Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

РОЗРОБКА ДОДАТКУ ДЛЯ АВТОМАТИЧНОГО РОЗМІЩЕННЯ ВІКОН РОБОЧОГО СТОЛУ

Текстова частина до кваліфікаційної роботи

за спеціальністю „Комп'ютерні науки" 122

Керівник курсової роботи
с.в. Вовк Н.Є.
(прізвище та ініціали)
_____

(підпис)

"____" _____ 2024 р.

Виконав студент _____

Вакуленко В.С.
(прізвище та ініціали)

"____" _____ 2024 р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри мультимедійних систем,

доцент, к.ф-м.н.

_____О. П. Жежерун (підпис)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Вакуленко Владиславу Сергійовичу факультету інформатики 4-го курсу

ТЕМА Розробка додатку для автоматичного розміщення вікон робочого столу

Календарний план

№ п/п	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	12.10.2023	
2.	Аналіз матеріалів за темою	01.11.2023	
3.	Розробка та програмування алгоритму	01.12.2023	
4.	Написання текстової частини до кваліфікаційної роботи	01.03.2024	
5.	Захист кваліфікаційної роботи	29.05.2024	

Вакуленко В. С. _____

Вовк Н.Є. _____

“_____”_____

Зміст

Анотація

Робота спрямована на розробку додактку для автоматичного розміщення вікон робочого столу. Застосунок буде корисний людям, які використовують декілька вікон одночасно і хочуть економити час на їх розміщенні.

Графічний інтерфейс та функціонал програми були реалізовані мовою Python[1]. Для впровадження голосового керування були використані можливості пакету speech_recognition [14]. Обробка голосових команд та генерування шаблонів здійснюється з використання OpenAI API [15].

Ключові слова: Python застосунок, голосове керування, штучний інтелект.

Вступ

Актуальність теми

На сьогодні існують програмні продукти для більшості потреб в усіх галузях. Здебільшого кожен з них так чи інакше вирішує одну задачу. Саме тому користувачам доводиться комбінувати різні ПП для виконання більш складних задач.

Найпоширенішим інструментом для використання цих ПП звісно ж є комп'ютер. Сучасні операційні системи мають вбудований функціонал автоматичного розміщення вікон. Однак здебільшого кількість вікон в таких шаблонах обмежена. До того ж, їх не можна змінити, що обмежує користувача.

Більшість людей виконують схожі задачі в одних і тих самих застосунках кожного дня. А отже вони змушені витрачати час на позиціонування одних і тих самих вікон щодня.

Тому розробка додатку для створення шаблонів і автоматичного розміщення вікон є актуальною задачею.

Мета кваліфікаційної роботи

Розробити додаток для створення шаблону і автоматичного розміщення вікон. Реалізувати голосове керування для пришвидшення взаємодії з застосунком. Використати можливості штучного унтелекту для покращення результатів роботи додатку.

Наукова новизна

Наукова новизна полягає у:

a) Розширенні базового функціоналу ОС для розміщення вікон робочого столу;
b) Впроваджені функціоналу генерування шаблонів з використанням штучного інтелекту;
c) Реалізації розпізнавання мови для голосового керування застосунком та штучного інтелекту для покращення результатів роботи застосунку.

Практичне значення

Додаток буде корисний людям, які часто використовують однакові комбінації з декількох вікон і хочу економити час на їх розміщенні.

Постановка задачі

1. Огляд наявних програмних рішень та аналіз їх функціоналу;
2. Дослідження наявного функціоналу автоматичного розміщення вікон та можливостей пакетів мови Python для керування вікнами;
3. Розробка формату збереження даних та алгоритмів збереження / застосування шаблонів;
4. Розробка графічного інтерфейсу;
5. Впровадження голосового керування;
6. Покращення результатів роботи застосунку за допомогою штучного інтелекту.

Додаток має надавати можливість створити шаблон шляхом генерації з використанням штучного інтелекту чи сканування розміщених вікон робочого столу. Користувач повинен мати можливість застосувати цей шаблон для будь-яких відкритих вікон використовуючи графічний інтерфейс.

Структура роботи

Кваліфікаційна робота містить анотацію, вступ, три розділи, висновки та список використаних джерел.

У першому розділі проаналізовано наявні програмні рішення та їх функціонал. Також розглянуто можливості пакетів мови Python для керування вікнами. Описано основні можливості, які будуть використані під час реалізації програмного продукту.

Другий розділ присвячений розробці основного функціоналу. Детально описана робота графічного інтерфейсу, а також формату збереження даних та алгоритмів для роботи з шаблонами.

Третій розділ описує розширення функціоналу за рахунок використання можливостей голосового керування та штучного інтелекту.

Основна частина

1 Керування вікнами у Windows

1.1 Огляд наявних програмних рішень та їх функціоналу

Divvy [10]

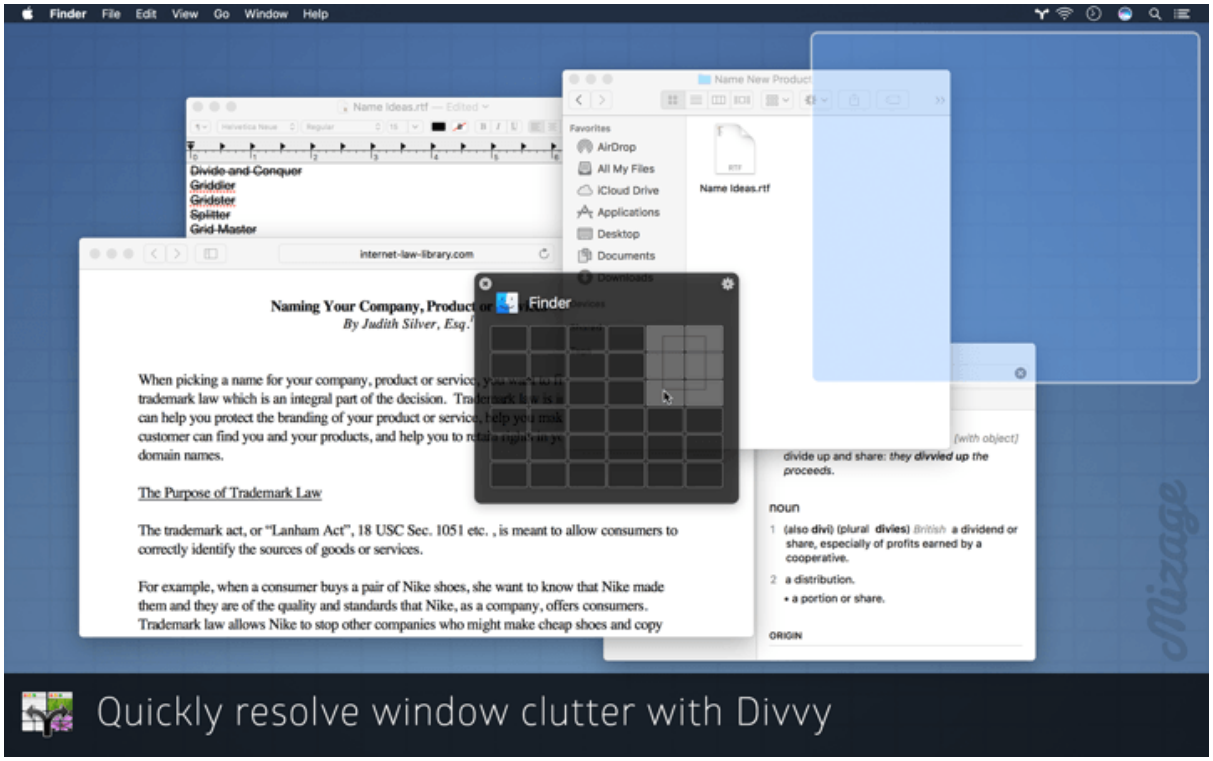


Рисунок 1.1 – Divvy [10] – додаток для розміщення вікон за сіткою

Divvy [10] – застосунок для розміщення вікон за сіткою. Він не надає функціоналу для створення і застосування шаблонів, проте доволі вдало реалізує задачу зручного і швидкого розміщення вікон робочого столу.

WindowGrid [13]

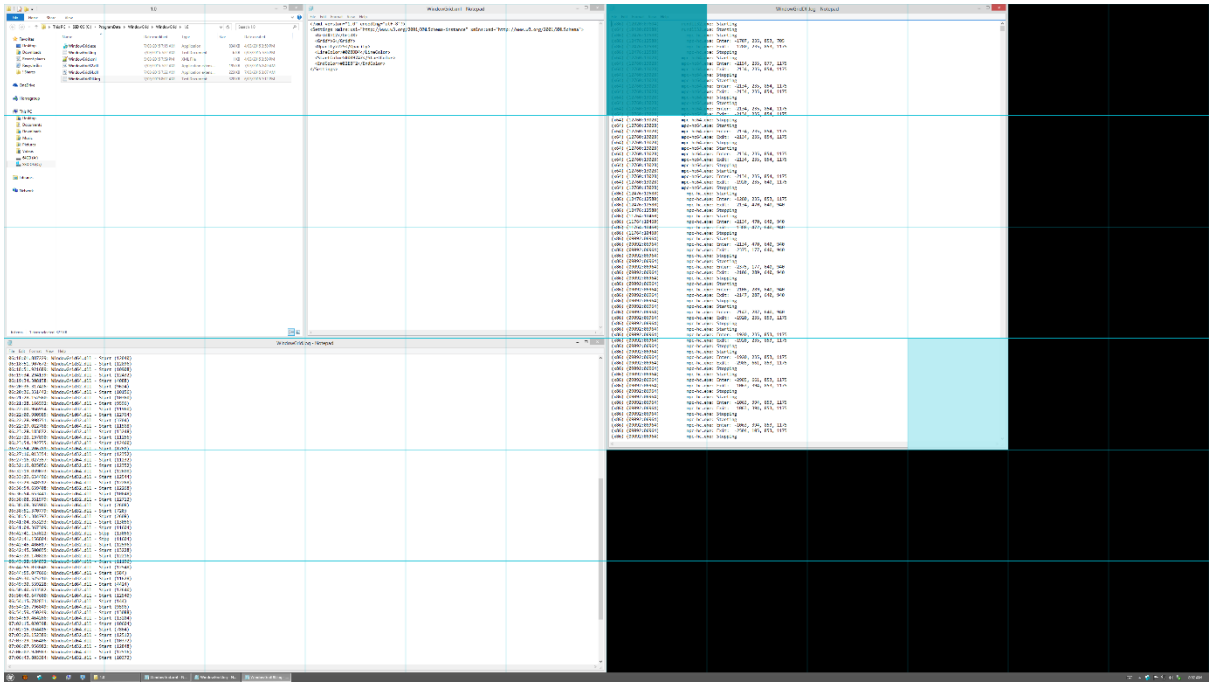


Рисунок 1.2 – WindowGrid [13] – додаток для розміщення вікон за сіткою

WindowGrid [13] – застосунок, який пропонує схожий до Divvy функціонал. Перевагою цього застосунку є можливість бачити майбутній вигляд вікна під час розміщення.

MaxTo [12]

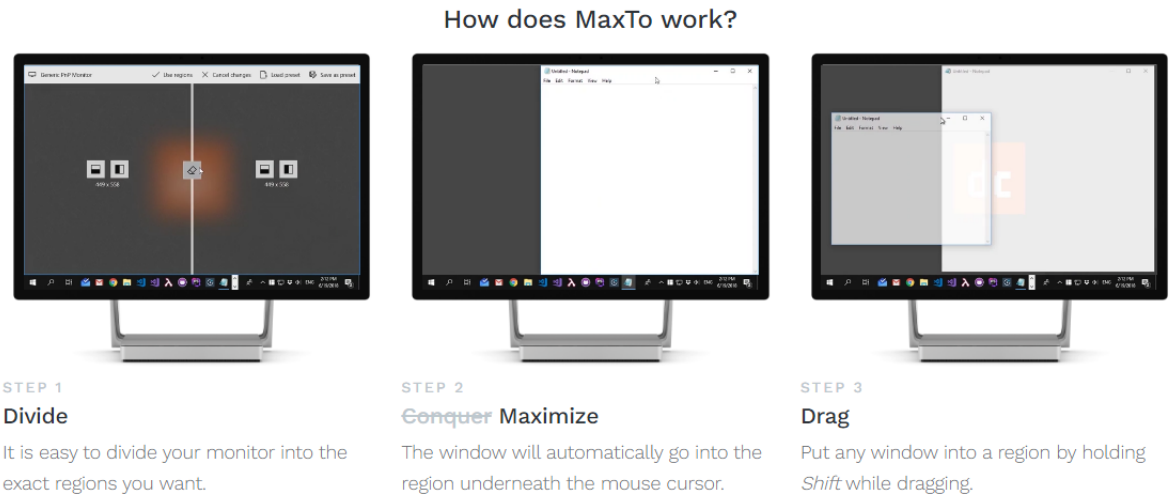


Рисунок 1.3 – MaxTo [12] – додаток для розміщення вікон за шаблоном

MaxTo [12] – застосунок для створення і застосування шаблонів. Дане програмне рішення частково реалізовує запланований функціонал цієї роботи.

Переваги:

a) Можливість заміни вікон в шаблоні без повторної взаємодії з застосунком;
b) Використання сітки для створення шаблону, що дозволяє зробити шаблон рівним, а також є інтуїтивно зрозумілим.

Недоліки:

a) Шаблон можна створити тільки для одного монітора, що не завжди відповідає потребам користувача;
b) Немає можливості візуалізувати розміщені вікна перед чи під час створення шаблона, через що створений шаблон може не відповідати очікуванням.

Mosaico [11]

Рисунок 1.4 – Mosaico [11] – *додаток для створення і застосування шаблонів*

Mosaico [11] – застосунок для створення і застосування шаблонів. Дане програмне забезпечення частково реалізовує заплановий функціонал цієї роботи.

Переваги:

a) Шаблон зберігається разом з програмами, що економить час користувача при його застосуванні;
b) Застосунок самостійно запускає всі потрібні для застосованого шаблону програми, що означає, що користувачу не потрібно відкривати окремі програми перед початком роботи.

Недоліки:

a) Шаблон неможливо зберегти без програм, що роботь його не універсальним;
b) Застосунок пропонує стандартні шаблони для розміщення вікон перед створенням шаблону, що обмежує можливість вибору користувача.

Проаналізувавши аналоги, можна виділити такі ідеї:

a) Розміщення вікон для шаблона з використанням сітки є вдалим рішення, яке дозволяє економити час і підвищувати якість. Водночас даний функціонал може певним чином обмежувати користувача і не є пріоритетним в рамках виконання цієї роботи. Саме тому, перевага буде надана скануванню вже розміщенних вікон на робочому столі.
b) Збереження програм в шаблоні також є вдалою ідеєю, хоч і потребує вдосконалення. В рамках виконання даної роботи ця задача не є пріоритетною, проте в якості можливих покращень буде надана ідея реалізації так званих станів шаблона. Це дозволить зберігати набори програм для кожного шаблона, що в свою чергу дозволить зберігати універсальність шаблона і при цьому додатково економити час користувача.

1.2 Дослідження функціоналу наявних Python бібліотек для керування вікнами

Дане дослідження проводитиметься з огляду на наступні потреби:

a) Сканування робочого столу та повернення списку процесів та / або вікон
b) Фільтрація користувацьких та системних процесів та / або вікон
c) Маніпулювання вікнами
    a. Відкриття / закриття
    b. Зміна розмірів і позиції

Розглянемо PyWin32 [2]. Це розширення, яке надає доступ до багатьох API операційної системи Windows [16], серед яких є і пакети, які задовольняють потреби даного дослідження. Згідно статистики, представленій на PyPI Stats [3] станом на лютий 2024 року, це розширення має понад 7 мільйонів скачувань на місяць. А згідно даних GitHub [17] репозиторія [4], останній реліз відбувся 26 березня 2023 року. Отже дане розширення може бути перспективним варіантом в рамках даного дослідження.

Почнемо з win32gui [5]. Це достатньо потужний інструмент для роботи з вікнами в операційній системі Windows [16]. Серед іншого, корисними при розробці додатка можуть виявитися:

a) EnumWindows – метод, який дозволяє отримати всі вікна;
b) GetWindowRect – метод, який дозволяє отримати координати вікна;
c) GetWindowPlacement - метод, який дозволяє визначити чи вікно згорнуте / розгорнуте і тд;
d) IsWindowVisible – метод, який дозволяє визначити чи видиме вікно;
e) ShowWindow – метод, який дозволяє відкрити вікно;
f) SetWindowPos – метод, який дозволяє спозиціонувати вікно.

Також розглянемо win32con [2], як інструмент для визначення певних властивостей вікна, як-от:

a) Розгорнуте / згорнуте;

b) Z позиція;
c) Блокування.

Після детальнішого дослідження розглянутого функціоналу, був зроблений висновок, що потреба в фільтруванні користувацьких та / або системних вікон не була покрита. Жоден серед розглянутих в подальшому розширень не покривав дану потребу повністю, тому було прийнято рішення використати симбіоз двох пакетів.

Розглянемо pywinauto [18]. Це розширення також спеціалізується на операційній системі Windows [16] і дозволяє взамодіяти з вікнами. Згідно статистики, представленій на PyPI Stats [6] станом на лютий 2024 року, це розширення має близько 1 мільйона скачувань на місяць. Отже дане розширення може бути перспективним варіантом в рамках даного дослідження.

Розглянемо клас Desktop, а саме його метод windows. Цей метод, як і win32gui.EnumWindows [5], повертає список вікон, але інший. Дослідимо ці два методи, щоб дізнатися чи отримаємо ми щось корисне. Дослідження проходитиме за умови єдинооо відкритого користувачем вікна “ComputerScience_4_Vakulenko.docx – Word”.

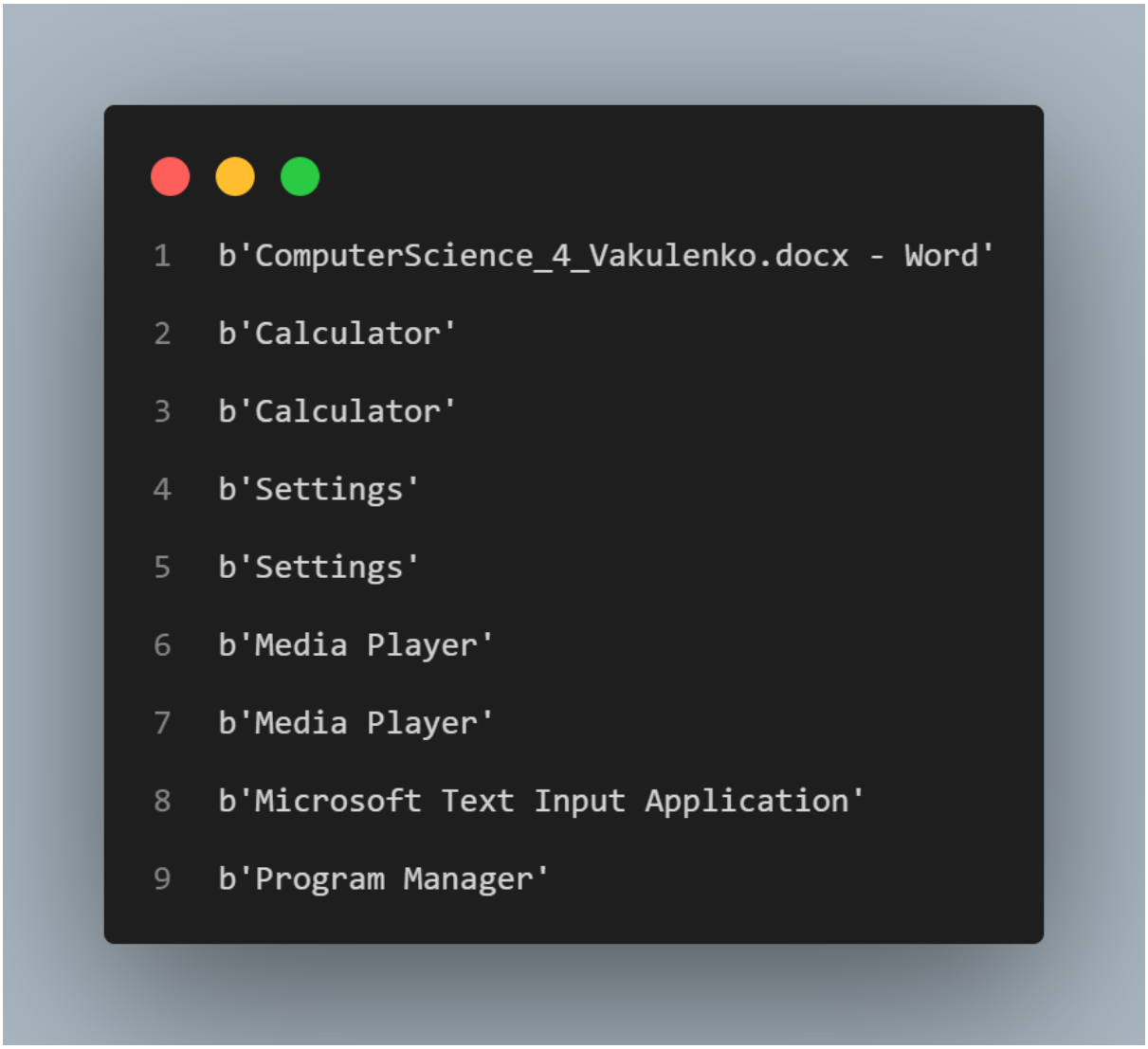


Рисунок 1.5 – Результат використання win32gui [5]

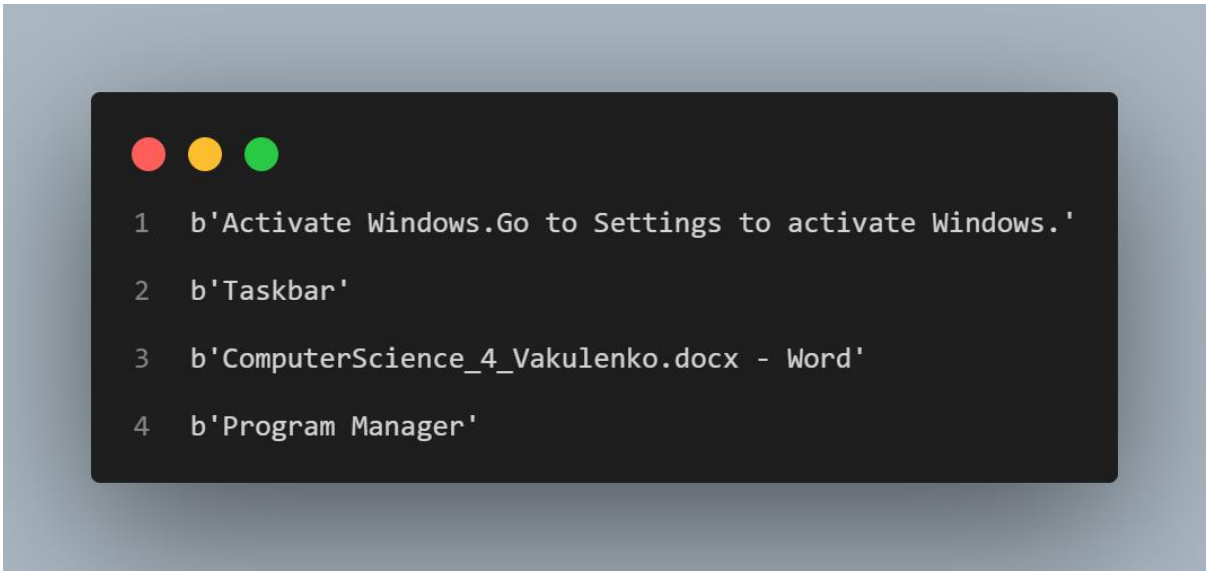


Рисунок 1.6 – Результат використання pywinauto [18]

Як можна побачити, обидва методи повертають багато інших результатів. Але також можна зробити висновок, що на перетині результатів цих двох методів лишається тільки:

a) відкрите користувачем вікно “ComputerScience_4_Vakulenko.docx – Word”
b) Program Manager

Отже для сортування можна використати перетин результатів win32gui.EnumWindows [5] та Desktop(backend="uia").windows() за виключенням “Program Manager”.

1.3 Результати дослідження і постановка задачі для розробки додатку

За результатами проведеного аналізу функціоналу наявних програмних рішень вирішено реалізувати мінімальний достатній функціонал для роботи з шаблонами і вікнами, як-от:

a) Сканування вже розміщенних вікон на робочому столі, відмовившись від надання додаткового функціоналу для розміщення вікон;
b) Збереження виключно розмірів вікон, відмовившись від прив’язки програм до шаблону;
c) Застосування шаблону шляхом вибору однієї зі списку відкритих програм для кожного місця на ньому, відмовившись від реалізації функціоналу перетягування вікон в шаблон.

І сконцентруватися на концептуально нових функціональних рішеннях, як-от:

d) Генерація шаблонів з використанням штучного інтелекту;
e) Голосове керування для розміщення вікон робочого столу.

За результатами проведеного дослідження наявного функціоналу мови Python [1] для керування вікнами вирішено використати такі розширення, як PyWin32 [2] та PyWinAuto [18].

На основі дослідження, потрібно розробити клас Window, в якому, за допомогою обраних розширень, реалізувати наступні методи:

a) Сканування вікон користувача;

b) Отримання таких даних про вікно, як-от: назва, координати та властивості;
c) Відкриття та позиціонування вікна.

Також на основі класу Window потрібно розробити клас Layout, в якому, за допомогою функціоналу класу Window, реалізувати наступні методи:

a) Сканування шаблона;
b) Генерація шаблона;
c) Збереження шаблона;
d) Читання шаблона;
e) Видалення шаблона;
f) Застосування шаблона для розміщення вікон.

В результаті використати розроблений функціонал в попередньо розробленому графічному інтерфейсі.

2 Основний функціонал

2.1 Графічний інтерфейс

Графічний інтерфейс додатку реалізований мовою Python [1] з використанням пакету tkinter [7].

Користувач має доступ до збережених та стандартних шаблонів. В нижній частині інтерфейсу розміщення кнопка для сканування шаблону з екрану. Відсканований шаблон одразу з’являється серед можливих варіантів, оскільки інтерфейс додатку оновлюється з певним інтервалом.

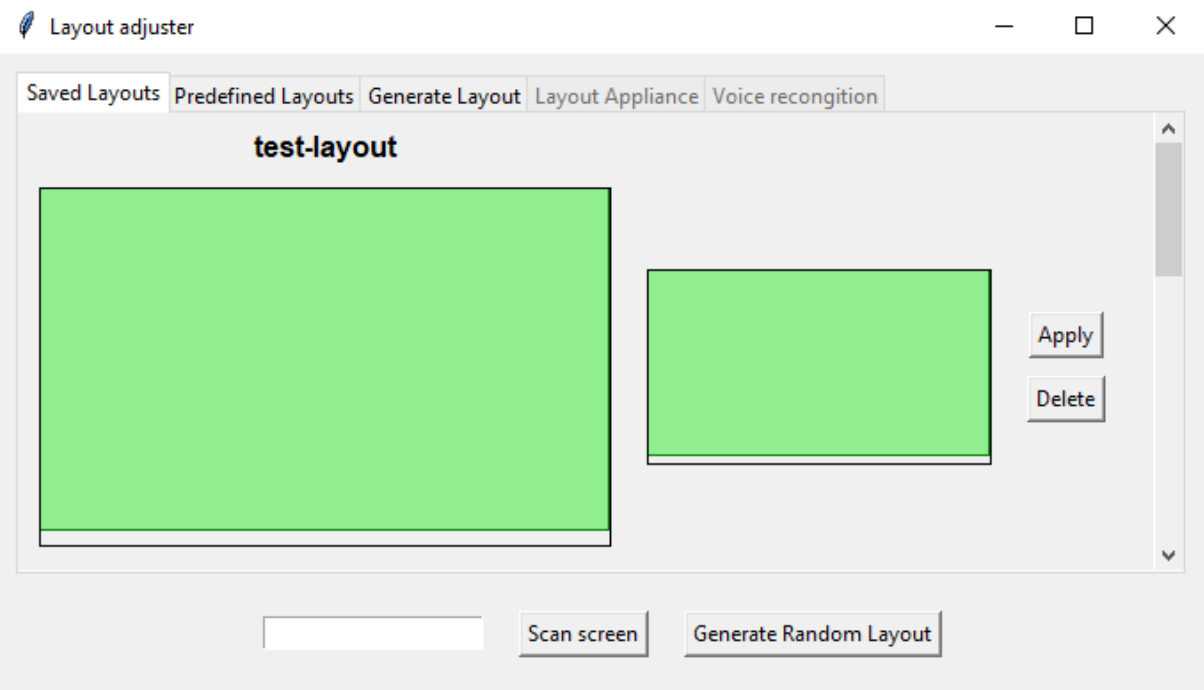


Рисунок 2.1 – Вікно збережених шаблонів

Після натиску на кнопку “Apply”, користувач отримує доступ до вікна використання шаблону. Тут він може вибрати вікна, які він хоче розмістити, та використати шаблон.

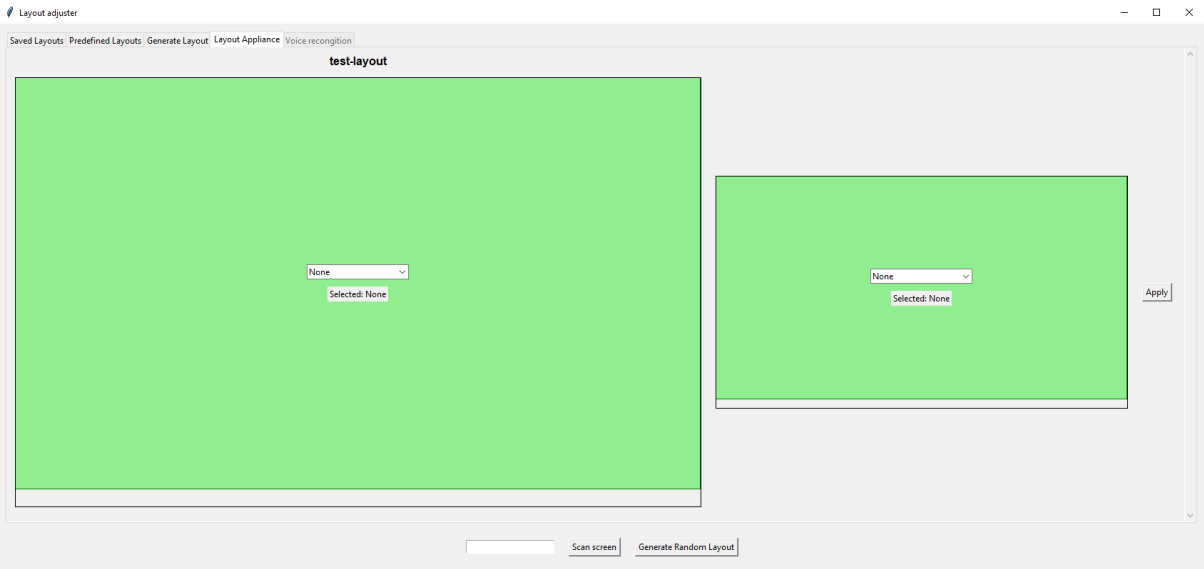


Рисунок 2.2 – Вікно використання шаблона

Користувач також має декілька можливостей для генерації шаблону. Можна згенерувати довільний шаблон. В ньому буде використано 2-

4 вікна. Додатково можна вказати 2 параметри (фах та завдання). Ці параметри будуть враховані при генерації і можуть покращити результат.

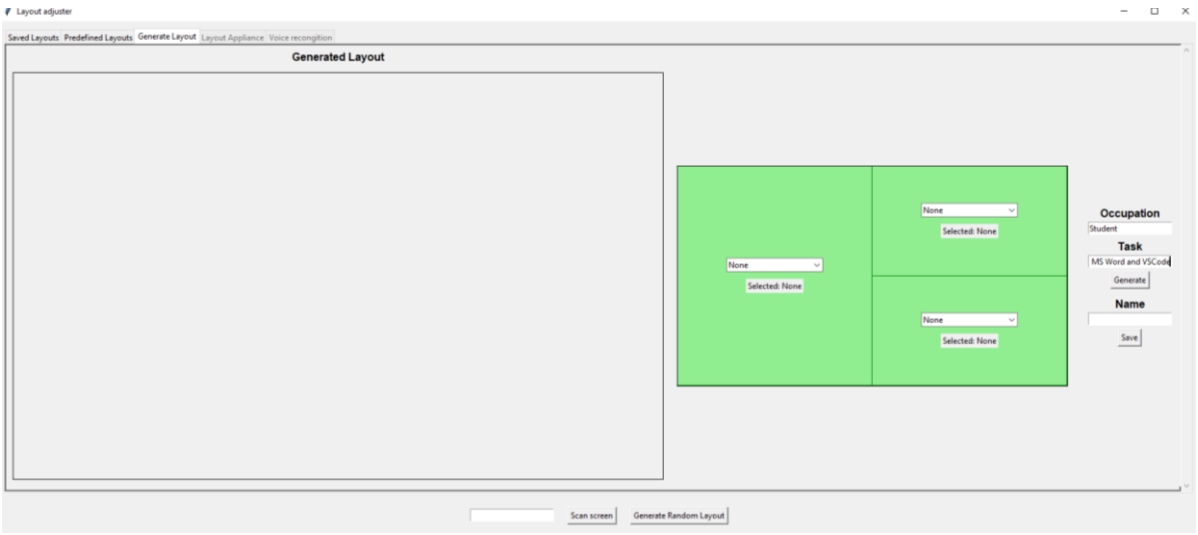


Рисунок 2.3 – Генерація шаблонів

Після роспізнавання фрази для активації голосового керування, відкривається вікно, яке інформує користувача, що він може надавати голосові команди.

Рисунок 2.4 – Вікно прослуховування голосових команд

2.2 Формат збереження даних

В даній версії додатку використаний базовий формат збереження даних. Він представляє собою об’єкт, який зберігає координати моніторів і вікон. Даний формат перекриває базові потреби. Разом з тим, він може бути розширений шляхом збереження певних властивостей вікон, базовим вибором вікон, тощо. Вибір, зроблений на користь простішого формату, пояснюється бажанням приділити більше часу дослідженню, реалізувавши мінімальний достатній базовий функціонал для роботи з додатком.

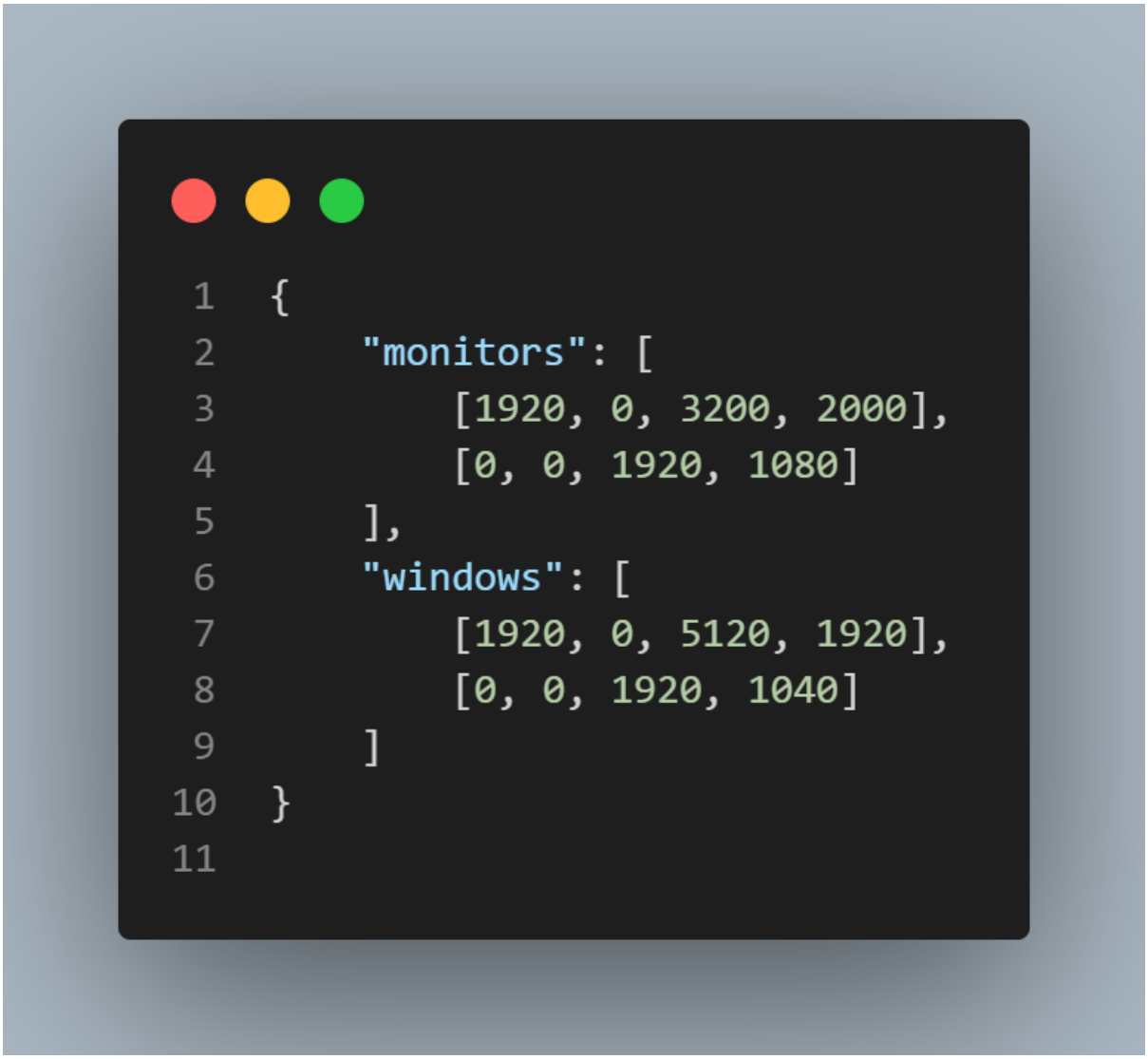


Рисунок 2.5 – Формат збереження шаблону

2.3 Алгоритми для роботи з шаблонани

2.3.1 Алгоритм створення шаблону

Скануваннння і збереження шаблонів відбувається в декілька етапів.

На першому етапі програма сканує і відфільтровує вікна за вже описаним в Розділі 1.2 алгоритмом. Також відфільтровуються закриті вікна та вікна з порожніми заголовками.

```python
@classmethod
def getOpenedWindows(cls, appTitle):
    windows = Window.get_visible()

    titles = list(dict.fromkeys([
        *cls.getPyWinTitles(),
    ]))

    res = []

    for w in windows:
        window_title = w.title
        window_class = w.window_class

        if (
            window_title != '' and
            window_title != appTitle and
            window_title in titles and
            window_class != 'Progman'and
            not w.is_minimized and
            w.is_visible and
            w.is_window
        ):
            res.append(w)

    return res
```

```python
@classmethod
def getPyWinTitles(cls):
    windows = Desktop(backend="uia").windows()
    titles = []

    for w in windows:
        try:
            window_text = w.window_text()
            if (window_text != ''):
                titles.append(window_text)
        except Exception as e:
            print(e)

    return titles
```

Рисунок 2.6 – Отримання списку вікон

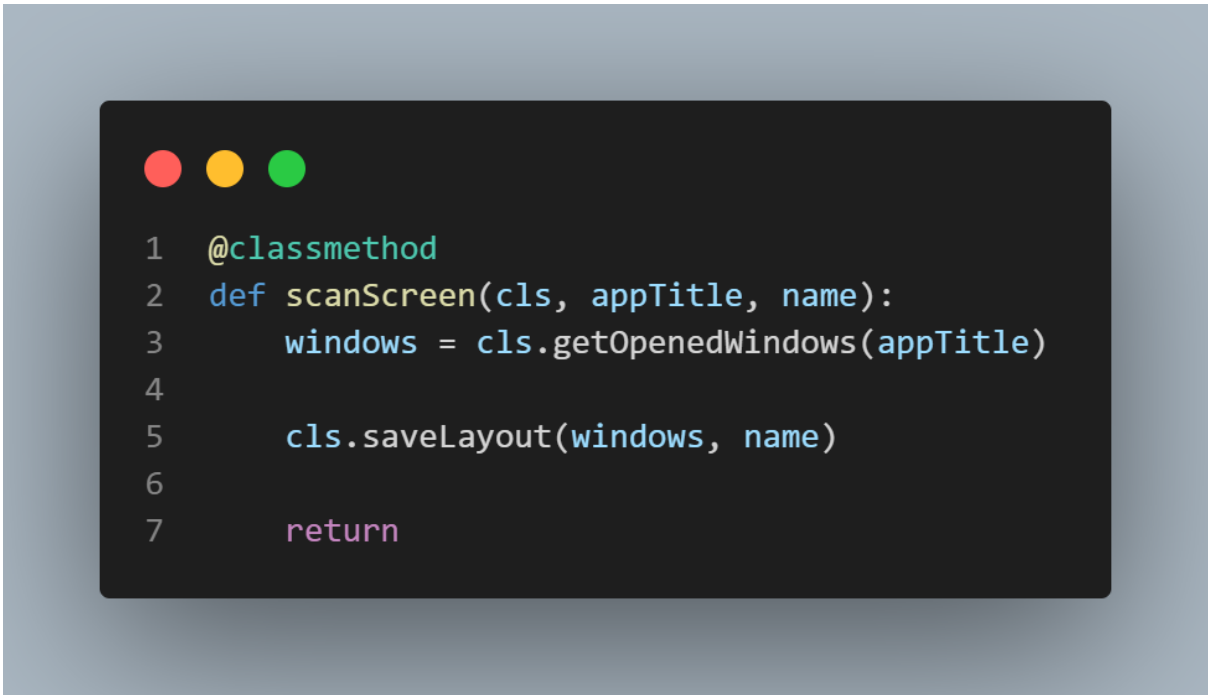
Рисунок 2.7 – Отримання заголовків списку вікон

На другому етапі скануються монітори з використанням пакету screeninfo [8].

```python
@classmethod
def getMonitorsRects(cls):
    monitors = []

    for monitor in get_monitors():
        monitors.append((monitor.x, monitor.y, monitor.width, monitor.height))

    return monitors
```

Рисунок 2.8 – Сканування координат моніторів

На заключному етапі всі дані зберігаються в локальний файл.



```python
@classmethod
def scanScreen(cls, appTitle, name):
    windows = cls.getOpenedWindows(appTitle)

    cls.saveLayout(windows, name)

    return
```

Рисунок 2.9 – Основна функція

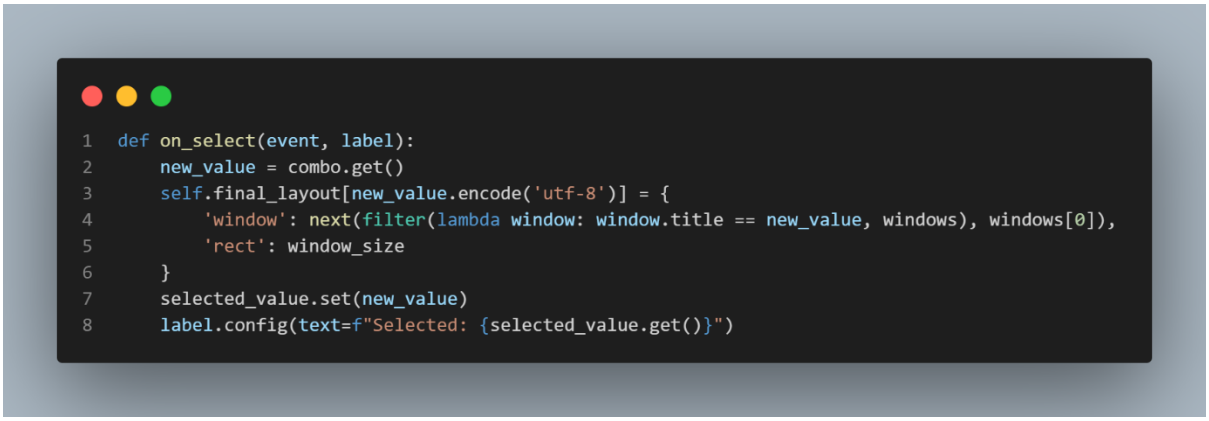
```python
@classmethod
def saveLayout(cls, windows, name):
    res = []
    monitors = cls.getMonitorsRects()

    for w in windows:
        window_rect = w.rect

        res.append(window_rect)

    isExist = os.path.exists(dir)
    if not isExist:
        os.makedirs(dir)
        print("The new directory is created!")

    with open(dir + name + '.json', 'w') as file:
        file.write(json.dumps({'monitors': monitors, 'windows': res}))

    return
```

Рисунок 2.10 – Функція збереження даних в локальний файл

2.3.2 Алгоритм застосування шаблону

Алгоритм застосування шаблону для розміщення вікон також відбувається в декілька етапів.

На першому етапі, використовуючи графічний інтерфейс, користувач обирає вікна для розміщення. В об’єкті зберігається вікно і координати для розміщення.



```
def on_select(event, label):
    new_value = combo.get()
    self.final_layout[new_value.encode('utf-8')] = {
        'window': next(filter(lambda window: window.title == new_value, windows), windows[0]),
        'rect': window_size
    }
    selected_value.set(new_value)
    label.config(text=f"Selected: {selected_value.get()}")
```

Рисунок 2.11 – Функція збереження даних в змінну

На другому етапі обрані вікна розміщуються за відповідними координатами.

```
@classmethod
def applyLayout(cls, final_layout):
    for key in list(final_layout.keys()):
        window_size = final_layout[key]['rect']

        window_x = window_size[0]
        window_y = window_size[1]
        window_width = window_size[2]
        window_height = window_size[3]

        final_layout[key]['window'].set_location(window_x, window_y, window_width, window_height)
```

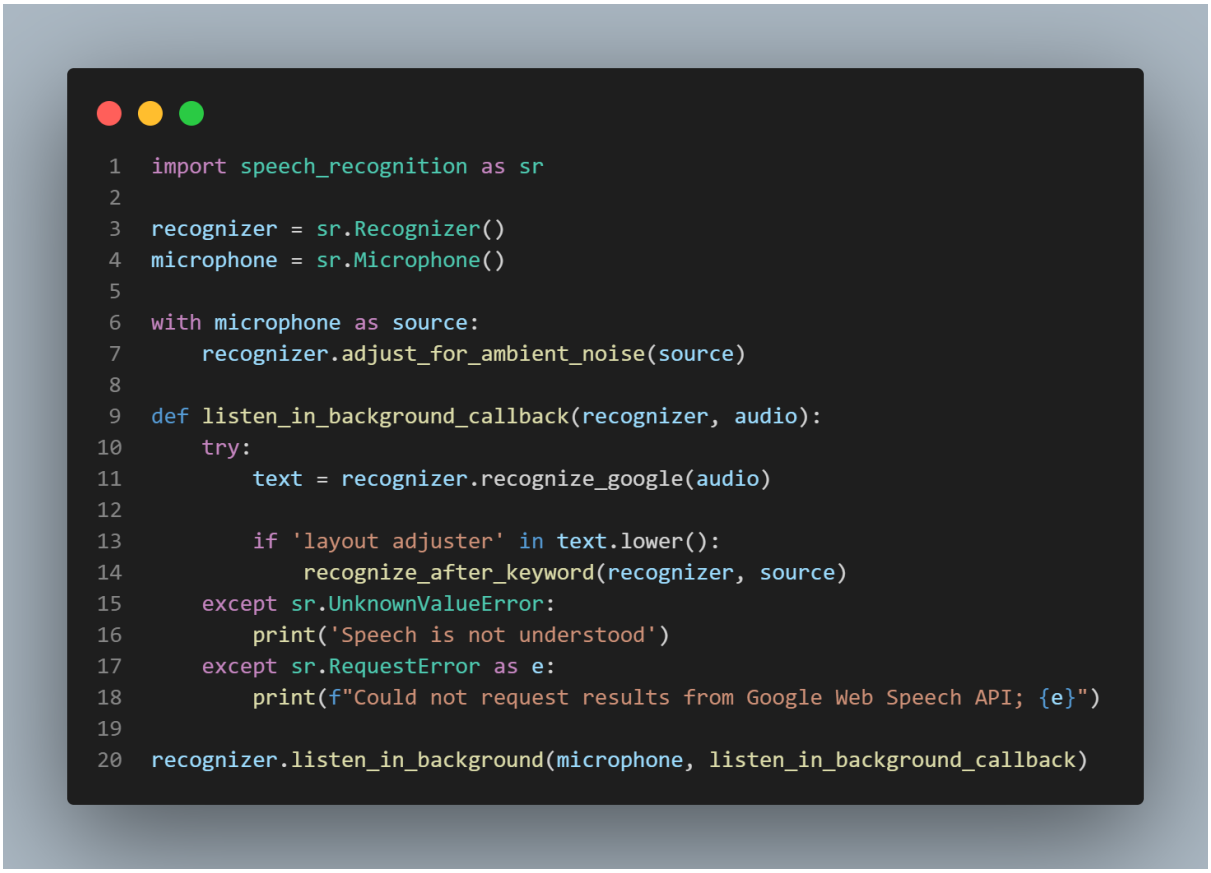
Рисунок 2.12 – Функція розміщення вікон

3 Голосове керування

3.1 Розпізнавання голосу

Голосове керування реалізоване з використанням пакета speech_recognition [14] в 2 етапи:

Перший етап – це фонове прослуховування, яке починається одночасно із запуском застосунку. Реалізований алгоритм за допомогою класів Recognizer та Microphone. Мікрофон використовується в якості джерела із застосування алгоритму фільтрування шуму. Після чого за допомогою методів listen_in_background та recognize_google реалізовується фонове прослуховування і подальший аналіз на наявність ключового слова.



```python
import speech_recognition as sr

recognizer = sr.Recognizer()
microphone = sr.Microphone()

with microphone as source:
    recognizer.adjust_for_ambient_noise(source)

def listen_in_background_callback(recognizer, audio):
    try:
        text = recognizer.recognize_google(audio)

        if 'layout adjuster' in text.lower():
            recognize_after_keyword(recognizer, source)
    except sr.UnknownValueError:
        print('Speech is not understood')
    except sr.RequestError as e:
        print(f"Could not request results from Google Web Speech API; {e}")

recognizer.listen_in_background(microphone, listen_in_background_callback)
```

Рисунок 3.1 – Фонове прослуховування

Другий етап – це візуальна демонстрація зміни стану застосунку на очікування голосової команди, обробка команди і її виконання. Для цього відкривається вікно прослуховування голосової команди. Додатково, якщо додаток згорнутий, він буде розгорнутий. Далі відбувається прослуховування, після чого отримана команда оброблюється за допомогою штучного інтелекту та виконується відповідна дія (дана частина алгоритму буде описана в наступних підрозділах).

```python
import speech_recognition as sr
import win32gui
import win32con
from aiRequest import generate_ai_suggestion
from aiActions import ai_actions, split_key

def recognize_after_keyword(recognizer, source, app):
    app.notebook.tab(3, state="normal")
    app.notebook.select(3)

    hwnd = win32gui.FindWindowEx(0,0,0, "Layout adjuster")

    win32gui.ShowWindow(hwnd, win32con.SW_MINIMIZE)
    win32gui.ShowWindow(hwnd, win32con.SW_RESTORE)

    while True:
        try:
            audio = recognizer.listen(source, timeout=60)
            text = recognizer.recognize_google(audio)

            aiResponse = generate_ai_suggestion(text)

            parts = split_key(aiResponse)
            ai_actions.execute_callback(parts[0], parts[1])
            app.notebook.tab(3, state="disabled")
            app.notebook.select(0)

            break

        except sr.UnknownValueError:
            print('Speech after keyword is not understood')
        except sr.RequestError as e:
            print(f"Could not request results from Google Web Speech API; {e}")
        except sr.WaitTimeoutError:
            print('Listening ended due to timeout')
            break
```

Рисунок 3.2 – Алгорит після відловлювання ключового слова

3.2 Використання API штучного інтелекту

3.2.1 Обробка голосових команд

Для комунікації зі штучним інтелектом використано пакет OpenAI [19] і модель gpt-3.5-turbo-0125 [20]. Даний вибір обумовлений виключно співвідношенням ціни до новизни моделі, адже завдання ставляться

доволі базові і не потребують більш потужних і, як наслідок, дорогих моделей.

```python
client = OpenAI(
    api_key="",
)

def generate_ai_suggestion(sysPrompt, userPrompt = ''):
    completion = client.chat.completions.create(
        messages=[
            {"role": "system", "content": sysPrompt},
            {"role": "user", "content": userPrompt}
        ],
        model="gpt-3.5-turbo-0125",
    )

    return completion.choices[0].message.content.replace('"', '')
```

Рисунок 3.3 – Функція запиту до API штучного інтелекту

```python
def generate_action(prompt):
    sysPrompt = f'''

    You are an assistant, skilled in defining which action user wants to perform. Actions that are allowed are:
        {action_string}
    Do not return anything but action name specified in quotes. If name contains brackets, keep brackets and replace its content with relative content.

    '''
    userPrompt = 'User said "' + prompt + '". Which one of the described actions user wants to perform.'

    return generate_ai_suggestion(sysPrompt, userPrompt)
```

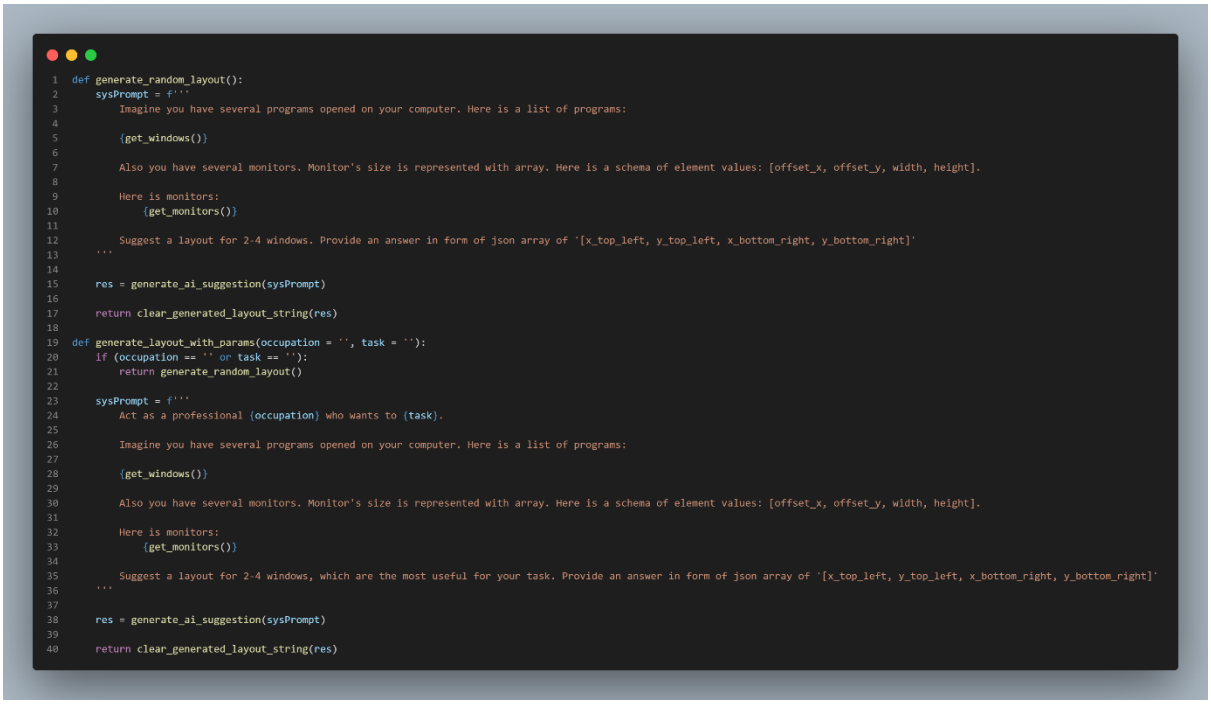
Рисунок 3.4 – Запит для обробки голосової команди

Запит до ШІ попудований відповідно до мануалу з інженерії запитів. В даному випадку використані такі техніки, як:

a) Використання конструкції «Ти є ...» з детальним описом потрібних навичок для виконання поставленої задачі;
b) Нумерований список для вибору;
c) Опис формату повернення даних;
d) Обгортання користувацької інформації в лапки;
e) Чітке формулювання задачі.

3.2.2 Генерування шаблону

В даному застосунку передбачено декілька можливостей генерації шаблону. Користувач має можливість згенерувати довільний шаблон або вказати параметри для покращення результатів.



```
def generate_random_layout():
    sysPrompt = f'''
        Imagine you have several programs opened on your computer. Here is a list of programs:

        {get_windows()}

        Also you have several monitors. Monitor's size is represented with array. Here is a schema of element values: [offset_x, offset_y, width, height].

        Here is monitors:
            {get_monitors()}

        Suggest a layout for 2-4 windows. Provide an answer in form of json array of '[x_top_left, y_top_left, x_bottom_right, y_bottom_right]'
    '''

    res = generate_ai_suggestion(sysPrompt)

    return clear_generated_layout_string(res)

def generate_layout_with_params(occupation = '', task = ''):
    if (occupation == '' or task == ''):
        return generate_random_layout()

    sysPrompt = f'''
        Act as a professional {occupation} who wants to {task}.

        Imagine you have several programs opened on your computer. Here is a list of programs:

        {get_windows()}

        Also you have several monitors. Monitor's size is represented with array. Here is a schema of element values: [offset_x, offset_y, width, height].

        Here is monitors:
            {get_monitors()}

        Suggest a layout for 2-4 windows, which are the most useful for your task. Provide an answer in form of json array of '[x_top_left, y_top_left, x_bottom_right, y_bottom_right]'
    '''

    res = generate_ai_suggestion(sysPrompt)

    return clear_generated_layout_string(res)
```

Рисунок 3.5 – Запити для генерації шаблонів

У випадку з довільним шаблоном, в запит передається список відкритих вікон та розміри моніторів і вимагається запропонувати розміри 2-4 вікон відповідно до занадих даних.

У випадку з генерацією шаблону з використанням параметрів, відповідь ШІ покращується за рахунок введення деталей щодо завдання користувача.

```python
def get_windows():
    windows = Layout.getOpenedWindows('Layout adjuster')

    windows_string = []

    for w in windows:
        windows_string.append(f"- {w.title.encode('utf-8')}")

    return '\n\t\t'.join(windows_string)

def get_monitors():
    monitors = Layout.getMonitorsRects()

    monitors_string = []

    for i, m in enumerate(monitors):
        monitors_string.append(f"- Monitor {i}: {json.dumps(m)}")

    return '\n\t\t\t'.join(monitors_string)
```

Рисунок 3.6 – Конвертація даних для запитів

Запити до ШІ попудований відповідно до мануалу з інженерії запитів. В даному випадку використані такі техніки, як:

a) Використання конструкції «Дій як професійний ...» з описом завдання користувача;
b) Чітке формулювання задачі;
c) Опис формату повернення даних.

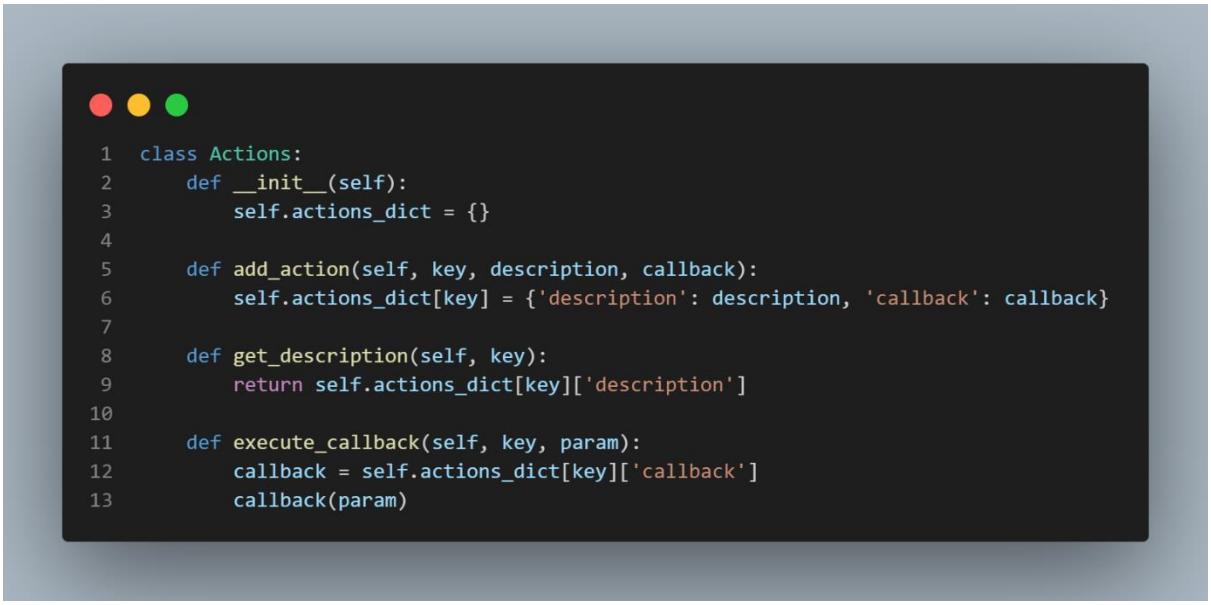
```python
def read_layout_from_ai_response(json_str):
    data = json_str
    if (isinstance(data, bytes)):
        data = data.decode('utf-8')

    json_strs = re.findall(r'\[.*?\]', data)

    return list(map(json.loads, json_strs))

def clear_generated_layout_string(layout_str):
    monitors = Layout.getMonitorsRects()

    layout = read_layout_from_ai_response(layout_str)
    windows = []

    for window_rect in layout:
        if isinstance(window_rect, list) and not isinstance(window_rect, str):
            windows.append(window_rect)

    res = {
        'name': 'Generated Layout',
        'windows': windows,
        'monitors': monitors,
    }

    return json.dumps(res)
```

Рисунок 3.6 – Обробка відповіді ШІ

Відповідь ШІ повертається в текстовому форматі, тому необхідно її обробити. Перша фінкція виділяє зі строки масиви з розмірами вікон і конвертує їх в JSON. Друга – конвертує отримані результати в стандартний внутрішній формат шаблону.

3.3 Виконання дій

Для роботи з діями був розроблений клас Actions, який дозволяє додавати дії, діставати потрібну інформацію та виконувати відповідну дію.

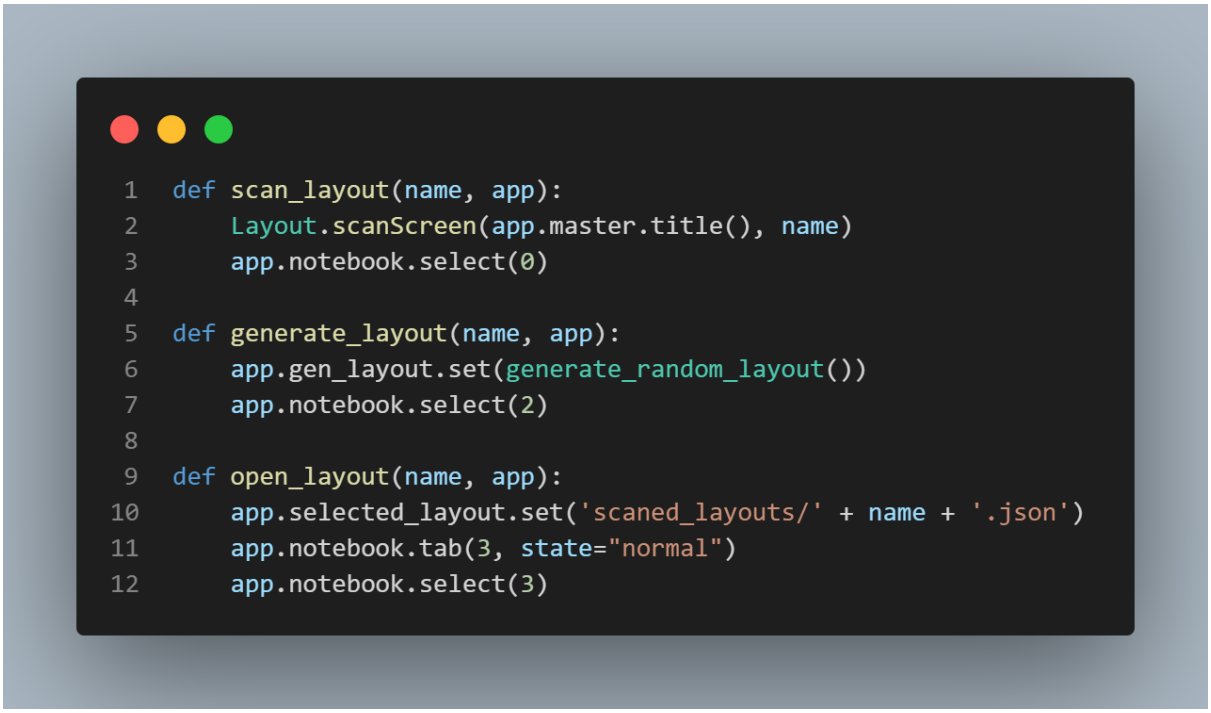


```python
class Actions:
    def __init__(self):
        self.actions_dict = {}

    def add_action(self, key, description, callback):
        self.actions_dict[key] = {'description': description, 'callback': callback}

    def get_description(self, key):
        return self.actions_dict[key]['description']

    def execute_callback(self, key, param):
        callback = self.actions_dict[key]['callback']
        callback(param)
```

Рисунок 3.7 – Клас Actions

Також були визначені відповідні можливі дії, а саме:

a) Сканування шаблону;
b) Генерація шаблону;
c) Використання шаблону.



```python
def scan_layout(name, app):
    Layout.scanScreen(app.master.title(), name)
    app.notebook.select(0)

def generate_layout(name, app):
    app.gen_layout.set(generate_random_layout())
    app.notebook.select(2)

def open_layout(name, app):
    app.selected_layout.set('scaned_layouts/' + name + '.json')
    app.notebook.tab(3, state="normal")
    app.notebook.select(3)
```

Рисунок 3.8 – Функції дій

І в решті-решт інаціалізація всіх описаних вище дій з відповідними описами для використання в запитах до штучного інтелекту.

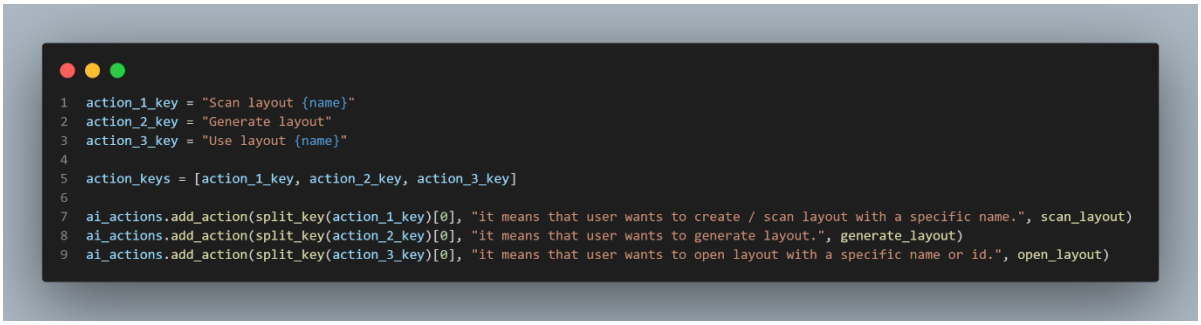


Рисунок 3.9 – Ініціалізація дій

Висновки

Результатом кваліфікаційної роботи є проведене дослідження наявного функціоналу для керування вікнами робочого столу і розроблений додаток для автоматизованого керування вікнами робочого столу.

В ході роботи було досліджено деякі пакети для роботи з вікнами операційної системи Windows [16] з використанням мови програмування Python [1]. На базі дослідженого матеріалу було розроблено функціонал для пришвидшення роботи з вікнами робочого столу шляхом розширення такого поняття, як шаблон, з базових варіантів до необмеженої кількості варіантів.

Також було впроваджено голосове керування і використано можливості штучного інтелекту для додаткового покращення отриманих результатів.

В майбутньому можна покращити:

a) Формат збереження шаблонів, розширивши його за допомогою властивостей вікон, таких як мінімізація, максимізація, тощо.
b) Впровадити стани шаблону;
c) Розширити кількість дій, які користувач може зробити, використовуючи голосове керування.
d) Покращити UI / UX.

Перелік прийнятих скорочень

ШІ – штучний інтелект

ГК – голосове керування

ПП – програмний продукт

ПЗ – програмне забезпечення

Список використаних джерел

[1] 3.12.2 Documentation (python.org)

[2] pywin32 · PyPI

[3] PyPI Download Stats (pypistats.org)

[4] mhammond/pywin32: Python for Windows (pywin32) Extensions (github.com)

[5] win32gui · PyPI

[6] PyPI Download Stats (pypistats.org)

[7] tkinter — Інтерфейс Python до Tcl/Tk — Python 3.12.2 documentation

[8] screeninfo · PyPI

[9] Prompt engineering - OpenAI API

[10] Mizage - Divvy

[11] Home - Soulid Studio

[12] Home - MaxTo

[13] WindowGrid

[14] SpeechRecognition · PyPI

[15] OpenAI API

[16] Технічна документація | Microsoft Learn

[17] GitHub

[18] pywinauto · PyPI

[19] openai · PyPI

[20] Models - OpenAI API