

***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)***

***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)***

***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)***

***if event == '-CLEAR-':***

***clear\_event(window)***

***if event == '-CHECK-':***

***check\_event(window, values)***

***if event == '-ACTIVATE-':***

***activate\_game\_bot(values)***

***if event == '-SPEECH-':***

***speech\_window()***

***if event == 'Save':***

***save\_actions\_logs(values)***

***psutility(window, net\_graph\_in, net\_graph\_out, disk\_graph\_read,***

***gpu\_usage\_graph, cpu\_usage\_graph, mem\_usage\_graph)***

***VALUES = {\*\*values}***

***window.close()***

***if \_\_name\_\_ == '\_\_main\_\_':***

***main()***

**components.py**

***from utils import resize\_image, get\_file\_path, create\_image***

***import PySimpleGUI as sg***

***import os***

***def get\_video\_banner(image\_path, image\_size):***

***sources = dict()***

***path = get\_file\_path(image\_path)***

***if os.path.exists(path):***

***sources.update(source=resize\_image(image\_path, image\_size))***

***else:***

***sources.update(source=create\_image(image\_size))***

***return sources***

***def layout\_all(windows\_list, image\_path, image\_size):***

***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, enable\_events=True,***

***key='-VIDEO STREAM-')***

***]***

***]***

***multiline\_menu = ['', ['Save']]***

***center\_column\_right = [***

***[***

***sg.Multiline(size=(55, 22), key='-ACTIONS LOGS-', autoscroll=True, disabled=True,***

***enable\_events=True, right\_click\_menu=multiline\_menu)***

***]***

***]***

***center\_layout = [***

***[***

***sg.Column(center\_column\_left,***

***element\_justification='left'),***

***sg.Column(center\_column\_right,***

***element\_justification='right', expand\_x=True)***

***]***

***]***

***GRAPH\_WIDTH, GRAPH\_HEIGHT = 120, 40***

***def GraphColumn(name, key):***

***layout = [***

***[sg.Text(name, size=(18, 1), font=('Helvetica 8'), key=key+'TXT\_')],***

***[sg.Graph((GRAPH\_WIDTH, GRAPH\_HEIGHT),***

***(0, 0),***

***(GRAPH\_WIDTH, 100),***

***background\_color='black',***

***key=key+'GRAPH\_')]]***

***return sg.Col(layout, pad=(2, 2))***

***dash\_layout = [***

***[GraphColumn('Net Out', '\_NET\_OUT\_'),***

***GraphColumn('Net In', '\_NET\_IN\_'),***

***GraphColumn('Disk Read', '\_DISK\_READ\_')],***

***[GraphColumn('GPU Usage', '\_GPU\_'),***

***GraphColumn('CPU Usage', '\_CPU\_'),***

***GraphColumn('Memory Usage', '\_MEM\_')],***

***]***

***footer\_column\_left = [***

***[***

***sg.Column(dash\_layout),***

***sg.Column([***

***[***

***sg.Button(button\_text='CHECK SETTINGS',***

***enable\_events=True, key='-CHECK-', expand\_x=True),***

***sg.Button(button\_text='ACTIVATE BOT',***

***key='-ACTIVATE-', expand\_x=True)***

***],***

***[***

***sg.Button(button\_text='SPEECH INTERVENE',***

***key='-SPEECH-', expand\_x=True)***

***]***

***])***

***]***

***]***

***footer\_column\_right = [***

***[***

***sg.Column([***

***[sg.Multiline(size=(55, 18), key='-SPEECH LOGS-', autoscroll=True, disabled=True,***

***enable\_events=True)]***

***])***

***]***

***]***

***footer\_layout = [***

***[***

***sg.Column(footer\_column\_left, element\_justification='left',***

***expand\_x=True, expand\_y=True),***

***sg.Column(footer\_column\_right,***

***element\_justification='right', expand\_x=True)***

***]***

***]***

**speech.py**

***import os***

***import sys***

***import json***

***import random***

***import time***

***import speech\_recognition as sr***

***from pydub import AudioSegment***

***def convert\_ogg\_to\_wav(file\_path):***

***input\_file\_path\_splited = file\_path.split('/')***

***output\_filename = input\_file\_path\_splited[-1].replace('.ogg', '.wav')***

***output\_file\_path = '/'.join(input\_file\_path\_splited[:-1]) + \***

***'/' + output\_filename***

***sound = AudioSegment.from\_ogg(file\_path)***

***sound.export(output\_file\_path, format="wav")***

***def recognition(file\_path, lang='uk-UA'):***

***r = sr.Recognizer()***

***audio\_file = sr.AudioFile(file\_path)***

***with audio\_file as source:***

***r.adjust\_for\_ambient\_noise(source)***

***audio = r.record(source)***

***recognited = r.recognize\_google(audio, language=lang)***

***return recognited***

***def save\_extract\_text(text, file\_path=None):***

***if file\_path is None:***

***file\_path = os.path.join(os.getcwd(), 'Текст.txt')***

***with open(file\_path, 'w', encoding='utf-8') as file:***

***file.write(text)***

***def get\_file\_path(filename):***

***bundle\_dir = getattr(***

***sys, '\_MEIPASS', os.path.abspath(os.path.dirname(\_\_file\_\_)))***

***path\_to\_file = os.path.abspath(***

***os.path.join(bundle\_dir, 'static/' + filename))***

***return path\_to\_file***

***def get\_lang\_tags():***

***json\_path = get\_file\_path('language\_tags.json')***

***with open(json\_path, 'r') as file:***

***lang\_tags = json.load(file)***

***return lang\_tags***

***def recognize\_speech\_from\_mic(recognizer, microphone):***

***if not isinstance(recognizer, sr.Recognizer):***

***raise TypeError("`recognizer` must be `Recognizer` instance")***

***if not isinstance(microphone, sr.Microphone):***

***raise TypeError("`microphone` must be `Microphone` instance")***

***with microphone as source:***

***recognizer.adjust\_for\_ambient\_noise(source)***

***audio = recognizer.listen(source)***

***response = {***

***"success": True,***

***"error": None,***

***"transcription": None***

***}***

***try:***

***response["transcription"] = recognizer.recognize\_google(audio)***

***except sr.RequestError:***

***response["success"] = False***

***response["error"] = "API unavailable"***

***except sr.UnknownValueError:***

***response["error"] = "Unable to recognize speech"***

***return response***

def recognition\_speech():

WORDS = ["vampire", "loot", "gold", "hold", "attack", "lemon"]

NUM\_GUESSES = 3

PROMPT\_LIMIT = 5

recognizer = sr.Recognizer()

microphone = sr.Microphone()

# get a random word from the list

word = random.choice(WORDS)

instructions = (

"I'm thinking of one of these words:\n"

"{words}\n"

"You have {n} tries to guess which one.\n"

).format(words=', '.join(WORDS), n=NUM\_GUESSES)

print(instructions)

time.sleep(3)

for i in range(NUM\_GUESSES):

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")

if guess["error"]:

print("ERROR: {}".format(guess["error"]))

break

print("You said: {}".format(guess["transcription"]))

guess\_is\_correct = guess["transcription"].lower() == word.lower()

user\_has\_more\_attempts = i < NUM\_GUESSES - 1

if guess\_is\_correct:

print("Correct! You win!".format(word))

break

elif user\_has\_more\_attempts:

print("Incorrect. Try again.\n")

else:

print("Sorry, you lose!\nI was thinking of '{}'.".format(word))

break

**wincapture.py**

import numpy as np

import win32gui

import win32ui

import win32con

class WindowCapture:

w = 0

h = 0

hwnd = None

cropped\_x = 0

cropped\_y = 0

offset\_x = 0

offset\_y = 0

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))

window\_rect = win32gui.GetWindowRect(self.hwnd)

self.w = window\_rect[2] - window\_rect[0]

self.h = window\_rect[3] - window\_rect[1]

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

self.offset\_x = window\_rect[0] + self.cropped\_x

self.offset\_y = window\_rect[1] + self.cropped\_y

def get\_screenshot(self):

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)

signedIntsArray = dataBitMap.GetBitmapBits(True)

img = np.fromstring(signedIntsArray, dtype='uint8')

img.shape = (self.h, self.w, 4)

dcObj.DeleteDC()

cDC.DeleteDC()

win32gui.ReleaseDC(self.hwnd, wDC)

win32gui.DeleteObject(dataBitMap.GetHandle())

img = img[..., :3]

img = np.ascontiguousarray(img)

return img

def list\_window\_names(self):

def winEnumHandler(hwnd, ctx):

if win32gui.IsWindowVisible(hwnd):

print(hex(hwnd), win32gui.GetWindowText(hwnd))

win32gui.EnumWindows(winEnumHandler, None)

def get\_screen\_position(self, pos):

return (pos[0] + self.offset\_x, pos[1] + self.offset\_y)

**utils.py**

***from io import BytesIO***

***from PIL import Image***

***import cv2***

***import os***

***import base64***

***import json***

***import win32gui***

***import sys***

***DEFAULT\_CONFIG = {***

***"FPS": 20,***

***"WINDOW\_SIZE": [***

***640,***

***360***

***],***

***"DISCONNECT\_INFO": "disconnect\_info.png",***

***"WINDOW\_ICON": "window\_icon.ico"***

***}***

***def get\_file\_path(filename):***

***bundle\_dir = getattr(***

***sys, '\_MEIPASS', os.path.abspath(os.path.dirname(\_\_file\_\_)))***

***path\_to\_file = os.path.abspath(***

***os.path.join(bundle\_dir, 'static/' + filename))***

***return path\_to\_file***

***def init\_config(conf\_file=None, file\_path='config.json'):***

***file\_path = get\_file\_path(file\_path)***

***if conf\_file is None:***

***mode = 'r'***

***else:***

***mode = 'w'***

***file\_creating = True***

***while True:***

***if os.path.exists(file\_path):***

***with open(file\_path, mode) as file:***

***if mode == 'r':***

***conf\_file = json.load(file)***

***return conf\_file***

***json.dump(conf\_file, file, indent=4)***

***break***

***if file\_creating:***

***with open(file\_path, 'w') as file:***

***json.dump(DEFAULT\_CONFIG, file, indent=4)***

***file\_creating = False***

***def get\_perm(filename, directory, window, values):***

***if filename != '' and directory != '':***

***if filename in os.listdir(directory):***

***abs\_path = os.path.join(directory, filename)***

***with open(abs\_path, 'r') as file:***

***ready\_check = json.load(file)***

***if ready\_check['READY'] and values['-TARGET WINDOW-'] == ready\_check['WINDOW']:***

***window['-START-'].update(disabled=False)***

***return True***

***window['-START-'].update(disabled=True)***

***return False***

***def get\_titles():***

***WINDOW\_LIST = []***

***def winEnumHandler(hwnd, ctx):***

***nonlocal WINDOW\_LIST***

***if win32gui.IsWindowVisible(hwnd):***

***window\_title = win32gui.GetWindowText(hwnd)***

***if window\_title != '':***

***WINDOW\_LIST.append(window\_title)***

***win32gui.EnumWindows(winEnumHandler, None)***

***return WINDOW\_LIST***

***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***

***def label\_detecting(stream, detector):***

***detections = detector.detectObjectsFromImage(***

***input\_image=stream,***

***input\_type='array',***

***output\_type='array',***

***minimum\_percentage\_probability=40,***

***extract\_detected\_objects=True,***

***thread\_safe=False***

***)***

***return detections[0], detections[1]***

***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***

**dashboard.py**

import PySimpleGUI as sg

import psutil

import GPUtil

# each individual graph size in pixels

GRAPH\_WIDTH, GRAPH\_HEIGHT = 120, 40

ALPHA = .7

class DashGraph(object):

def \_\_init\_\_(self, graph\_elem, starting\_count, color):

self.graph\_current\_item = 0

self.graph\_elem = graph\_elem # type:sg.Graph

self.prev\_value = starting\_count

self.max\_sent = 1

self.color = color

self.graph\_lines = []

def graph\_value(self, current\_value):

delta = current\_value - self.prev\_value

self.prev\_value = current\_value

self.max\_sent = max(self.max\_sent, delta)

percent\_sent = 100 \* delta / self.max\_sent

line\_id = self.graph\_elem.draw\_line(

(self.graph\_current\_item, 0), (self.graph\_current\_item, percent\_sent), color=self.color)

self.graph\_lines.append(line\_id)

if self.graph\_current\_item >= GRAPH\_WIDTH:

self.graph\_elem.delete\_figure(self.graph\_lines.pop(0))

self.graph\_elem.move(-1, 0)

else:

self.graph\_current\_item += 1

return delta

def graph\_percentage\_abs(self, value):

self.graph\_elem.draw\_line(

(self.graph\_current\_item, 0), (self.graph\_current\_item, value), color=self.color)

if self.graph\_current\_item >= GRAPH\_WIDTH:

self.graph\_elem.move(-1, 0)

else:

self.graph\_current\_item += 1

def human\_size(bytes, units=(' bytes', 'KB', 'MB', 'GB', 'TB', 'PB', 'EB')):

""" Returns a human readable string reprentation of bytes"""

return str(bytes) + units[0] if bytes < 1024 else human\_size(bytes >> 10, units[1:])

**approx.py**

import numpy as np

import scipy as sp

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

from utils import read\_data

def drawing(x, y):

***plt.figure(figsize=(8, 6))***

***plt.title("Human - RAM usage")***

***plt.xlabel("Seconds")***

***plt.ylabel("Usages")***

***plt.scatter(x, y)***

    legend = []

    fx = sp.linspace(x[0], x[-1] + 60, 1000)

    for d in range(1, 4):

        fp, residuals, rank, sv, rcond = sp.polyfit(x, y, d, full=True)

        f = sp.poly1d(fp)

        plt.plot(fx, f(fx), linewidth=2)

        legend.append("d=%i" % f.order)

        f2 = f - 1000

        t = fsolve(f2, x[-1])

    plt.legend(legend, loc="upper left")

    plt.grid()

    plt.show()

***def main():***

***data = read\_data('human\_usage.json')***

***data\_value = [data[key] for key in data]***

***x = [x for x in range(len(data\_value[-1]))]***

***y = [y for y in data\_value[2]]***

***drawing(x, y)***

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**draw.py**

***import matplotlib.pyplot as plt***

***from utils import read\_data***

***import numpy as np***

***def moving\_avg(x, n):***

***cumsum = np.cumsum(np.insert(x, 0, 0))***

***return (cumsum[n:] - cumsum[:-n]) / float(n)***

***def main():***

***data = read\_data('bot\_usage.json')***

***data\_value = [data[key] for key in data]***

***data\_x = [x for x in range(len(data\_value[-1]))]***

***data\_y = [y for y in data\_value[2]]***

***avg\_y = moving\_avg(data\_y, 15)***

***avg\_x = [i for i in range(len(avg\_y))]***

***draw(data\_x, data\_y, avg\_x, avg\_y)***

***avg\_usage = round(sum(data\_y)/len(data\_y), 1)***

***print(f'BOT RAM: {avg\_usage}%')***

***def draw(data\_x, data\_y, avg\_x, avg\_y):***

***plt.figure()***

***plt.plot(data\_x, data\_y, linewidth=1, label='normal')***

***plt.plot(avg\_x, avg\_y, linewidth=4, label='average')***

***plt.xlabel('Seconds')***

***plt.ylabel('Usages')***

***plt.title('Bot - CPU usage')***

***plt.legend()***

***plt.grid()***

***plt.show()***

***if \_\_name\_\_ == '\_\_main\_\_':***

***main()***

**monitor.py**

***import time***

***import numpy as np***

***from utils import psutility, save\_data***

***DATA = {'CPU': [], 'GPU': [], 'RAM': []}***

***def write\_data(state):***

***global DATA***

***for key, value in zip(DATA.keys(), state):***

***DATA[key].append(round(value, 1))***

***def main(start=0, end=10, delay=1):***

***step = delay/60***

***for x in np.arange(start, end, step):***

***state = psutility()***

***write\_data(state)***

***save\_data(DATA)***

***time.sleep(delay)***

***if \_\_name\_\_ == '\_\_main\_\_':***

***main(0, 5, 1)***

**utils.py (analysis)**

***import psutil***

***import GPUtil***

***import json***

***def psutility():***

***cpu = psutil.cpu\_percent(0)***

***gpu = GPUtil.getGPUs()[-1].load\*100***

***ram = psutil.virtual\_memory().percent***

***return cpu, gpu, ram***

***def read\_data(filename):***

***with open(filename, 'r') as f:***

***data = json.load(f)***

***return data***

***def save\_data(state):***

***with open('monitor\_usage.json', 'w') as f:***

***json.dump(state, f, indent=4)***

**convert\_exe.sh**

***pyinstaller \***

***--onefile --windowed \***

***--add-data "static/config.json;/static/." --add-data "static/disconnect\_info.png;/static/." --add-data "static/window\_icon.ico;/static/." \***

***--hidden-import="utils" --hidden-import="dashboard" --hidden-import="windowcapture" --hidden-import="components" \***

***--name "AI Bot" \***

***main.py***

**requirements.txt**

***absl-py==0.15.0***

***altgraph==0.17.2***

***astunparse==1.6.3***

***cachetools==5.2.0***

***certifi==2022.6.15.1***

***charset-normalizer==2.1.1***

***cycler==0.11.0***

***flatbuffers==1.12***

***future==0.18.2***

***gast==0.3.3***

***google-auth==2.11.0***

***google-auth-oauthlib==0.4.6***

***google-pasta==0.2.0***

***grpcio==1.32.0***

***h5py==2.10.0***

***idna==3.3***

***imageai==2.1.6***

***importlib-metadata==4.12.0***

***Keras==2.4.3***

***Keras-Preprocessing==1.1.2***

***keras-resnet==0.2.0***

***kiwisolver==1.4.4***

***Markdown==3.4.1***

***MarkupSafe==2.1.1***

***matplotlib==3.3.2***

***numpy==1.19.3***

***oauthlib==3.2.1***

***opencv-python==4.6.0.66***

***opt-einsum==3.3.0***

***pefile==2022.5.30***

***Pillow==7.0.0***

***protobuf==3.9.2***

***psutil==5.9.2***

***pyasn1==0.4.8***

***pyasn1-modules==0.2.8***

***pyinstaller==5.3***

***pyinstaller-hooks-contrib==2022.10***

***pyparsing==3.0.9***

***PySimpleGUI==4.60.3***

***python-dateutil==2.8.2***

***pywin32==304***

***pywin32-ctypes==0.2.0***

***PyYAML==6.0***

***requests==2.28.1***

***requests-oauthlib==1.3.1***

***rsa==4.9***

***scipy***

***six==1.15.0***

***tensorboard==2.10.0***

***tensorboard-data-server==0.6.1***

***tensorboard-plugin-wit==1.8.1***

***tensorflow==2.4.0***

***tensorflow-estimator==2.4.0***

***termcolor==1.1.0***

***typing-extensions==3.7.4.3***

***urllib3==1.26.12***

***Werkzeug==2.2.2***

***wrapt==1.12.1***

***zipp==3.8.1***

**ДОДАТОК Б**

**Репозиторій:**

<https://github.com/Wizak/BIgProject.git>

**Файли ШІ:**

[https://drive.google.com/drive/folders/14WRtk8WBhtnuTLLVf\_2mkZNk5lX2AbS5?usp=sharing](https://drive.google.com/drive/folders/14WRtk8WBhtnuTLLVf_2mkZNk5lX2AbS5?usp=sharinghttps://drive.google.com/drive/folders/14WRtk8WBhtnuTLLVf_2mkZNk5lX2AbS5?usp=sharinghttps://drive.google.com/drive/folders/14WRtk8WBhtnuTLLVf_2mkZNk5lX2AbS5?usp=sharing)

**Відеодемонстрація:**

<https://youtu.be/xeejWRi8t00>