**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Львівський національний університет імені Івана Франка Факультет електроніки та комп’ютерних технологій Кафедра**

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:

**доц. Сидоренко С.С.**

Львів 2022

**АНОТАЦІЯ**

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.

Приклад з грою, всього лиш частковий випадок реалізації деякої ідеї щодо автоматизації процесу, у якому, безпосередньо, рішення приймає людина. У роботі представлена одна з описаного (в курсовій) набору програм (оптимальний варіант, на якому і надалі буде акцентована увага), принцип, якої базується на використанні ШІ для самостійного прийняття рішення. Для розробки програми обрано мову розробки Python, як потужний інструмент та використані технології для роботи із нейронними мережами, обробкою зображень тощо. Розроблено зручний віконний інтерфейс, для моніторингу активності, стану бота та з можливістю передавання керування на людину через голосовий канал.

У розділах описано теоретичні відомості. Розглянуто: методи обробки інформації, описано поняття та значення комп’ютерного зору (комп’ютерне бачення), розпізнавання мовлення людини, розробку програмного забезпечення та автоматизацію 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. Розповідається про реалізацію проєкту, його труднощі розробки та результати. Описано етапи розробки та структура проєкту. Розробка модулів та подальша інтеграція у програму.

В останньому розділі відбувається тестування, аналіз та представлення отриманих результатів, проводиться дослідження.

**ABSTRACT**

Assistive or preferential programs may be used in games to optimize gameplay. Also, the development of monitoring and analysis tools with subsequent implementation in the robotics control system shows the result on the example of Boston Dynamics. A software bot (robot) is a useful tool that is used, in particular, by both skilled players and not.

An example with a game is only a partial case of the implementation of some idea regarding the automation of a process in which, directly, a person makes a decision. The work presents one of the described (in the course) set of programs (the optimal option, which will continue to be the focus of attention), the principle of which is based on the use of AI for independent decision-making. To develop the program, the Python development language was chosen as a powerful tool and used technologies for working with neural networks, image processing, etc. A convenient window interface has been developed for monitoring the activity and status of the bot and with the possibility of transferring control to a person through the voice channel.

Theoretical information is described in the sections. Considered: information processing methods, the concept and meaning of computer vision (computer vision), human speech recognition, software development and process automation are described. Technologies and tools for project implementation are given. The task and solution architecture are being formed. Described: development of a window interface, methods of interaction with the operating system through API, use of libraries for working with audio and video streams, their processing and analysis. It tells about the implementation of the project, its development difficulties and results. The stages of development and the structure of the project are described. Development of modules and further integration into the program.

In the last section, testing, analysis and presentation of the obtained results take place, research is conducted.

**ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ**

ШІ Штучний інтелект  
ПК Персональний комп’ютер  
БД База даних  
ООП Об’єктноорієнтоване програмування  
АП Автоматизація процесу  
ТЗ Технічні засоби  
ПЗ Програмне забезпечення

Зміст

[**ВСТУП** 6](#_Toc120498814)

[РОЗДІЛ I. ОБРОБКА ІНФОРМАЦІЇ. АВТОМАТИЗАЦІЯ ПРОЦЕСІВ. РОЗРОБКА ПРОГРАМНИХ РІШЕНЬ 8](#_Toc120498815)

[**1.1.** **Методи обробки інформації** 8](#_Toc120498816)

[**1.2.** **Комп’ютерний зір** 14](#_Toc120498817)

[**1.3.** **Розпізнавання мовлення** 18](#_Toc120498818)

[**1.4.** **Задачі автоматизації** 21](#_Toc120498819)

[**1.5.** **Комп’ютерна система взаємодій** 24](#_Toc120498820)

[**1.6.** **Переваги мови Python** 28](#_Toc120498821)

[РОЗДІЛ II. ТЕХНОЛОГІЇ ПРОГРАМНОЇ РОЗРОБКИ ТА АНАЛІЗУ. СИСТЕМА ПРИЙНЯТТЯ РІШЕНЬ 29](#_Toc120498822)

[2.1. Постановка задачі та архітектура рішень 29](#_Toc120498823)

[2.2. Розробка віконних програм 29](#_Toc120498824)

[2.3. Програмна взаємодія з операційної системою 29](#_Toc120498825)

[2.4. Бібліотеки для роботи з фото, аудіо матеріалом 29](#_Toc120498826)

[2.5. Бібліотеки для роботи з технологіями розпізнавання 29](#_Toc120498827)

[2.6. Підготовка набору даних та навчання нейронної мережі 29](#_Toc120498828)

[РОЗДІЛ III. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. РЕАЛІЗАЦІЯ ПРОЄКТУ 30](#_Toc120498829)

[3.1. Створення прототипів та тестування 30](#_Toc120498830)

[3.2. Реалізація віконної програми з майбутнім функціоналом 30](#_Toc120498831)

[3.3. Написання модулів для роботи комп’ютерного зору та розпізнавання мовлення 30](#_Toc120498832)

[3.4. Підготовка моделі, тренування нейронної мережі 30](#_Toc120498833)

[3.5. Реалізація програмної взаємодії з грою 30](#_Toc120498834)

[3.6. Інтеграція роботи модулів у програмі та тестування 30](#_Toc120498835)

[РОЗДІЛ IV. ТЕСТУВАННЯ РОЗРОБКИ. ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ 31](#_Toc120498836)

[4.1. Запуск програми в тестовому середовищі 31](#_Toc120498837)

[4.2. Аналіз результатів тестування та корекція програми 31](#_Toc120498838)

[4.3. Збір даних із довготривалого користування в “бойовому” режимі 31](#_Toc120498839)

[4.4. Аналіз результатів дослідження 31](#_Toc120498840)

[ВИСНОВКИ 32](#_Toc120498841)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 33](#_Toc120498842)

[ДОДАТОК А 36](#_Toc120498843)

[**ДОДАТОК Б** 51](#_Toc120498844)

**ВСТУП**

Розвиток інформаційних технологій дає можливості створювати автоматизовані системи, що інтегруються у різні сфери людського життя. Технології задовільняють потреби людини, в умовах життя на нашій планеті та дають можливості розвитку шляхом розв`язання проблем. Розвиток мов програмування прискорює створення проєктів та виводить розробку на новий рівень, Python у цьому сенсі – це інструмент, що підійде краще інших, якщо задача полягає у реалізації ідеї та результаті.

Напрям розвитку інформаційних технологій приведе до повного автоматизування систем, послужить розвиток штучного інтелекту, який буде використовуватися для контролю процесів на виробництві, чим займалася скорше людина, прикладом є те, що продовж вже довго часу люди втрачають роботу, через те, що їхню спеціальність можна з легкістю замінити роботами. Штучні нейронні мережі - математична модель, яка програмується на штучних системах, яка дає змогу в рази затрачати менше ресурси на розв`язання проблем, це хороша альтернатива, яка має перспективу. Поєднання штучного інтелекту із мовою програмування Python дає змогу швидко та доволі просто спроєктувати і реалізувати деяку систему.

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 дляаналізу на основі, якої керують віртуально периферією ПК так і звичайними, що слідують одному статичному сценарію.

У роботі представлено різноманітні боти, вони побудовані за різним принципом взаємодії з ігровим середовищем. Автоклікер, це скрипт, що виконується незалежно від умов в грі. Чіт, це програма, яка працює з пам`яттю і бере дані з ігрового процесу, або змінює їх. Ці варіанти будуть розглянуті в роботі та буде визначено, який із них ефективний, функціональний чи реальний для реалізації.

**Ідея:** полегшення монотонної роботи задля економії сил та часу, які можуть бути відведені для цікавої частини гри, а також для допомоги у складних ігрових моментах. Ботів можна розміщувати у великій кількості на різних фізичних, чи віртуальних системах.

**Мета:** створення автоматизованих ботів для емуляції гравця в ігровому середовищі на основі підготовлених моделей для штучної нейромережі, використовуючи мову Python.

**Завдання:** ознайомитися із принципами роботи штучної нейронної мережі, особливостями мови Python, різними фреймворками для роботи із зображеннями, реалізувати програму для розпізнавання об’єктів та виділення їх на екрані та подальшої маніпуляції віртуальною периферією.

# РОЗДІЛ I. ОБРОБКА ІНФОРМАЦІЇ. АВТОМАТИЗАЦІЯ ПРОЦЕСІВ. РОЗРОБКА ПРОГРАМНИХ РІШЕНЬ

* 1. **Методи обробки інформації**

Обробка інформації, отримання, запис, організація, пошук, відображення та розповсюдження інформації. Останніми роками цей термін часто застосовують до комп’ютерних операцій.

У загальному вживанні термін «інформація» стосується фактів і думок, наданих і отриманих протягом повсякденного життя: людина отримує інформацію безпосередньо від інших живих істот, із засобів масової інформації, з електронних банків даних і з усіх видів спостережуваних явищ у навколишньому середовищі. навколишнє середовище. Людина, яка використовує такі факти та думки, генерує більше інформації, частина якої передається іншим під час розмови, інструкцій, у листах і документах, а також через інші засоби масової інформації. Інформацію, організовану відповідно до певних логічних зв’язків, називають сукупністю знань, які необхідно отримати шляхом систематичного ознайомлення або вивчення. Застосування знань (чи навичок) дає досвід, а додаткові аналітичні чи емпіричні ідеї вважаються прикладами мудрості. Використання терміну «інформація» не обмежується винятково її передачею за допомогою природної мови. Інформація також реєструється та передається за допомогою мистецтва, міміки та жестів або таких інших фізичних реакцій, як тремтіння. Крім того, кожна жива істота наділена інформацією у вигляді генетичного коду. Ці інформаційні явища пронизують фізичний і психічний світ, і їхня різноманітність така, що досі не піддається всім спробам уніфікованого визначення інформації.

Інтерес до інформаційних явищ різко зріс у 20-му столітті, і сьогодні вони є об’єктами вивчення в ряді дисциплін, включаючи філософію, фізику, біологію, лінгвістику, інформацію та комп’ютерні науки, електроніку та комунікаційну інженерію, науку про управління та соціальні науки. наук. З комерційного боку індустрія інформаційних послуг стала однією з найновіших галузей у всьому світі. Майже всі інші галузі — виробництво та сфера послуг — дедалі більше стурбовані інформацією та її обробкою. Різні, хоча часто і збігаються, точки зору та явища цих сфер призводять до різних (і іноді суперечливих) концепцій та «визначень» інформації.

Інтерес до того, як інформація передається і як її носії передають значення, з часів досократівських філософів займав сферу досліджень, звану семіотикою, вивчення знаків і знакових феноменів. Знаки є незвідними елементами комунікації та носіями значення. Американському філософу, математику та фізику Чарльзу С. Пірсу приписують те, що він вказав на три виміри знаків, які стосуються, відповідно, тіла або носія знака, об’єкта, який знак позначає, та інтерпретанта або тлумачення знака. Пірс визнав, що фундаментальні відносини інформації по суті триадичні; навпаки, усі відносини фізичних наук зводяться до діадичних (бінарних) відносин. Інший американський філософ, Чарльз В. Морріс, назвав ці три виміри знаків синтаксичним, семантичним і прагматичним, іменами, під якими вони відомі сьогодні.

Інформаційні процеси виконуються інформаційними процесорами. Для певного інформаційного процесора, фізичного чи біологічного, маркер є об’єктом, позбавленим сенсу, який процесор розпізнає як повністю відмінний від інших маркерів. Група таких унікальних маркерів, розпізнаних процесором, становить його базовий «алфавіт»; наприклад, крапка, тире та пробіл складають базовий алфавіт символів процесора азбуки Морзе. Об’єкти, що несуть значення, представлені шаблонами токенів, які називаються символами. Останні об’єднуються, щоб утворити символічні вирази, які є входами або виходами інформаційних процесів і зберігаються в пам’яті процесора.

Інформаційні процесори - це компоненти інформаційної системи, які є класом конструкцій. Абстрактна модель інформаційної системи містить чотири основні елементи: процесор, пам’ять, рецептор і ефектор (рис. 1). Процесор виконує декілька функцій: (1) виконувати елементарні інформаційні процеси над символьними виразами, (2) тимчасово зберігати в короткочасній пам’яті процесора вхідні та вихідні вирази, на яких ці процеси працюють і які вони генерують, (3) планувати виконання цих процесів і (4) змінювати цю послідовність операцій відповідно до вмісту короткочасної пам'яті. Пам'ять зберігає символічні вирази, в тому числі ті, що представляють складні інформаційні процеси, які називаються програмами. Два інших компоненти, рецептор і ефектор, є механізмами введення та виведення, функціями яких є, відповідно, отримання символічних виразів або стимулів із зовнішнього середовища для маніпулювання з боку процесора та випромінювання оброблених структур назад у середовище.

Потужність цієї абстрактної моделі системи обробки інформації забезпечується здатністю її складових процесорів виконувати невелику кількість елементарних інформаційних процесів: читання; порівнюючи; створення, модифікація та іменування; копіювання; зберігання; і написання. Модель, яка представляє широкий спектр таких систем, була визнана корисною для пояснення створених людиною інформаційних систем, реалізованих на послідовних інформаційних процесорах.

Оскільки було визнано, що в природі інформаційні процеси не є суворо послідовними, з 1980 року все більше уваги було зосереджено на вивченні людського мозку як інформаційного процесора паралельного типу. Когнітивні науки, міждисциплінарна галузь, яка зосереджується на вивченні людського розуму, зробили внесок у розробку нейрокомп’ютерів, нового класу паралельних розподілених інформаційних процесорів, які імітують функціонування людського мозку, включаючи його здатність до самоконтролю. організації та навчання. Так звані нейронні мережі, які є математичними моделями, натхненними мережею нейронних ланцюгів людського мозку, все частіше знаходять застосування в таких сферах, як розпізнавання образів, керування промисловими процесами та фінанси, а також у багатьох наукових дисциплінах.

Наприкінці 20 століття інформація набула двох основних утилітарних конотацій. З одного боку, він вважається економічним ресурсом, дещо нарівні з іншими ресурсами, такими як праця, матеріал і капітал. Ця точка зору випливає з доказів того, що володіння, маніпулювання та використання інформації може підвищити економічну ефективність багатьох фізичних і когнітивних процесів. Зростання діяльності з обробки інформації в промисловому виробництві, а також у вирішенні людських проблем було надзвичайним. Аналіз одного з трьох традиційних підрозділів економіки, сфери послуг, показує різке зростання інформаційно-інтенсивної діяльності з початку 20 століття. До 1975 року ці види діяльності становили половину робочої сили Сполучених Штатів.

Як індивідуальний і суспільний ресурс інформація має деякі цікаві характеристики, які відрізняють її від традиційних уявлень про економічні ресурси. На відміну від інших ресурсів, інформація є експансивною, з обмеженнями, очевидно, накладеними лише часом і когнітивними можливостями людини. Його експансивність пояснюється наступним: (1) він природно розповсюджується, (2) він відтворюється, а не споживається через використання, і (3) ним можна лише ділитися, а не обмінюватися в транзакціях. У той же час інформація стискається як синтаксично, так і семантично. У поєднанні з його здатністю замінювати інші економічні ресурси, його транспортабельністю на дуже високих швидкостях і його здатністю надавати переваги власнику інформації, ці характеристики лежать в основі таких галузей суспільства, як дослідження, освіта, видавництво, маркетинг, і навіть політика. Суспільна стурбованість використанням інформаційних ресурсів поширилася з традиційної сфери бібліотек і архівів, щоб охопити організаційну, інституційну та урядову інформацію під егідою управління інформаційними ресурсами.

Друге сприйняття інформації полягає в тому, що вона є економічним товаром, який допомагає стимулювати всесвітнє зростання нового сегмента національних економік — сектору інформаційних послуг. Використовуючи переваги властивостей інформації та спираючись на сприйняття її індивідуальної та суспільної корисності та цінності, цей сектор надає широкий спектр інформаційних продуктів і послуг. До 1992 року частка ринку інформаційних послуг США зросла приблизно до 25 мільярдів доларів. Це було еквівалентно приблизно одній сьомій комп’ютерного ринку країни, який, у свою чергу, становив приблизно 40 відсотків світового ринку комп’ютерів того року. Однак можливе зближення комп’ютерів і телебачення (частка ринку якого в 100 разів більша, ніж комп’ютери) і його вплив на інформаційні послуги, розваги та освіту, швидше за все, реструктуризують відповідні частки ринку інформаційної індустрії.

Люди сприймають інформацію за допомогою органів чуття: звуки — через слух; зображення та текст через зір; форма, температура та відчуття через дотик; і запахи через запах. Щоб інтерпретувати сигнали, отримані від органів чуття, люди розробили та вивчили складні системи мов, що складаються з «алфавітів» символів і стимулів, а також відповідних правил використання. Це дозволило їм розпізнавати предмети, які вони бачать, розуміти повідомлення, які вони читають або чують, і розуміти знаки, отримані через тактильні та нюхові відчуття.

Носіями інформаційних знаків, що сприймаються органами чуття, є енергетичні явища — звукові хвилі, світлові хвилі, хімічні та електрохімічні подразники. Говорячи інженерною мовою, люди є приймачами аналогових сигналів; і, за дещо вільною угодою, повідомлення, що передаються через ці носії, називаються інформацією в аналоговій формі або просто аналоговою інформацією. До появи цифрового комп’ютера когнітивна інформація зберігалася й оброблялася лише в аналоговій формі, в основному за допомогою технологій друку, фотографії та телефонії.

Хоча люди вправно обробляють інформацію, що зберігається в їхній пам’яті, аналогова інформація, що зберігається поза розумом, нелегко обробляється. Сучасні інформаційні технології значно полегшують маніпулювання інформацією, що зберігається назовні, в результаті її представлення у вигляді цифрових сигналів, тобто у вигляді наявності або відсутності енергії (електрики, світла чи магнетизму). Інформацію, представлену в цифровому вигляді в двокомпонентній або двійковій формі, часто називають цифровою інформацією. Сучасні інформаційні системи характеризуються великими метаморфозами аналогової та цифрової інформації. Що стосується зберігання інформації та комунікації, то перехід від аналогової інформації до цифрової є настільки поширеним, що приносить історичну трансформацію того, як люди створюють, отримують доступ і використовують інформацію.

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).

Таким чином, алфавітні символи систем письма природної мови можуть бути представлені цифровим способом у вигляді комбінацій нулів (без пульсу) та одиниць (імпульс). Таблиці еквівалентності буквено-цифрових символів і рядків двійкових цифр називаються системами кодування, відповідниками систем письма. Комбінація трьох двійкових цифр може представляти до восьми таких символів; один із чотирьох цифр, до 16 символів; і так далі. Вибір конкретної системи кодування залежить від розміру набору символів, який потрібно представити. Широко використовувані системи: американський стандартний код для обміну інформацією (ASCII), семи- або восьмибітний код, що представляє англійський алфавіт, цифри та деякі спеціальні символи стандартної комп’ютерної клавіатури; і відповідний восьмибітовий розширений двійковий кодований десятковий код обміну (EBCDIC), який використовується для комп’ютерів виробництва IBM (International Business Machines Corp.) і більшості сумісних систем. Цифрове представлення символу вісьмома бітами називається байтом.

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. **Комп’ютерний зір**

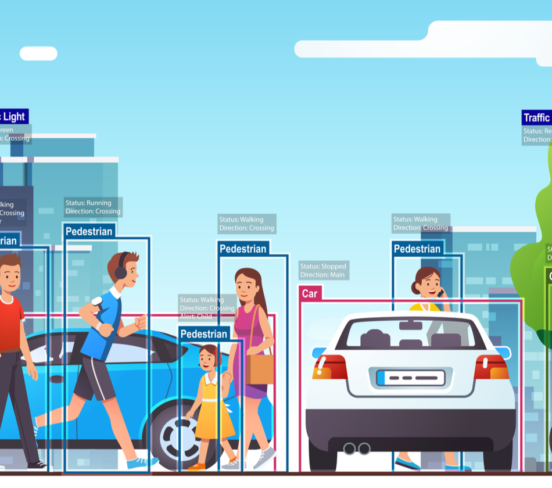
Обробка інформації, отримання, запис, комп’ютерний зір — це сфера штучного інтелекту (ШІ), яка дозволяє комп’ютерам і системам отримувати значущу інформацію з цифрових зображень, відео та інших візуальних вхідних даних — і вживати дій або давати рекомендації на основі цієї інформації. Якщо ШІ дозволяє комп’ютерам мислити, комп’ютерний зір дозволяє їм бачити, спостерігати та розуміти.

Комп’ютерний зір працює майже так само, як і людський, за винятком того, що люди мають перевагу. Людський зір має перевагу тривалості контексту, щоб навчитися розрізняти об’єкти, як далеко вони знаходяться, чи рухаються вони та чи є щось не так на зображенні.

Комп’ютерний зір навчає машини виконувати ці функції, але робити це потрібно набагато швидше за допомогою камер, даних і алгоритмів, а не сітківки, зорових нервів і зорової кори. Оскільки система, навчена перевіряти продукти або спостерігати за виробничим активом, може аналізувати тисячі продуктів або процесів за хвилину, помічаючи непомітні дефекти чи проблеми, вона може швидко перевершити людські можливості.

Комп’ютерне бачення (див. Рис. 1.2.1) використовується в різних галузях промисловості, від енергетики та комунальних послуг до виробництва та автомобілебудування – і ринок продовжує зростати. Очікується, що до 2022 року він досягне 48,6 млрд доларів США.

Комп’ютерний зір потребує великої кількості даних. Він проводить аналіз даних знову і знову, доки не розпізнає відмінності та остаточно розпізнає зображення. Наприклад, щоб навчити комп’ютер розпізнавати автомобільні шини, йому потрібно надати величезну кількість зображень шин і предметів, пов’язаних із шинами, щоб дізнатися про відмінності та розпізнати шину, особливо без дефектів.



**Рис. 1.2.1.** Комп’ютерний зір.

Машинне навчання використовує алгоритмічні моделі, які дозволяють комп’ютеру навчитися контексту візуальних даних. Якщо через модель подано достатньо даних, комп’ютер «перегляне» дані та навчиться відрізняти одне зображення від іншого. Алгоритми дозволяють машині навчатися самостійно, а не хтось програмує її розпізнавати зображення.

CNN допомагає моделі машинного або глибокого навчання «виглядати», розбиваючи зображення на пікселі, яким присвоюють теги або мітки. Він використовує мітки для виконання згорток (математична операція над двома функціями для отримання третьої функції) і робить прогнози щодо того, що він «бачить». Нейронна мережа запускає згортки та перевіряє точність своїх прогнозів у серії ітерацій, доки прогнози не почнуть збуватися. Тоді це розпізнавання або бачення зображень у спосіб, подібний до людей.

Подібно до того, як людина створює зображення на відстані, CNN спочатку розрізняє жорсткі грані та прості форми, а потім заповнює інформацію, коли виконує ітерації своїх прогнозів. CNN використовується для розуміння окремих зображень. Повторювана нейронна мережа (RNN) використовується подібним чином для відеозастосунків, щоб допомогти комп’ютерам зрозуміти, як зображення в серії кадрів пов’язані одне з одним.

Багато організацій не мають ресурсів для фінансування лабораторій комп’ютерного зору та створення моделей глибокого навчання та нейронних мереж. Їм також може бракувати обчислювальної потужності, необхідної для обробки величезних наборів візуальних даних. Такі компанії, як IBM, допомагають, пропонуючи послуги з розробки програмного забезпечення комп’ютерного зору. Ці сервіси надають готові моделі навчання, доступні з хмари, а також зменшують попит на обчислювальні ресурси. Користувачі підключаються до служб через інтерфейс прикладного програмування (API) і використовують їх для розробки програм комп’ютерного зору.

IBM також представила платформу комп’ютерного бачення, яка вирішує питання як розвитку, так і обчислювальних ресурсів. IBM Maximo Visual Inspection містить інструменти, які дають змогу експертам із предметної тематики маркувати, навчати та розгортати бачні моделі глибокого навчання — без програмування чи досвіду глибокого навчання. Моделі бачення можна розгортати в локальних центрах обробки даних, у хмарі та периферійних пристроях.

Незважаючи на те, що стає легше отримати ресурси для розробки програм комп’ютерного зору, важливе запитання, на яке потрібно відповісти на ранній стадії: що саме ці програми будуть робити? Розуміння та визначення конкретних завдань комп’ютерного зору може зосередити та перевірити проекти та програми та полегшити початок роботи.

Ось кілька прикладів усталених завдань комп’ютерного зору:

* Класифікація зображень бачить зображення та може його класифікувати (собака, яблуко, обличчя людини). Точніше, він здатний точно передбачити, що дане зображення належить до певного класу. Наприклад, компанія соціальних медіа може використовувати його для автоматичної ідентифікації та відокремлення небажаних зображень, завантажених користувачами;
* Виявлення об’єктів може використовувати класифікацію зображень для ідентифікації певного класу зображень, а потім виявлення та таблиці їх появи на зображенні чи відео. Приклади включають виявлення пошкоджень на складальній лінії або ідентифікацію обладнання, яке потребує технічного обслуговування;
* Відстеження об’єкта стежить за об’єктом після його виявлення. Це завдання часто виконується за допомогою послідовних зображень або відео в реальному часі. Автономні транспортні засоби, наприклад, повинні не тільки класифікувати та виявляти такі об’єкти, як пішоходи, інші автомобілі та дорожня інфраструктура, вони повинні відстежувати їх рух, щоб уникнути зіткнень і дотримуватися правил дорожнього руху;
* Пошук зображень на основі вмісту використовує комп’ютерний зір для перегляду, пошуку та отримання зображень із великих сховищ даних на основі вмісту зображень, а не пов’язаних із ними тегів метаданих. Це завдання може включати автоматичне анотування зображення, яке замінює ручне тегування зображення. Ці завдання можна використовувати для систем управління цифровими активами та можуть підвищити точність пошуку та вилучення.

Вчені та інженери вже близько 60 років намагаються розробити способи, за допомогою яких машини зможуть бачити та розуміти візуальні дані. Експерименти почалися в 1959 році, коли нейрофізіологи показали кішці низку зображень, намагаючись співвіднести реакцію в її мозку. Вони виявили, що він спочатку реагує на жорсткі краї або лінії, і з наукової точки зору це означає, що обробка зображень починається з простих форм, таких як прямі краї.(2)

Приблизно в той же час була розроблена перша технологія комп’ютерного сканування зображень, яка дозволила комп’ютерам оцифровувати та отримувати зображення. Ще одна віха була досягнута в 1963 році, коли комп’ютери змогли перетворити двовимірні зображення в тривимірні форми. У 1960-х роках штучний інтелект став академічною галуззю дослідження, і це також поклало початок пошукам штучного інтелекту щодо вирішення проблеми людського зору.

У 1974 році була представлена технологія оптичного розпізнавання символів (OCR), яка могла розпізнавати текст, надрукований будь-яким шрифтом або гарнітурою. (3) Подібним чином інтелектуальне розпізнавання символів (ICR) могло розшифровувати рукописний текст за допомогою нейронних мереж. (4) Відтоді , OCR та ICR знайшли свій шлях до обробки документів і рахунків-фактур, розпізнавання номерних знаків транспортних засобів, мобільних платежів, машинного перекладу та інших поширених програм.

У 1982 році нейробіолог Девід Марр встановив, що зір працює ієрархічно, і запровадив алгоритми для машин, щоб виявляти краї, кути, криві та подібні основні форми. Паралельно комп’ютерний вчений Куніхіко Фукусіма розробив мережу клітин, які можуть розпізнавати шаблони. Мережа під назвою Neocognitron включала згорткові шари в нейронну мережу.

До 2000 року основна увага приділялася розпізнаванню об’єктів, а до 2001 року з’явилися перші програми для розпізнавання обличчя в реальному часі. Стандартизація того, як набори візуальних даних позначаються тегами та анотуються, з’явилася в 2000-х роках. У 2010 році набір даних ImageNet став доступним. Він містив мільйони позначених тегами зображень у тисячі класів об’єктів і забезпечує основу для CNN і моделей глибокого навчання, які використовуються сьогодні. У 2012 році команда з Університету Торонто взяла участь у конкурсі CNN на розпізнавання зображень. Модель під назвою AlexNet значно знизила частоту помилок при розпізнаванні зображень. Після цього прориву рівень помилок впав лише до кількох відсотків.

* 1. **Розпізнавання мовлення**

Розпізнавання мовлення або перетворення мовлення в текст — це здатність машини або програми ідентифікувати слова, сказані вголос, і перетворювати їх на читабельний текст. Просте програмне забезпечення для розпізнавання мовлення має обмежений словниковий запас і може розпізнавати слова та фрази, лише якщо вони вимовлені чітко. Більш складне програмне забезпечення може обробляти природне мовлення, різні акценти та різні мови.



**Рис. 1.3.1.** Приклад віртуального асистента.

Для розпізнавання мовлення використовується широкий спектр досліджень у галузі інформатики, лінгвістики та комп’ютерної техніки. Багато сучасних пристроїв і програм, орієнтованих на текст, мають функції розпізнавання мовлення, що дозволяє легше або без рук використовувати пристрій.

Розпізнавання мовлення та розпізнавання голосу — це дві різні технології, і їх не слід плутати:

* Розпізнавання мовлення використовується для ідентифікації слів усної мови;
* Розпізнавання голосу – це біометрична технологія ідентифікації голосу людини.

Системи розпізнавання мовлення використовують комп’ютерні алгоритми для обробки та інтерпретації вимовлених слів і перетворення їх у текст. Програма перетворює звук, записаний мікрофоном, на письмову мову, зрозумілу комп’ютерам і людям, виконавши такі чотири кроки:

* Аналізувати аудіозапис;
* Розбити його на частини;
* Оцифрувати його у формат, який читається комп’ютером; і
* Використовуйте алгоритм, щоб зіставити його з найбільш підходящим представленням тексту.

Програмне забезпечення для розпізнавання мовлення має адаптуватися до надзвичайно мінливої та залежно від контексту природи людської мови. Алгоритми програмного забезпечення, які обробляють і організовують аудіо в текст, навчаються на різних моделях мовлення, стилях мовлення, мовах, діалектах, акцентах і фразах. Програмне забезпечення також відокремлює розмовний звук від фонового шуму, який часто супроводжує сигнал.

Щоб відповідати цим вимогам, системи розпізнавання мовлення використовують два типи моделей.

Системи розпізнавання мовлення мають чимало застосувань. Ось їх вибірка:

* Мобільні пристрої. Смартфони використовують голосові команди для маршрутизації викликів, обробки мови в текст, голосового набору та голосового пошуку. Користувачі можуть відповідати на повідомлення, не дивлячись на свої пристрої. На Apple iPhone функція розпізнавання мовлення забезпечує роботу клавіатури та Siri, віртуального помічника. Функціональність також доступна додатковими мовами. Розпізнавання мовлення також можна знайти в програмах для обробки текстів, таких як Microsoft Word, де користувачі можуть диктувати слова, які потрібно перетворити на текст;
* Акустичні моделі. Вони представляють зв’язок між мовними одиницями мови та звуковими сигналами;
* Мовні моделі. Тут звуки зіставляються з послідовністю слів, щоб розрізняти слова, які схожі за звучанням;
* Освіта. Програмне забезпечення для розпізнавання мовлення використовується під час навчання мови. Програмне забезпечення чує мову користувача та пропонує допомогу з вимовою;
* Обслуговування клієнтів. Автоматизовані голосові помічники слухають запити клієнтів і надають корисні ресурси;
* Додатки для охорони здоров'я. Лікарі можуть використовувати програмне забезпечення для розпізнавання мовлення, щоб транскрибувати нотатки в медичні записи в режимі реального часу;
* Допомога по інвалідності. Програмне забезпечення для розпізнавання мовлення може перекладати вимовлені слова в текст за допомогою закритих субтитрів, щоб дозволити людині з втратою слуху зрозуміти, що говорять інші. Розпізнавання мовлення також може дозволити людям з обмеженим використанням рук працювати з комп’ютером, використовуючи голосові команди замість введення;
* Судова звітність. Програмне забезпечення можна використовувати для транскрибування судових засідань, виключаючи потребу в транскрипторах;
* Розпізнавання емоцій. Ця технологія може аналізувати певні вокальні характеристики, щоб визначити, які емоції відчуває оратор. У поєднанні з аналізом настроїв це може виявити, як хтось ставиться до продукту чи послуги;
* Спілкування без рук. Водії використовують голосове керування, наприклад, для спілкування без рук, керування телефонами, радіо та системами глобального позиціонування.

Хороші програми розпізнавання мовлення дозволяють користувачам налаштувати їх відповідно до своїх потреб. Функції, які це дозволяють, включають:

* Мовне зважування. Ця функція вказує алгоритму приділяти особливу увагу певним словам, наприклад тим, які часто вимовляються або є унікальними для розмови чи теми. Наприклад, програмне забезпечення можна навчити отримувати посилання на конкретні продукти;
* Акустична підготовка. Програмне забезпечення налаштовує навколишній шум, який забруднює голосовий звук. Програмне забезпечення з акустичним навчанням може розрізнити стиль, темп і гучність розмови серед галасу багатьох людей, які говорять в офісі;
* Маркування динаміків. Ця можливість дозволяє програмі позначати окремих учасників і визначати їхній конкретний внесок у розмову;
* Фільтрація ненормативної лексики. Тут програмне забезпечення відфільтровує небажані слова та мову.

Потужність функцій розпізнавання мовлення походить від набору алгоритмів і технологій. Вони включають наступне:

* Прихована модель Маркова. HMM використовуються в автономних системах, де стан можна спостерігати частково або коли вся інформація, необхідна для прийняття рішення, недоступна відразу для датчика (у випадку розпізнавання мови, мікрофона). Прикладом цього є акустичне моделювання, де програма повинна зіставляти мовні одиниці з аудіосигналами за допомогою статистичної ймовірності;
* Обробка природної мови. НЛП полегшує і прискорює процес розпізнавання мови;
* N-грами. Цей простий підхід до мовних моделей створює розподіл ймовірностей для послідовності. Прикладом може бути алгоритм, який розглядає кілька останніх сказаних слів, приблизно оцінює історію зразка мовлення та використовує це для визначення ймовірності наступного слова чи фрази, які будуть сказані;
* Штучний інтелект. Штучний інтелект і методи машинного навчання, як-от глибоке навчання та нейронні мережі, поширені в розширеному програмному забезпеченні розпізнавання мовлення. Ці системи використовують граматику, структуру, синтаксис і композицію звукових і голосових сигналів для обробки мови. Системи машинного навчання отримують знання з кожним використанням, завдяки чому вони добре підходять для таких нюансів, як акценти.

Розпізнавання мовлення – це технологія, що розвивається. Це один із багатьох способів, якими люди можуть спілкуватися з комп’ютером, практично не вводячи текст. Різноманітні комунікаційні бізнес-додатки використовують зручність і швидкість усного спілкування, які забезпечує ця технологія. Програми розпізнавання мовлення значно вдосконалилися за 60 років розробки. Вони все ще вдосконалюються, зокрема завдяки ШІ.

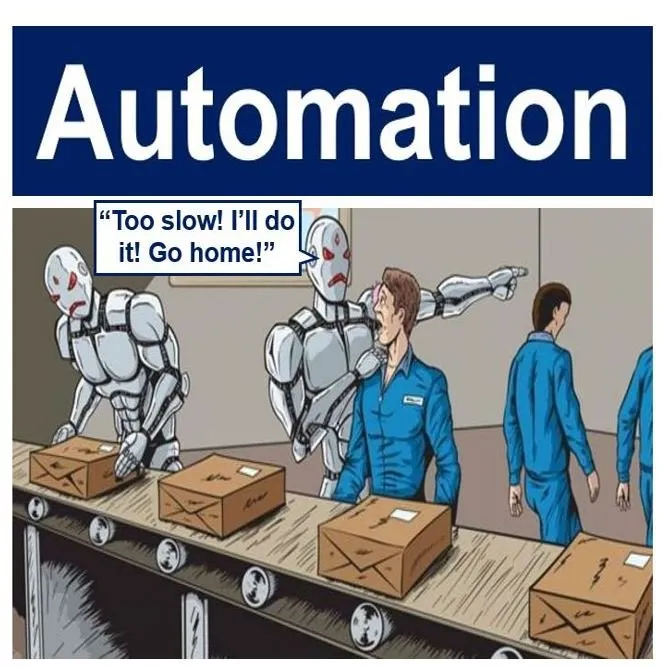
* 1. **Задачі автоматизації**

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. 1.4.1). 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.

Сьогодні автоматизація існує в усіх галузевих функціях, включаючи інтеграцію, установку, закупівлю, технічне обслуговування та навіть маркетинг і продажі.



**Рис. 1.4.1.** 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:

*«Ми наближаємось до того часу, коли машини зможуть перевершувати людей майже у будь-якому завданні. Я вважаю, що суспільство має відповісти на це питання, перш ніж воно постане перед нами: якщо машини здатні виконувати майже будь-яку роботу, яку можуть виконувати люди, що робитимуть люди?»*

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). Далі йдеться:

*«Оскільки взаємодія людини та комп’ютера вивчає людину та машину в спілкуванні, вона спирається на допоміжні знання як з боку машини, так і з боку людини. З боку машини актуальні методи комп’ютерної графіки, операційних систем, мов програмування та середовищ розробки. . З боку людини актуальними є теорія комунікації, дисципліни графічного та промислового дизайну, лінгвістика, соціальні науки, когнітивна психологія, соціальна психологія та людські фактори, такі як задоволеність користувача комп’ютера. І, звичайно, актуальними є методи інженерії та дизайну». [4]*

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].

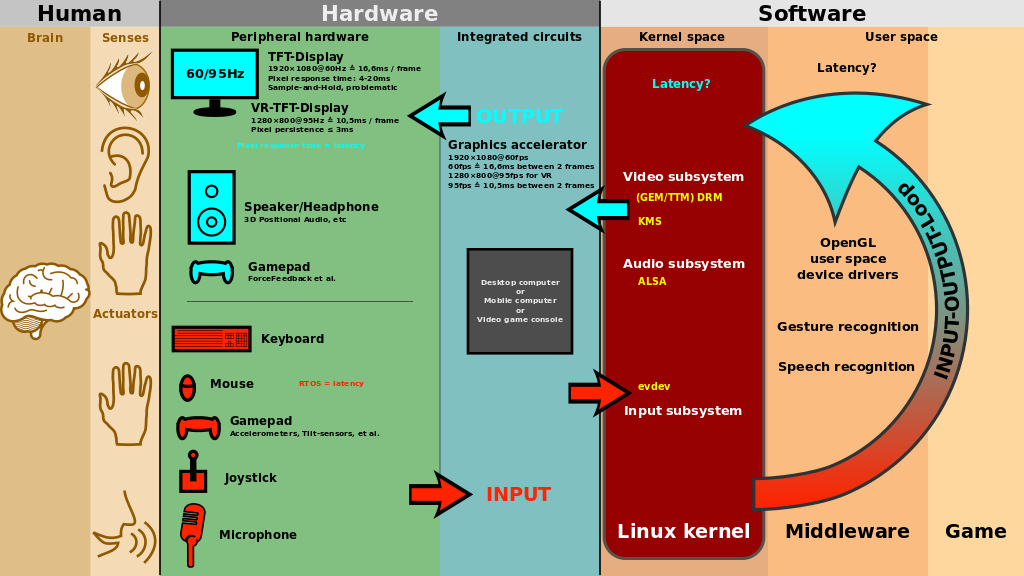
Бачення того, чого дослідники в цій галузі прагнуть досягти, може відрізнятися. Переслідуючи когнітивістську точку зору, дослідники HCI можуть прагнути узгодити комп’ютерні інтерфейси з ментальною моделлю своєї діяльності, яку мають люди. Переслідуючи посткогнітивістську перспективу, дослідники HCI можуть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

Дослідники HCI зацікавлені в розробці методологій проектування, експериментуванні з пристроями, створенні прототипів програмного та апаратного забезпечення, дослідженні парадигм взаємодії та розробці моделей і теорій взаємодії.

Під час оцінки поточного інтерфейсу користувача або розробки нового інтерфейсу користувача враховуються такі принципи експериментального дизайну (див. рис. 1.5.1):

* 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.

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



**Рис. 1.5.1.** Взаємодія користувача із обладненням.

* 1. **Переваги мови Python**

Python (див. рис. 1.1) проста у використанні, та водночас повноцінна мова програмування, що надає набагато більше засобів для структурування і підтримки великих програм, ніж shell. З іншого боку, вона краще за С обробляє помилки, і, будучи мовою дуже високого рівня, має вбудовані типи даних високого рівня, такі як гнучкі масиви і словники, ефективна реалізація яких на C потребує значних витрат часу.

Завдяки більш загальним типам даних, Python застосовують до більш широкого кола задач, ніж Awk і навіть Perl, у той ж час багато речей на мові Python робляться настільки ж просто.Python дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах. Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм або як приклади при вивченні мови. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережними з'єднаннями і навіть інтерфейсами до різних графічних бібліотек.



**Рис. 1.6.1.** Логотип Python.

Python - інтерпретована мова, що дозволяє заощадити значну кількість часу, що зазвичай витрачається на компіляцію. Інтерпретатор можна використовувати інтерактивно, що дозволяє експериментувати з можливостями мови, писати шаблони програм або тестувати функції при розробці “знизу-вверх”. Він також зручний як настільний калькулятор. Python дозволяє писати дуже компактні й зручні для читання програми. Програми, написані мовою Python, звичайно значно коротші еквівалента на C або C++ з декількох причин:

* Типи даних високого рівня дозволять Вам виразити складні операції однією інструкцією;
* Групування інструкцій виконується за допомогою відступів замість фігурних дужок;
* Немає необхідності в оголошенні змінних.

Python розширювана мова: знання C дозволяє додавати нові функції, що вбудовуються, або модулі для виконання критичних операцій з максимальною швидкістю або написання інтерфейсу до комерційних бібліотек, доступним тільки у двійковій формі. Інтерпретатор мови Python може бути вбудований у програму, написану на C, і використовувати його як розширення або командну мову для цієї програми. Python використовується в даний час десятками тисяч програмістів в усьому світі, і число людей, що використовують його, швидко зростає, подвоюється і потроюється щороку. Python приваблює користувачів з ряду причин. Він використовується для розробки програм і дозволяє провести розробку набагато швидше, ніж традиційні мови типу C, C++ або Java. Ця мова працює однаково добре на Windows, UNIX, Macintosh, і OS/2, може використовуватися, для легкої розробки як малих додатків чи сценаріїв, так і для розгортання великих програм. Python пропонує доступ до могутнього і легкого у використанні комплекту 29 інструментальних засобів графічного інтерфейсу користувача. Традиційні машинні мови типу C і Pascal мають ряд характеристик, наприклад, сувора типізація, базові типи, складні (і звичайно довгі) цикли, і потреба у великих кількостях кодів для виконання відносно малих задач. Java досить новий, але розділяє більшість характеристик, включених у цей перелік. Програмісти, знайомі з традиційними мовами погодяться, що відсутність суворої типізації полегшує роботу з Python.

Відмінностей Python від інших мов доволі багато, перерахуємо основні з них:

* Керування пам'яттю - цілком автоматичне — не потрібно хвилюватися щодо розподілу або звільнення пам'яті. Немає загрози “небезпечного посилання”. Java - єдина мова, що пропонує таку концепцію;
* Типи зв'язані з об'єктами, а не зі змінними. Це означає, що змінній може бути призначене значення будь-якого типу, і що (наприклад) масив може містити об'єкти різних типів. Традиційні мови не надають такої можливості;
* Операції звичайно виконуються в більш високому рівні абстракції.

Це частково результат того, як написана мова, і частково результат розширеної стандартної бібліотеки кодів, що поставляється разом з Python. Ці та інші особливості Python роблять розгортання додатків надзвичайно швидким. Продуктивність створеного додатку залежить від його особливостей. Звичайно, для чисельного алгоритму, що виконує звичайну арифметику цілого числа в циклі 'for', неважливо, на якій мові він написаний. Але для “середнього” додатка, збільшення продуктивності може бути просто дивовижним. Один недолік Python, у порівнянні з найбільш традиційними мовами, полягає в тому, що це - не цілком компільована мова; замість цього, вона частково транслює програму до внутрішньої форми байт-коду, і цей байт-код виконується інтерпретатором Python. Однак, у перспективі – сучасні комп'ютери мають так багато невикористовуваного обчислювального потенціалу, що для 90% додатків швидкодія зв'язана з вибором мови. Java теж компілюється в байт-код, але в даний час працює повільніше ніж Python у більшості випадків. Крім того, дуже просто об'єднати Python з модулями, написаними на C або C++, які можна використовувати, щоб збільшити швидкість роботи програм в критичних ділянках.

Розробка програм з використанням ШІ відрізняється від звичайної розробки. Для роботи з ШІ потрібен спеціальний стек технологій та спеціальні навички. Крім того, створення додатків на основі ШІ потребує глибоких досліджень. Для реалізації ваших ідей, пов'язаних із застосуванням ШІ, вам знадобиться надійна, гнучка мова програмування з багатим інструментарієм. Python — саме така мова, і тому сьогодні на ній розробляється безліч проектів ШІ. Python допомагає розробникам працювати продуктивно та впевнено, причому на всіх стадіях проекту, від розробки до підтримки. Ця мова має певні характеристики, які роблять її найкращим вибором для ML- та ШІ-проектів: він простий і логічний, гнучкий і мультиплатформний, має відмінні бібліотеки та фреймворки для машинного навчання та роботи з ІІ, а ще за ним стоїть численна спільнота розробників. Завдяки цьому Python є однією з найпопулярніших мов програмування у світі, причому не тільки у сфері ML та ШІ.

У Python легко писати лаконічний і читальний код. Незважаючи на те, що за машинним навчанням та штучним інтелектом стоять складні алгоритми та процеси, простота Python дозволяє створювати надійні системи. Розробники можуть повністю зосередитись на завданнях, які вони намагаються вирішити за допомогою ML, не відволікаючись на технічні нюанси мови. Крім того, Python простий у вивченні, чим і приваблює багатьох розробників. Написаний у ньому код легко зрозумілий людині, що спрощує створення моделей машинного навчання. Хтось вважає, що Python куди інтуїтивніший за інші мови програмування. Хтось вказує на наявність безлічі фреймворків, бібліотек та розширень, що спрощують реалізацію задуманих функцій. І всі погоджуються, що Python добре підходить для командної розробки. Оскільки Python - мова загального призначення, за його допомогою можна вирішувати багато складних завдань машинного навчання і швидко створювати прототипи для подальшого налагодження.

Створення алгоритмів ШІ та машинного навчання - це складне завдання, що вимагає багато часу. І щоб було легше знаходити оптимальні шляхи вирішення завдань, програмістам потрібне добре структуроване та надійне середовище розробки. Численні фреймворки та бібліотеки Python допомагають суттєво зменшити кількість часу, необхідного для розробки програм. Програмно бібліотека це попередньо написаний код, який розробники використовують для вирішення найпоширеніших завдань. Python, з його багатим стеком технологій, має великий набір бібліотек для штучного інтелекту та машинного навчання.

Ось деякі з них:

* Keras, TensorFlow та Scikit-learn — для машинного навчання ;
* NumPy — для високопродуктивних наукових обчислень та аналізу даних;
* SciPy - для просунутих обчислень ;

Scikit-learn пропонує різні алгоритми класифікації, регресії та кластеризації, включаючи допоміжні векторні машини, випадкові ліси, підвищення градієнта, k-середніх та DBSCAN. Ця бібліотека призначена для роботи з числовими та науковими бібліотеками Python NumPy та SciPy. За допомогою цих готових рішень ви зможете створювати свої продукти набагато швидше. Вашій команді розробників не доведеться знову винаходити велосипед, тому що для реалізації необхідних функцій вони зможуть використовувати існуючі бібліотеки.

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

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

У курсовій роботі 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, що і призвело до написання, як курсової так і дипломної роботи.

Проект мав підстави розвиватися і надалі, адже програма без взаємодії з людиною не могла бути користувацькою, так як не було інтерфейсної частини. Ідеєю цієї дипломної було розвиток проекту курсової, 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

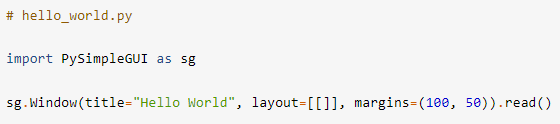
PySimpleGUI обгортає частини кожного з цих пакетів і полегшує їх використання. Однак кожен із портів потрібно встановлювати окремо.

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.

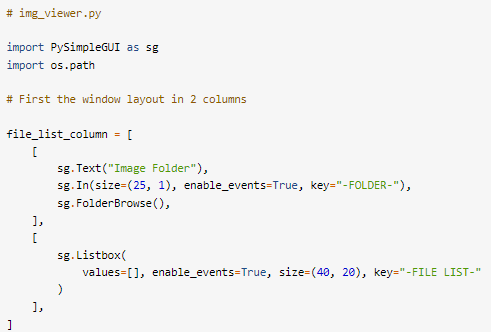
З іншого боку, якщо ви встановите порт PySimpleGUIQt, ви побачите, що Qt підтримує більше форматів зображень із коробки, ніж Tkinter. Ось макет того, як має виглядати переглядач зображень у кінці (див. рис. 2.2.3):



**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):



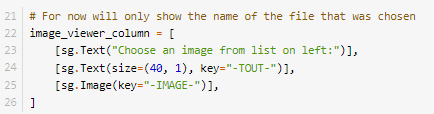
**Рис. 2.2.4.** Компоненти вікна.

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.

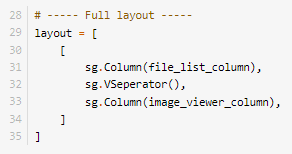
Коли ви вперше завантажуєте свій інтерфейс користувача, ви хочете, щоб Listbox() був порожнім, тому ви передаєте йому порожній список. Ви вмикаєте події для цього елемента, установлюєте його розмір і надаєте йому унікальний ідентифікатор, як ви робили з елементом input. Тепер ви можете подивитися на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



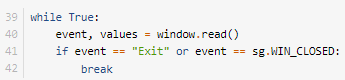
**Рис. 2.2.6.** Приклад компонування.

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):



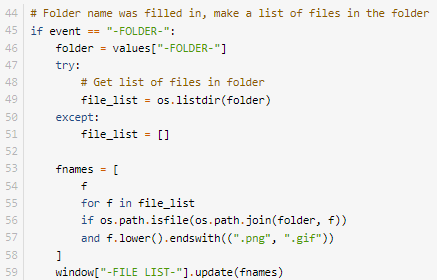
**Рис. 2.2.7.** Компонування для вікна.

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):



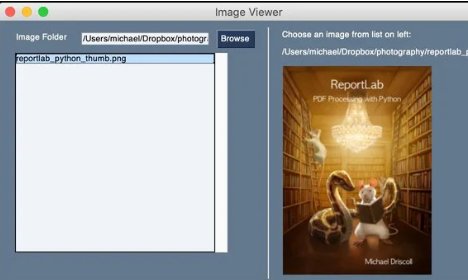
**Рис. 2.2.10.** Умовний вираз.

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):



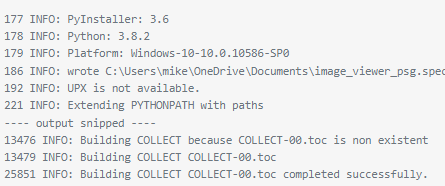
**Рис. 2.2.13.** Віконна програма для захоплення відеопотоку з камери.

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.

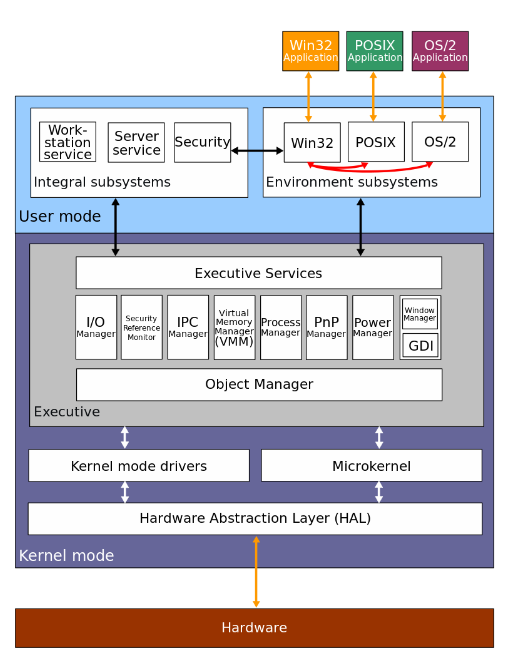
* 1. **Програмна взаємодія з операційної системою**

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

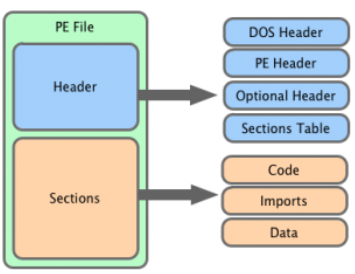
Не всі DLL створюються однаково. На додаток до стандартних DLL, які складають Windows API, існує набір DLL, які складають Native API. Нативний API працює так само, як Windows API, але є кілька ключових відмінностей. Власний API розроблено для підтримки меншого набору основних функцій до завантаження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



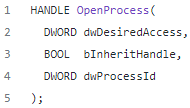
**Рис. 2.3.2.** Блок-схема архітектури Windows 2000.

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.

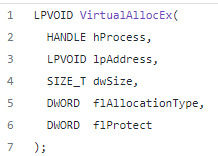
Усі функції, які можна імпортувати, мають бути десь визначені; у файлах, які визначають ці експортовані функції. Експорт функцій — це просто спосіб зробити функціональні можливості цих функцій доступними для використання іншими файлами. На фундаментальному рівні Windows API — це велика колекція файлів,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

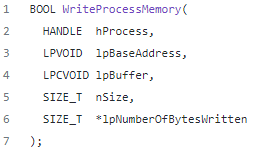
* dwDesiredAccess — визначає доступ, який буде призначено повернутому дескриптору процесу. Значення цього параметра можна знайти тут;
* bInheritHandle — дозволяє всім підпроцесам, створеним цим процесом, використовувати один дескриптор;
* dwProcessId — ідентифікатор процесу (PID) запущеного процесу.



**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:

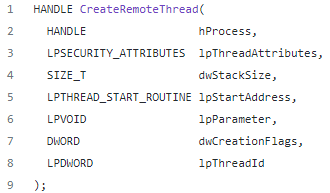
* hProcess — дескриптор запущеного процесу (отримано з OpenProcess);
* lpAddress — вказівник на бажану початкову адресу для виділеної пам’яті;
* dwSize — обсяг пам'яті, який ми хочемо виділити;
* flAllocationType — визначає тип пам’яті, яку ми будемо виділяти;
* flProtect — дозволи, які буде встановлено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

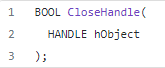
* hProcess — дескриптор процесу (отримано з OpenProcess);
* lpBaseAddress — вказівник на базову адресу виділеної пам’яті, куди ми будемо записувати наші дані;
* lpBuffer — покажчик на дані, які ми будемо записувати у виділену пам’ять;
* nSize — кількість байтів із нашого буфера, які ми запишемо у виділену 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.



**Рис. 2.3.7.** CloseHandle.

Функція CloseHandle (див. рис. 2.3.7) зробить саме те, що вона каже, і закриє дескриптор процесу. Для цього потрібен один аргумент:

* hObject — дескриптор, повернутий OpenProcess.
  1. **Бібліотеки для роботи з фото, аудіо матеріалом**

Як випливає з назви, обробка зображень означає обробку зображення, і це може включати безліч різних технік, поки ми не досягнемо своєї мети. Кінцевий результат може бути у вигляді зображення або відповідної ознаки цього зображення. Це може бути використано для подальшого аналізу та прийняття рішень.

Зображення може бути представлене як 2D-функція F (x, y), де x і y - просторові координати. Амплітуда F при певному значенні x, y відома як інтенсивність зображення в цій точці. Якщо x, y і значення амплітуди скінченні, тоді ми називаємо це цифровим зображенням. Це масив пікселів, розташованих у стовпцях і рядках. Пікселі - це елементи зображення, що містять інформацію про інтенсивність та колір. Зображення також може бути представлене у 3D, де x, y та z стають просторовими координатами. Пікселі розташовані у формі матриці. Це відомо як RGB-зображення.

Python - одна з широко використовуваних мов програмування для цієї мети. Його дивовижні бібліотеки та інструменти допомагають досягти завдання обробки зображень дуже ефективно. Важливо знати, що саме являє собою обробка зображень і яка її роль у загальній картині, перш ніж зануритися в її способи. Обробку зображень найчастіше називають «цифровою обробкою зображень», а домен, в якому вона часто використовується, - «Комп’ютерне бачення». Не плутайте - ми поговоримо про обидва ці терміни та про те, як вони пов’язані. Як алгоритми обробки зображень, так і алгоритми Computer Vision (CV) беруть зображення як вхідні дані; однак, при обробці зображень, вихід також є зображенням, тоді як при комп'ютерному зорі виходом можуть бути деякі особливості / інформація про зображення.

Дані, які ми збираємо або генеруємо, є переважно необробленими даними, тобто вони не підходять для безпосереднього використання в додатках через низку можливих причин. Тому нам потрібно спочатку проаналізувати його, виконати необхідну попередню обробку, а потім використовувати. Наприклад, припустимо, що ми намагалися створити класифікатор котів. Наша програма брала зображення як вхідний сигнал, а потім повідомляла нам, чи містить це зображення кота чи ні. Першим кроком для побудови цього класифікатора було б зібрати сотні фотографій котів. Одним із поширених питань є те, що всі зіскоблені нами картинки не мають однакового розміру розмірів, тому перед тим, як подавати їх на модель для навчання, нам потрібно буде змінити розмір / попередньо обробити їх до стандартного розміру. Це лише одна з багатьох причин, чому обробка зображень є важливою для будь-якої програми комп'ютерного зору.

Що таке обробка зображень та її використання в області машинного зору машинного навчання. Перш ніж використовувати зображення в наших програмах, ми говорили про деякі типові типи шуму та про те, як ми можемо видалити його із наших зображень за допомогою різних фільтрів. Обробка зображень відіграє важливу роль у таких висококласних програмах, як виявлення об'єктів або класифікація. Зверніть увагу, що ця стаття була лише верхівкою айсберга, і в цифровій обробці зображень є набагато більше, що неможливо висвітлити в одному підручнику. Прочитавши це, ви зможете заглибитися глибше та дізнатись про інші передові концепції, пов’язані з обробкою зображень.

Matplotlib є додатковим вибором для відображення кадрів із відео чи зображень. Ми покажемо кілька прикладів його використання тут. Numpy використовується для всього, що стосується «чисел і Python». В основному ми використовуємо функціональність масиву Numpy. Нарешті, ми використовуємо специфічні для python прив’язки для OpenCV під назвою python-OpenCV.

Існують деякі операції для OpenCV, які ви не зможете виконати без повної інсталяції OpenCV (розміром близько 3 ГБ), але насправді ви можете зробити чимало за допомогою досить мінімальної інсталяції python-OpenCV. Пізніше в цій серії ми закінчимо використовувати повну інсталяцію OpenCV, тому ви можете сміливо отримати її, якщо хочете, але ці 3 модулі займуть нас деякий час!

Переконайтеся, що ваші інсталяції пройшли успішно, запустивши Python і виконавши (див. рис. 2.4.1):



**Рис. 2.4.1.** Імпорти.

Якщо у вас немає помилок, ви готові до роботи. готовий Давайте пірнемо з глибини! По-перше, ми повинні зрозуміти кілька основних припущень і парадигм, коли йдеться про аналіз зображень і відео. Зважаючи на те, як сьогодні записує майже кожна відеокамера, записи насправді є кадрами, які відображаються один за одним 30-60+ разів на секунду. Однак у своїй основі це статичні кадри, як і зображення. Таким чином, розпізнавання зображень і аналіз відео використовують здебільшого ідентичні методи. Для деяких речей, як-от відстеження спрямованості, знадобиться послідовність зображень (кадрів), але щось на кшталт виявлення обличчя чи розпізнавання об’єктів можна зробити за допомогою майже однакового коду на зображеннях і відео. Далі багато аналізу зображень і відео зводиться до максимального спрощення джерела. Це майже завжди починається з перетворення в градації сірого, але це також може бути кольоровий фільтр, градієнт або їх комбінація. Звідси ми можемо робити всілякі аналізи та трансформації джерела. Як правило, відбувається трансформація, потім аналіз, а потім будь-які накладення, які ми хочемо застосувати, застосовуються назад до вихідного джерела, тому ви часто можете побачити «готовий продукт», можливо, розпізнавання об’єктів або обличчя відображаються на повнокольоровому зображенні чи відео. Однак рідко дані фактично обробляються в необробленому вигляді. Деякі приклади того, що ми можемо зробити на базовому рівні. Усе це робиться за допомогою звичайної веб-камери, нічого особливого (див. рис. 2.4.2-3):



**Рис. 2.4.2.** Віднімання фону.



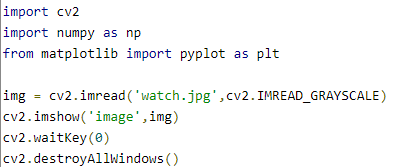
**Рис. 2.4.3.** Фільтрація кольору.

У разі виявлення країв чорний колір відповідає значенням пікселів (0,0,0), а білі лінії – (255,255,255). Кожне зображення та кадр із відео розбивається на пікселі таким чином, і ми можемо зробити висновок, як у випадку виявлення країв, на основі того, де білі пікселі порівнюються з чорними. Потім, якщо ми хочемо побачити вихідне зображення з позначеними краями, ми зазначаємо всі координати розташування білих пікселів, а потім позначаємо ці місця на оригінальному вихідному зображенні чи відео.

До кінця цього підручника ви зможете робити все вищезазначене та зможете навчити свою машину розпізнавати будь-який об’єкт, який ви хочете. Як я вже казав спочатку, першим кроком зазвичай є перетворення на градації сірого. Перед цим нам потрібно завантажити зображення. Отже, давайте зробимо це! У всьому цьому підручнику я наполегливо рекомендую вам використовувати власні дані для гри. Якщо у вас є веб-камера, обов’язково використовуйте її, інакше знайдіть зображення, з яким, на вашу думку, буде цікаво працювати. Якщо у вас виникли проблеми, ось зображення годинника (див. рис. 2.4.4-5):



**Рис. 2.4.4.** Фото екземпляр.



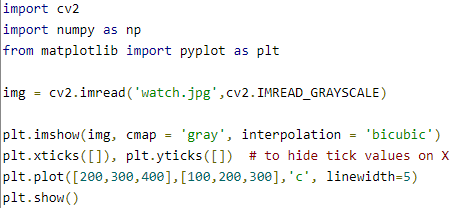
**Рис. 2.4.5.** Частина коду.

По-перше, ми імпортуємо кілька речей, ті три модулі, які я вам наказав встановити. Далі ми визначаємо img як cv2.read(файл зображення, parms). За замовчуванням буде IMREAD\_COLOR, тобто колір без альфа-каналу. Якщо ви не знайомі, альфа – це ступінь непрозорості (протилежність прозорості). Якщо вам потрібно зберегти альфа-канал, ви також можете використовувати IMREAD\_UNCHANGED. Багато разів ви будете читати в кольоровій версії, а потім перетворювати її на сіру. Якщо у вас немає веб-камери, це буде основний метод, який ви використовуватимете протягом цього підручника, завантажуючи зображення.

Замість використання IMREAD\_COLOR... тощо, ви також можете використовувати прості числа. Ви повинні бути знайомі з обома варіантами, щоб зрозуміти, що людина робить. Для другого параметра можна використовувати -1, 0 або 1. Колір дорівнює 1, відтінки сірого дорівнює 0, а незмінний дорівнює -1. Таким чином, для відтінків сірого можна зробити img = cv2.imread('watch.jpg', 0)

Після завантаження ми використовуємо cv2.imshow(title,image), щоб показати зображення. Звідси ми використовуємо cv2.waitKey(0), щоб дочекатися натискання будь-якої клавіші. Коли це буде зроблено, ми використовуємо cv2.destroyAllWindows(), щоб закрити все.

Як згадувалося раніше, ви також можете відображати зображення за допомогою Matplotlib, ось код, як це можна зробити:



**Рис. 2.4.6.** Фрагмент коду для обробки.

Зауважте, що тут можна побудувати лінії так само, як і будь-який інший графік Matplotlib, використовуючи розташування пікселів як координати. Проте, якщо ви хочете малювати на своїх зображеннях, Matplotlib не потрібен. OpenCV пропонує чудові методи для цього. Коли ви закінчите вносити зміни, ви можете зберегти, наприклад (див. рис. 2.4.7):



**Рис. 2.4.7.** Запис у файл.

* 1. **Бібліотеки для роботи з технологіями розпізнавання**

Домінуючий підхід до розпізнавання мови базується на генеративній моделі аналізу та розпізнавання образів. Ідея синтезного аналізу для багатьох завдань розпізнавання мовного сигналу виявилася найбільш продуктивною порівняно з дискримінантною моделлю. Теорія генеративної моделі успішно застосовується для обробки та розпізнавання різних об'єктів, полів, сигналів та зображень, зокрема в радіофізиці та біоінформатиці. У процесі розпізнавання вхідний сигнал перетворюється в послідовність векторів ознак, що надходять з інтервалом, наприклад, 10 мс. Таким чином, формуються вибірки сигналів, з якими розпізнавач або декодер продовжує працювати, реалізує алгоритм за певною схемою розпізнавання. Будь-який розпізнавач використовує базу даних та знання.

Сучасні системи розпізнавання мови пройшли довгий шлях від своїх давніх аналогів. Вони можуть розпізнавати мовлення кількох мовців і мають величезний словниковий запас на багатьох мовах. Перший компонент розпізнавання мови - це, звичайно, мова. Мова повинна бути перетворена з фізичного звуку в електричний сигнал за допомогою мікрофона, а потім у цифрові дані за допомогою аналого- цифрового перетворювача. Оцифрувавши кілька моделей, можна використовувати для транскрипції звуку в текст. Більшість сучасних систем розпізнавання мовлення покладаються на те, що відоме як модель прихованого Маркова (HMM). Цей підхід працює на припущенні, що мовленнєвий сигнал при розгляді на досить короткому часовому масштабі (скажімо, десять мілісекунд) може бути розумно апроксимований як стаціонарний процес - тобто процес, в якому статистичні властивості не змінюються з часом. У типовому HMM мовний сигнал розділений на 10-мілісекундні фрагменти. Спектр потужності кожного фрагмента, який, по суті, є графіком потужності сигналу як функції від частоти, відображається на вектор дійсних чисел, відомий як цепстральні коефіцієнти. Розмір цього вектора, як правило, невеликий - іноді до 10, хоча більш точні системи можуть мати розмір 32 і більше. Кінцевий вихід HMM є послідовністю цих векторів.

Деякі з цих пакетів, такі як д та apiai, пропонують вбудовані функції, такі як обробка природної мови для виявлення наміру мовця, які виходять за рамки базового розпізнавання мови. Інші, такі як google-cloud-промова, зосереджуються виключно на перетворенні мови в текст. Існує один пакет, який вирізняється простотою використання: SpeechRecognition [2]. Розпізнавання мови вимагає введення звуку, а SpeechRecognition робить отримання цього введення справді простим.

Замість того, щоб створювати сценарії для доступу до мікрофонів та обробки аудіофайлів з нуля, SpeechRecognition забезпечить вам роботу лише за кілька хвилин. Бібліотека SpeechRecognition [3] виконує роль обгортки для декількох популярних мовних API і, отже, надзвичайно гнучка. Один із них - Google Web Speech API - підтримує ключ API за замовчуванням, який жорстко закодований у бібліотеці SpeechRecognition. Це означає, що ви можете зійти з ніг, не маючи необхідності підписуватись на послугу. Гнучкість та простота використання пакета SpeechRecognition роблять його чудовим вибором для будь-якого проекту Python. Однак підтримка кожної функції кожного обгортаного API не гарантується. Вам доведеться витратити деякий час на вивчення доступних варіантів, щоб з’ясувати, чи буде функція SpeechRecognition працювати у вашому конкретному випадку.

Нейронні мережі [2] - це багатошарові мережі (рис.1.4), що складаються з

нейронів або вузлів. Ці нейрони є основними процесорними одиницями нейронної мережі. Вони покликані діяти як мозок людини. Вони беруть дані, тренуються розпізнавати закономірності в даних, а потім прогнозують результат.

Вхідні шари отримують вхідні дані, вихідний рівень передбачає вихідні дані, а приховані шари виконують більшість розрахунків. Кількість прихованих шарів можна змінювати відповідно до вимог. У нейронній мережі повинен бути принаймні один прихований шар. Розглянемо зображення, кожен піксель подається як вхідний сигнал до кожного нейрона першого шару, нейрони одного шару пов’язані з нейронами наступного шару через канали. Кожному з цих каналів присвоюється числове значення, відоме як вага. Вхідні дані помножуються на відповідні ваги, і ця зважена сума потім подається як вхідні дані до прихованих шарів.Вихідні дані з прихованих шарів передаються через функцію активації, яка визначатиме, чи буде активований певний нейрон чи ні. Активовані нейрони передають дані в наступні приховані шари. Таким чином, дані поширюються через мережу, це відоме як пряме розповсюдження. У вихідному рівні нейрон з найбільшим значенням передбачає вихід. Ці результати є значеннями ймовірності. Прогнозований результат порівнюється з фактичним результатом для отримання помилки. Потім ця інформація передається назад через мережу, процес відомий як зворотне розповсюдження. На основі цієї інформації коригуються ваги. Цей цикл прямого та зворотного розповсюдження виконується кілька разів на декількох входах, поки мережа не прогнозує результат правильно в більшості випадків. На цьому навчальний процес нейронної мережі закінчується. Час, необхідний для тренування нейронної мережі, в деяких випадках може збільшитися.

* 1. **Підготовка набору даних та навчання нейронної мережі**

Коли більшість людей думають про штучний інтелект (ШІ), вони думають про два можливі майбутні. Позитивне майбутнє, де безпілотні автомобілі допомагають нам орієнтуватися на наших дорогах, а роботи-слуги допомагають нам обслуговувати наші будинки. Або більш негативний, де машини забирають наші робочі місця та роботу.

На щастя, схоже, що негативне майбутнє — це не те, про що ми повинні турбуватися. Системи штучного інтелекту не замінять людей у робочій силі, а скоріше вони існуватимуть поруч із людьми як безцінні помічники. Незважаючи на те, що безпілотні автомобілі вже на шляху до реальності, деякі інші наші грандіозні цілі щодо штучного інтелекту все ще чекають реалізації, однак, перш ніж ми досягнемо цього, потрібно зробити ще більше.

Потрапити на сцену, на якій ми вже знаходимося, не сталося миттєво. Системи штучного інтелекту потрібно було навчити, щоб вони могли надавати нам ті переваги, до яких ми вже звикли.

Процес навчання ШІ складається з трьох етапів. Перший крок, навчання, передбачає введення даних у комп’ютерний алгоритм для створення прогнозів і оцінки їх точності. Другий крок, перевірка, оцінює, наскільки добре навчена модель працює на раніше невидимих даних. Нарешті, проводиться тестування, щоб з’ясувати, чи робить кінцева модель точні прогнози з новими даними, яких вона ніколи раніше не бачила.

У цій публікації ми докладніше розглянемо кожен із цих кроків і пояснимо, як вони взаємодіють один з одним.

Першим кроком під час навчання ШІ є передача даних у комп’ютерну систему. Це змушує його робити прогнози та оцінювати свою точність щодо кожного нового циклу або проходити через усі доступні точки даних. Завдяки використанню методів машинного навчання (ML), включаючи глибоке навчання, алгоритм може аналізувати дані та робити кращі прогнози.

Таким чином ми навчаємо програмне забезпечення, як ідентифікувати різні особливості, які можуть бути присутніми на зображенні, наприклад тон шкіри або колір волосся. Згодом ці початкові припущення стають дедалі точнішими, поки не досягнуть точки, коли більше немає місця для вдосконалення.

У контрольованому навчанні алгоритм «навчається» з навчального набору даних шляхом ітерації через передбачення невідомих змінних. У моделях керованого машинного навчання потрібна робота людини, щоб «навчати» комп’ютерну систему шляхом надання відповідних міток для вхідних даних. Озираючись назад до нашого попереднього прикладу, використовуючи модель навчання під наглядом, обличчя, які вводяться, будуть належним чином позначені, а інші елементи також будуть введені з правильними мітками. Таким чином відображення у вікні не сплутаєш з людиною. Іншим прикладом моделі навчання під наглядом є прогноз подорожі на основі щоденних поїздок. Навчаючи модель розуміти вплив погоди та часу доби, вона може робити точніші прогнози на основі поточних умов.

Моделі неконтрольованого навчання працюють незалежно, щоб знайти структури, які можуть існувати в немаркованих даних. Це розпізнавання шаблонів може бути корисним для пошуку кореляцій у даних, які можуть бути неочевидними відразу, допомагаючи визначити викиди, які варто додатково дослідити. Моделі неконтрольованого навчання набагато швидше навчаються, але все одно потребують втручання людини для перевірки вихідних змінних.

Три типи неконтрольованого навчання: кластеризація, аналіз правил асоціації та виявлення викидів.

* Кластеризація допомагає групувати непомічені дані разом на основі певних критеріїв. Дані, про які йдеться, можна згрупувати на основі схожості чи відмінності, а конкретні точки даних об’єднати в групи. Цей тип неконтрольованого навчання корисний для сегментації ринку;
* Інтелектуальний аналіз правил асоціації розглядає дані дещо інакше, намагаючись знайти зв’язки між точками даних. Цей тип неконтрольованого навчання корисний для аналізу зв’язків між різними групами предметів і визначення комбінацій, які з більшою ймовірністю зустрічаються разом;
* Виявлення викидів можна використовувати для пошуку точок даних, які виходять за певні межі. Цей тип неконтрольованого навчання також корисний для пошуку аномалій у наборах даних, що потенційно може призвести до виявлення незвичної чи шахрайської поведінки.

Нова підгрупа неконтрольованого навчання відома як навчання з підкріпленням. Навчання з підкріпленням – це тип машинного навчання, який використовує винагороди та покарання, намагаючись максимізувати показник винагороди. Найчастіше використовується для ігор і безпілотних автомобілів.

Після завантаження даних у модель можна розпочинати наступний етап навчання.

Другим кроком у навчанні штучному інтелекту є тестування перевірки, яке оцінює, як моделі працюють із даними, які модель раніше не бачила. Перевірочний тест використовується для оцінки того, наскільки добре навчена модель працює з невидимими даними, що може допомогти визначити, чи потрібно продовжити навчання або якимось чином змінити.

Моделі навчання з підкріпленням оцінюються, намагаючись максимізувати їх майбутню метрику винагороди, тому вони продовжуються, доки не зникне потенціал для вдосконалення. Навпаки, контрольоване та неконтрольоване навчання мають кінцеві точки, де розмір набору даних визначає, які ваги мають бути призначені та підтверджені відповідно.

Загальна стратегія відома як «завчасна зупинка», за якої оцінка продуктивності змушує тренерів усвідомлювати, що навряд чи будь-які подальші зміни значно покращать прогнози за наявних ресурсів (наприклад, часу). Якщо це трапляється, часто доцільно припинити навчання та дослідити інші варіанти.

Чим точніші рішення може приймати ваш штучний інтелект, тим краще ви будете підготовлені, коли він запрацює, однак вам потрібно дивитися глибше, якщо ви також отримуєте 100% точність.

Однією з класичних проблем із навчанням ШІ є переобладнання, коли ваша програма добре працює з навчальними даними, але не так добре з новими даними. На протилежній стороні шкали, недостатнє пристосування означає, що у вас є моделі, які погано справляються зі старими та новими даними. Якщо на цьому етапі він не працює так, як передбачалося, поверніться до процесу навчання та повторюйте, доки не будете задоволені точністю.

Colaboratory від Google (скорочено Google Colab) — це середовище виконання на базі блокнота Jupyter, яке дозволяє запускати код повністю в хмарі.

Це необхідно, оскільки це означає, що ви можете тренувати великомасштабні моделі ML і DL, навіть якщо у вас немає доступу до потужної машини або високошвидкісного доступу до Інтернету.

Google Colab підтримує екземпляри як GPU, так і TPU, що робить його ідеальним інструментом для глибокого навчання та ентузіастів аналізу даних через обмеження обчислень на локальних машинах.

Оскільки до блокнота Colab можна отримати віддалений доступ із будь-якої машини через браузер, він також добре підходить для комерційних цілей.

# РОЗДІЛ III. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ. РЕАЛІЗАЦІЯ ПРОЄКТУ

* 1. **Створення прототипів та тестування**

У курсовій 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.

* 1. Реалізація віконної програми з майбутнім функціоналом
  2. Написання модулів для роботи комп’ютерного зору та розпізнавання мовлення
  3. Підготовка моделі, тренування нейронної мережі

* 1. Реалізація програмної взаємодії з грою
  2. Інтеграція роботи модулів у програмі та тестування

# РОЗДІЛ IV. ТЕСТУВАННЯ РОЗРОБКИ. ДОСЛІДЖЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

* 1. Запуск програми в тестовому середовищі
  2. Аналіз результатів тестування та корекція програми
  3. Збір даних із довготривалого користування в “бойовому” режимі
  4. Аналіз результатів дослідження

# 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. Robert M.HaralickLinda G.Shapiro; Image segmentation techniques [Електронний ресурс] — Режим доступу: https://www.sciencedirect.com/science/ article/ pii/S0734189X85901537.
18. Dilpreet Kaur, Yadwinder Kaur; Various Image Segmentation.Techniques: A Review// IJCSMC, Vol. 3, Issue. 5, May 2014, pg.809 – 814. ISSN 2320–088X.
19. Su Hnin Hlaing, Aung Soe Khaing; Weed and crop segmentation and classification using area thresholding //International Journal of Research in Engineering and Technology eISSN: 2319-1163 pISSN: 2321-7308.
20. Sebastian Haug, Andreas Michaels; Plant classification system for crop /weed discrimination without segmentation// Published in IEEE Winter Conference 2014. DOI:10.1109/WACV.2014.6835733.
21. 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.
22. 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.
23. The 2016 Sugar Beets Dataset Recorded at Campus Klein Altendorf in Bonn,Germany [Електронний ресурс] — Режим доступу: http://www.ipb.unibonn.de/data /sugabeets2016.
24. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551, Winter 1989.
25. Le Hou ; Dimitris Samaras ; Tahsin M. Kurc ; Patch-Based Convolutional Neural Network for Whole Slide Tissue Image Classification //2016 IEEE Conference on Computer Vision and Pattern Recognition, ISSN: 1063-6919.
26. Fully Convolutional Networks for Semantic Segmentation [Електронний ресурс] — Режим доступу: <https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf>.
27. 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.
28. 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

SLAVE\_MESSAGE = {

'commands': [

'[ACTION] \tpress',

'[ACTION] \tloot',

'[MOVE] \tv',

'[MOVE] \t^',

'[MOVE] \t<-',

'[MOVE] \t->',

'[TARGET] \tvampire #1',

'[TARGET] \tvampire #2',

'[TARGET] \tvampire #3'

],

'weights': [2, 2, 4, 3, 4, 3, 1, 1, 1]

}

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\_\_':

**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 random

import time

import speech\_recognition as sr

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

"""Transcribe speech from recorded from `microphone`.

Returns a dictionary with three keys:

"success": a boolean indicating whether or not the API request was

successful

"error": `None` if no error occured, otherwise a string containing

an error message if the API could not be reached or

speech was unrecognizable

"transcription": `None` if speech could not be transcribed,

otherwise a string containing the transcribed text

"""

# check that recognizer and microphone arguments are appropriate type

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")

# adjust the recognizer sensitivity to ambient noise and record audio

# from the microphone

with microphone as source:

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

audio = recognizer.listen(source)

# set up the response object

response = {

"success": True,

"error": None,

"transcription": None

}

# try recognizing the speech in the recording

# if a RequestError or UnknownValueError exception is caught,

# update the response object accordingly

try:

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

except sr.RequestError:

# API was unreachable or unresponsive

response["success"] = False

response["error"] = "API unavailable"

except sr.UnknownValueError:

# speech was unintelligible

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

return response

def recognition\_speech():

# set the list of words, maxnumber of guesses, and prompt limit

WORDS = ["apple", "banana", "grape", "orange", "mango", "lemon"]

NUM\_GUESSES = 3

PROMPT\_LIMIT = 5

# create recognizer and mic instances

recognizer = sr.Recognizer()

microphone = sr.Microphone()

# get a random word from the list

word = random.choice(WORDS)

# format the instructions string

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)

# show instructions and wait 3 seconds before starting the game

print(instructions)

time.sleep(3)

for i in range(NUM\_GUESSES):

# get the guess from the user

# if a transcription is returned, break out of the loop and

# continue

# if no transcription returned and API request failed, break

# loop and continue

# if API request succeeded but no transcription was returned,

# re-prompt the user to say their guess again. Do this up

# to PROMPT\_LIMIT times

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 there was an error, stop the game

if guess["error"]:

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

break

# show the user the transcription

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

# determine if guess is correct and if any attempts remain

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

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

# determine if the user has won the game

# if not, repeat the loop if user has more attempts

# if no attempts left, the user loses the game

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:

# properties

w = 0

h = 0

hwnd = None

cropped\_x = 0

cropped\_y = 0

offset\_x = 0

offset\_y = 0

# constructor

def \_\_init\_\_(self, window\_name):

# find the handle for the window we want to capture

self.hwnd = win32gui.FindWindow(None, window\_name)

if not self.hwnd:

raise Exception('Window not found: {}'.format(window\_name))

# get the window size

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

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

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

# account for the window border and titlebar and cut them off

border\_pixels = 8

titlebar\_pixels = 30

self.w = self.w - (border\_pixels \* 2)

self.h = self.h - titlebar\_pixels - border\_pixels

self.cropped\_x = border\_pixels

self.cropped\_y = titlebar\_pixels

# set the cropped coordinates offset so we can translate screenshot

# images into actual screen positions

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

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

def get\_screenshot(self):

# get the window image data

wDC = win32gui.GetWindowDC(self.hwnd)

dcObj = win32ui.CreateDCFromHandle(wDC)

cDC = dcObj.CreateCompatibleDC()

dataBitMap = win32ui.CreateBitmap()

dataBitMap.CreateCompatibleBitmap(dcObj, self.w, self.h)

cDC.SelectObject(dataBitMap)

cDC.BitBlt((0, 0), (self.w, self.h), dcObj,

(self.cropped\_x, self.cropped\_y), win32con.SRCCOPY)

# convert the raw data into a format opencv can read

#dataBitMap.SaveBitmapFile(cDC, 'debug.bmp')

signedIntsArray = dataBitMap.GetBitmapBits(True)

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

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

# free resources

dcObj.DeleteDC()

cDC.DeleteDC()

win32gui.ReleaseDC(self.hwnd, wDC)

win32gui.DeleteObject(dataBitMap.GetHandle())

# drop the alpha channel, or cv.matchTemplate() will throw an error like:

# error: (-215:Assertion failed) (depth == CV\_8U || depth == CV\_32F) && type == \_templ.type()

# && \_img.dims() <= 2 in function 'cv::matchTemplate'

img = img[..., :3]

# make image C\_CONTIGUOUS to avoid errors that look like:

# File ... in draw\_rectangles

# TypeError: an integer is required (got type tuple)

# see the discussion here:

# https://github.com/opencv/opencv/issues/14866#issuecomment-580207109

img = np.ascontiguousarray(img)

return img

# find the name of the window you're interested in.

# once you have it, update window\_capture()

# https://stackoverflow.com/questions/55547940/how-to-get-a-list-of-the-name-of-every-open-window

def list\_window\_names(self):

def winEnumHandler(hwnd, ctx):

if win32gui.IsWindowVisible(hwnd):

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

win32gui.EnumWindows(winEnumHandler, None)

# translate a pixel position on a screenshot image to a pixel position on the screen.

# pos = (x, y)

# WARNING: if you move the window being captured after execution is started, this will

# return incorrect coordinates, because the window position is only calculated in

# the \_\_init\_\_ constructor.

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')):

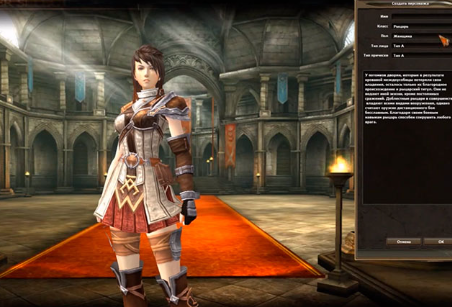
""" Returns a human readable string reprentation of bytes"""

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

**ДОДАТОК Б**

**Опис гри (в зображеннях).**















s

