Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки та комп’ютерних технологій

Кафедра системного проектуваннння

Звіт

про проходження виробничої практики

Виконав:
студент групи ФЕІм-12
Товкач Б. М.
Керівник практики:
доц. Ненчук Т. М.

Львів – 2022

Результати виконання практики

Протягом 4 – 25 жовтня 2022 р. взяв участь у роботі семінарів, на яких було представлені плани виконання та дослідження дипломних робіт студентів групи ФЕІм-12, які проходили практику на кафедрі системного проектування.

Було узгоджено план виконання робіт на період проведення виробничої практики. Випробувано розроблені прототипи програм: отримані результати вплинули на майбутнє внесення змін. Були зібрані, оброблені, проаналізовані вихідні дані. Код робочого прототипу програм (додаток А). Візуалізація даних: графіки та таблиці (додаток Б).

Керівником практики був Ненчук Т. М. Опрацювання дипломної роботи велось протягом усього терміну проходження практики і його результати подано в Додатках.

ДОДАТОК А

```
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='blac
k', 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__':
    main()
```

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

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:])

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


if __name__ == "__main__":
    recognition_speech()
```

```
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

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
```

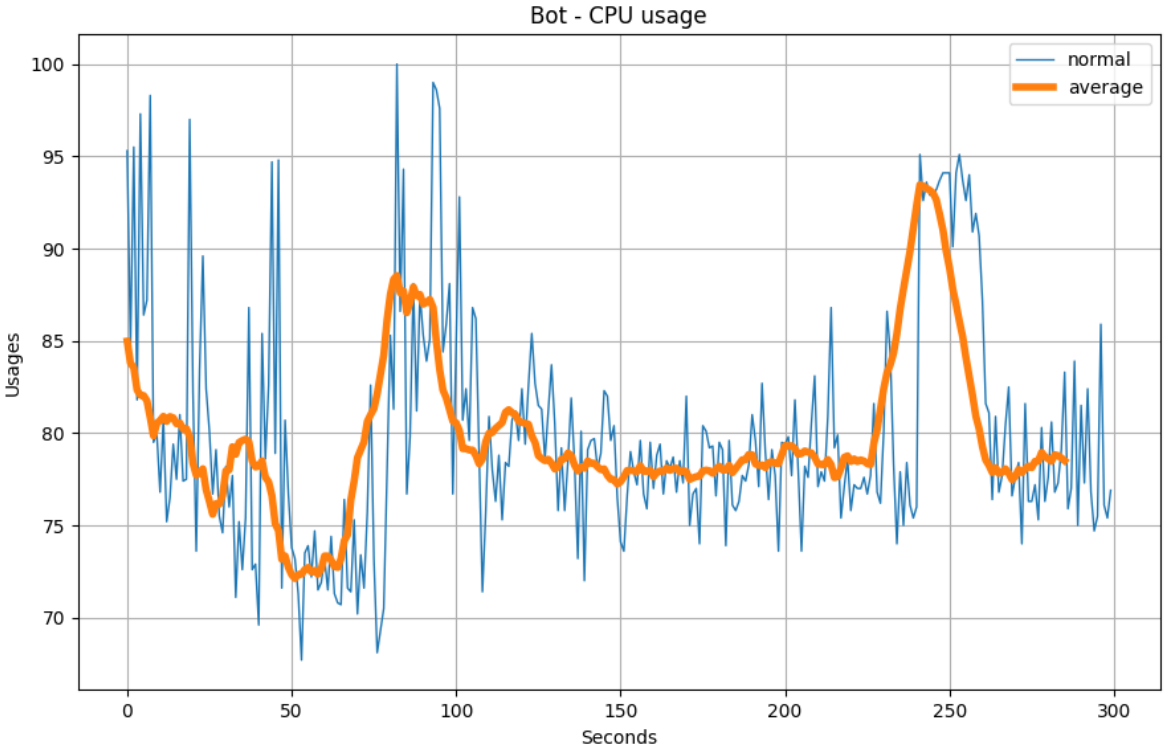
```
    def get_screenshot(self):
        # get the window image data
        wDC = win32gui.GetWindowDC(self.hwnd)
        dcObj = win32ui.CreateDCFromHandle(wDC)
        cDC = dcObj.CreateCompatibleDC()
        dataBitMap = win32ui.CreateBitmap()
        dataBitMap.CreateCompatibleBitmap(dcObj, self.w, 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)
```

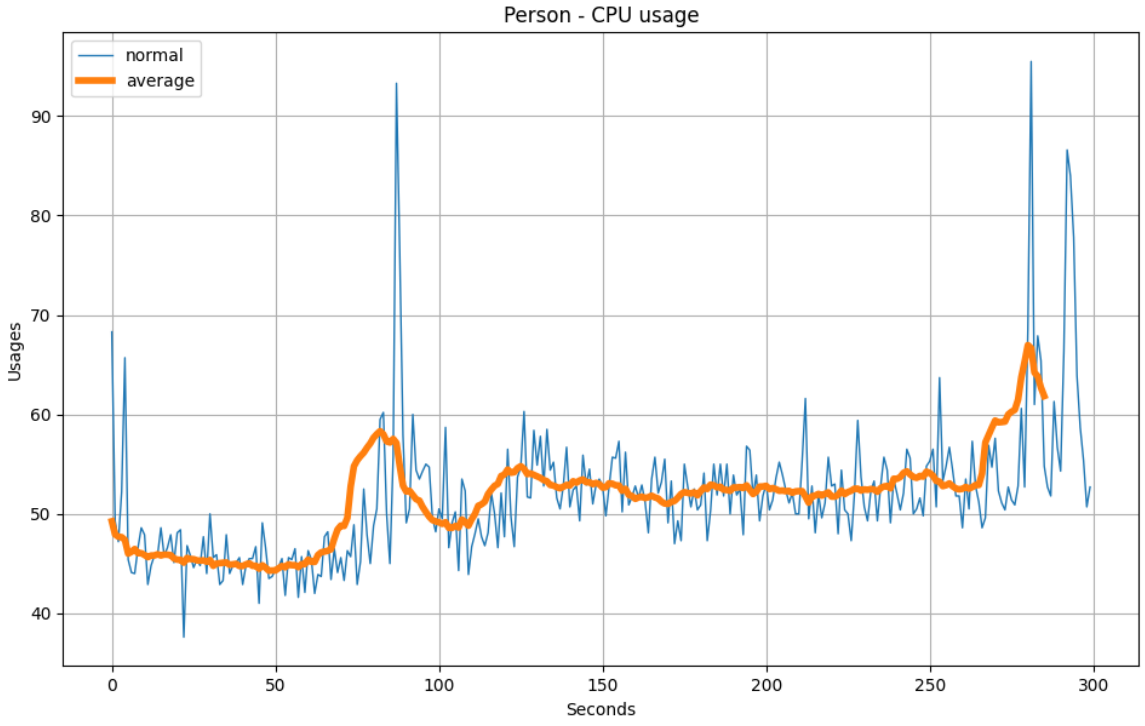
```
    # WARNING: if you move the window being captured after execution is started,
this will
    # return incorrect coordinates, because the window position is only
calculated in
    # the __init__ constructor.
    def get_screen_position(self, pos):
        return (pos[0] + self.offset_x, pos[1] + self.offset_y)

pyinstaller --onefile --windowed --add-data "static/config.json;/static/." --
add-data "static/disconnect_info.png;/static/." --add-data
"static/window_icon.ico;/static/." --hidden-import="utils" --hidden-
import="dashboard" --hidden-import="windowcapture" --hidden-import="components"
--name "AI Bot" main.py
```

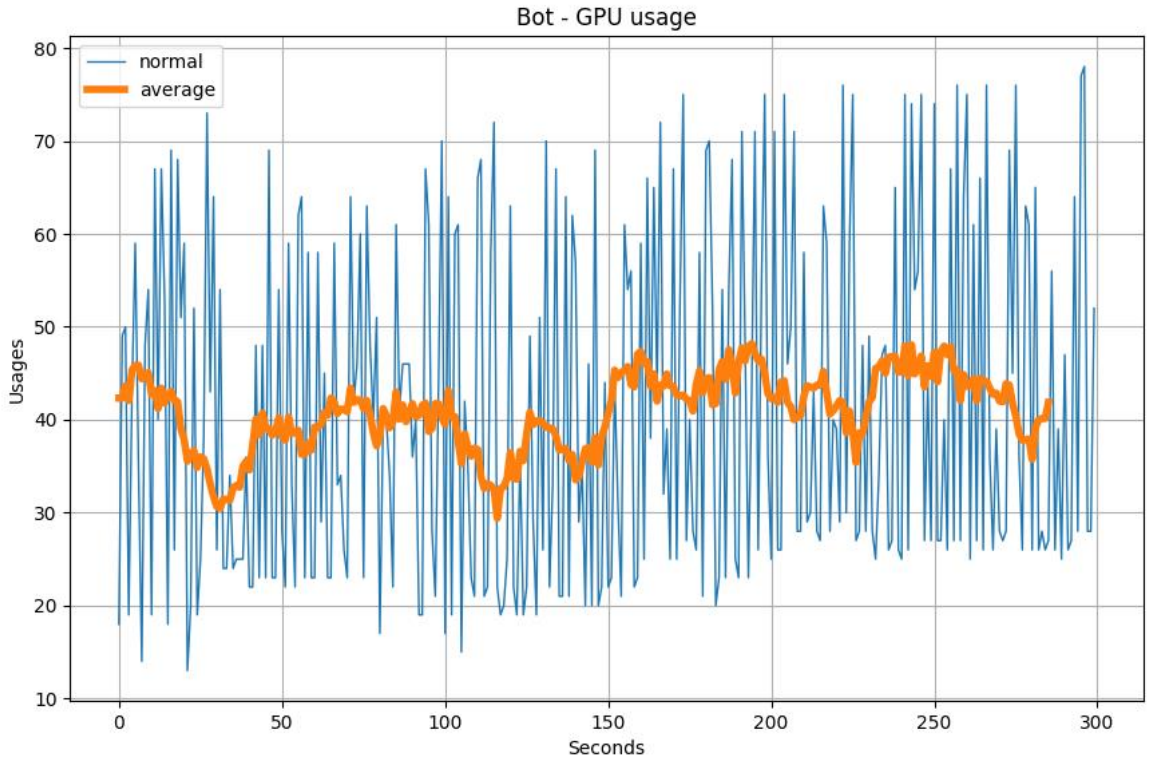
Відео демонстрація за посиланням:

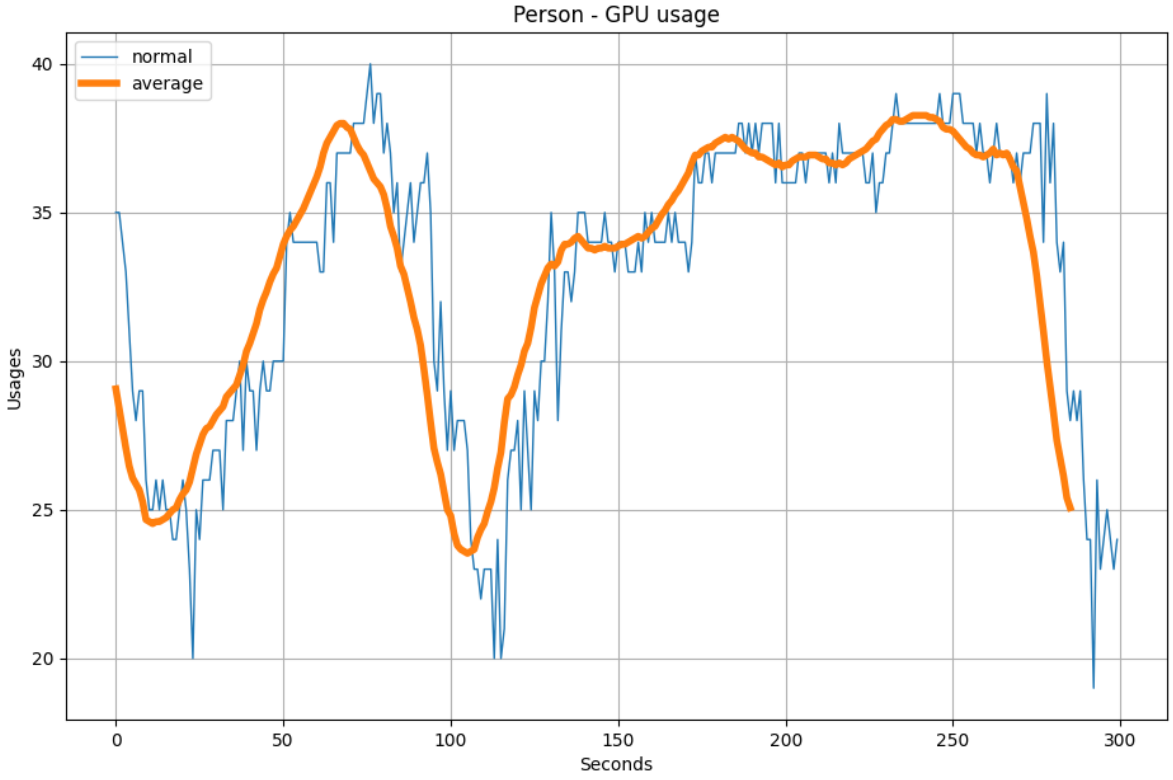
ai_bot_example - YouTube

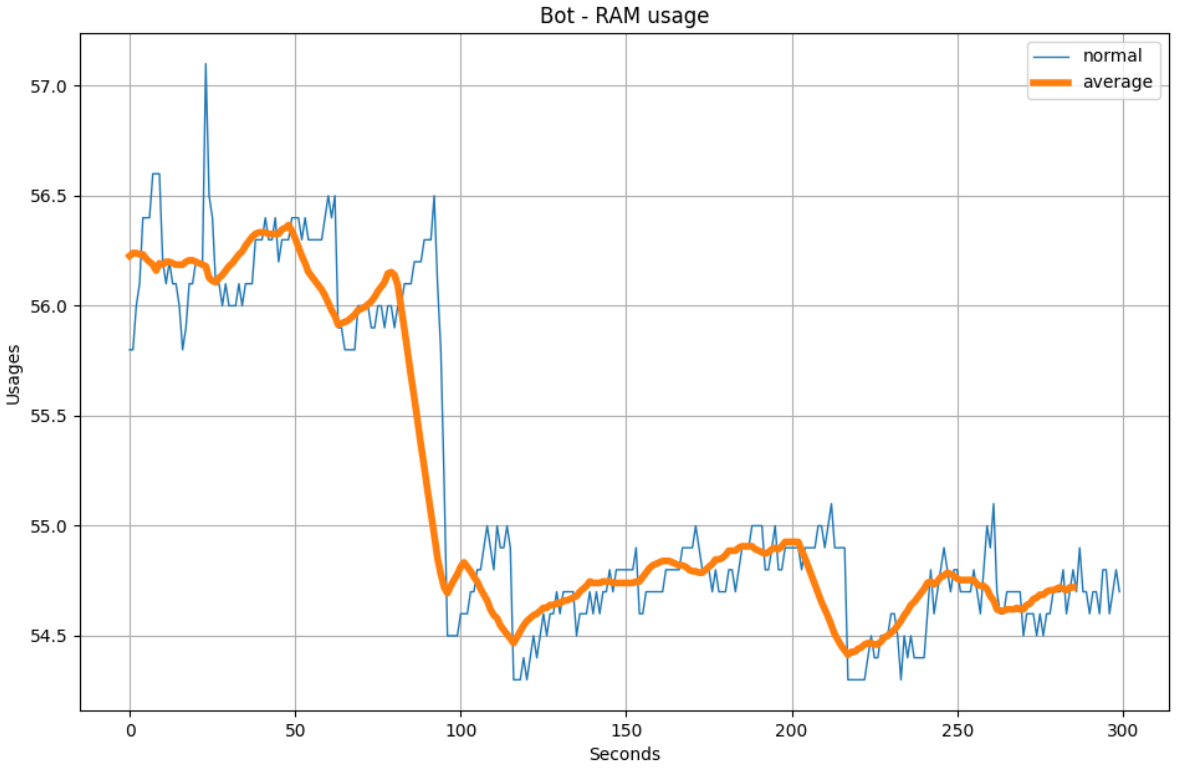
ДОДАТОК Б

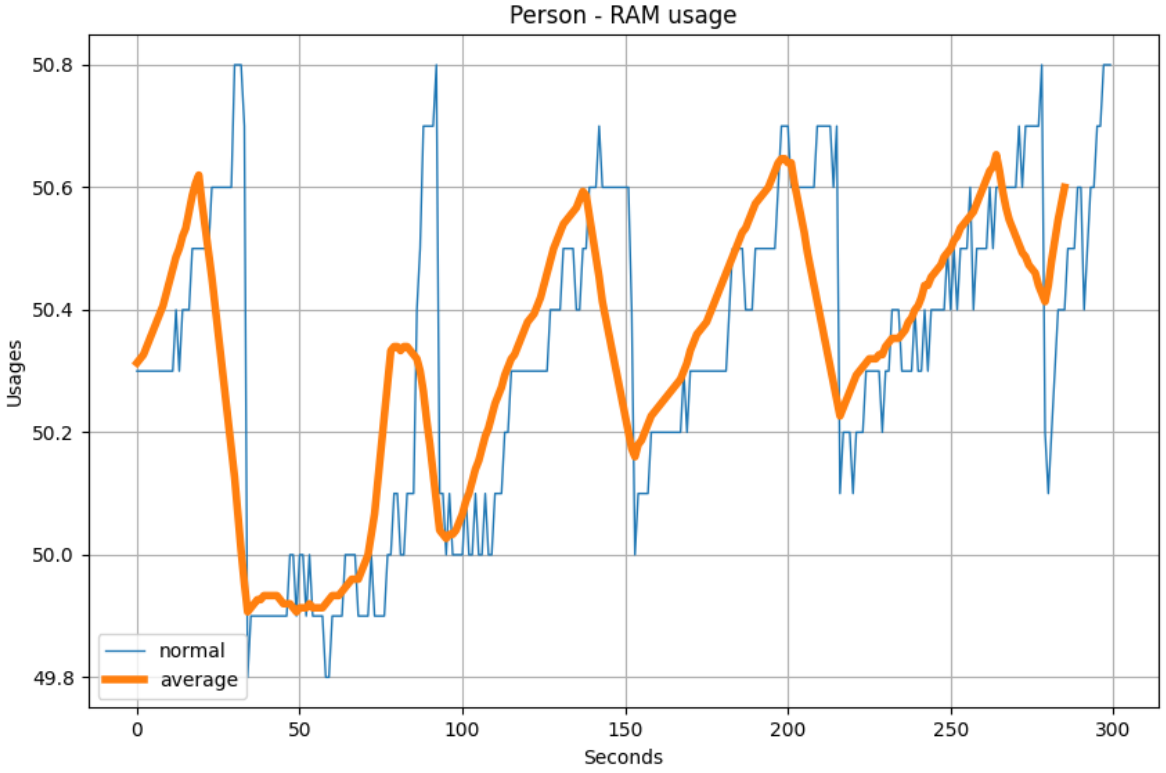

Bot - CPU usage
normal
average
Usages
Seconds
100
95
90
85
80
75
70
0
50
100
150
200
250
300


Person - CPU usage
normal
average
Usages
Seconds
90
80
70
60
50
40
0
50
100
150
200
250
300

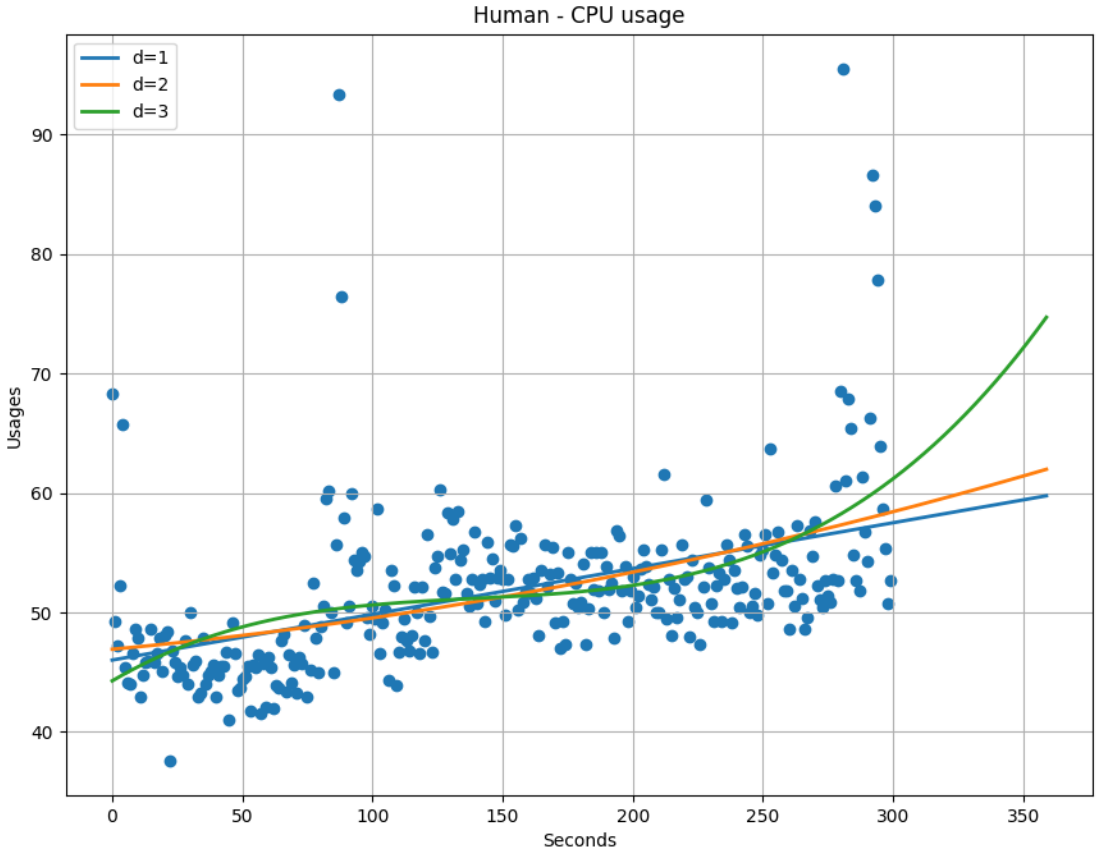

Bot - GPU usage
normal
average
Usages
Seconds
80
70
60
50
40
30
20
10
0
50
100
150
200
250
300

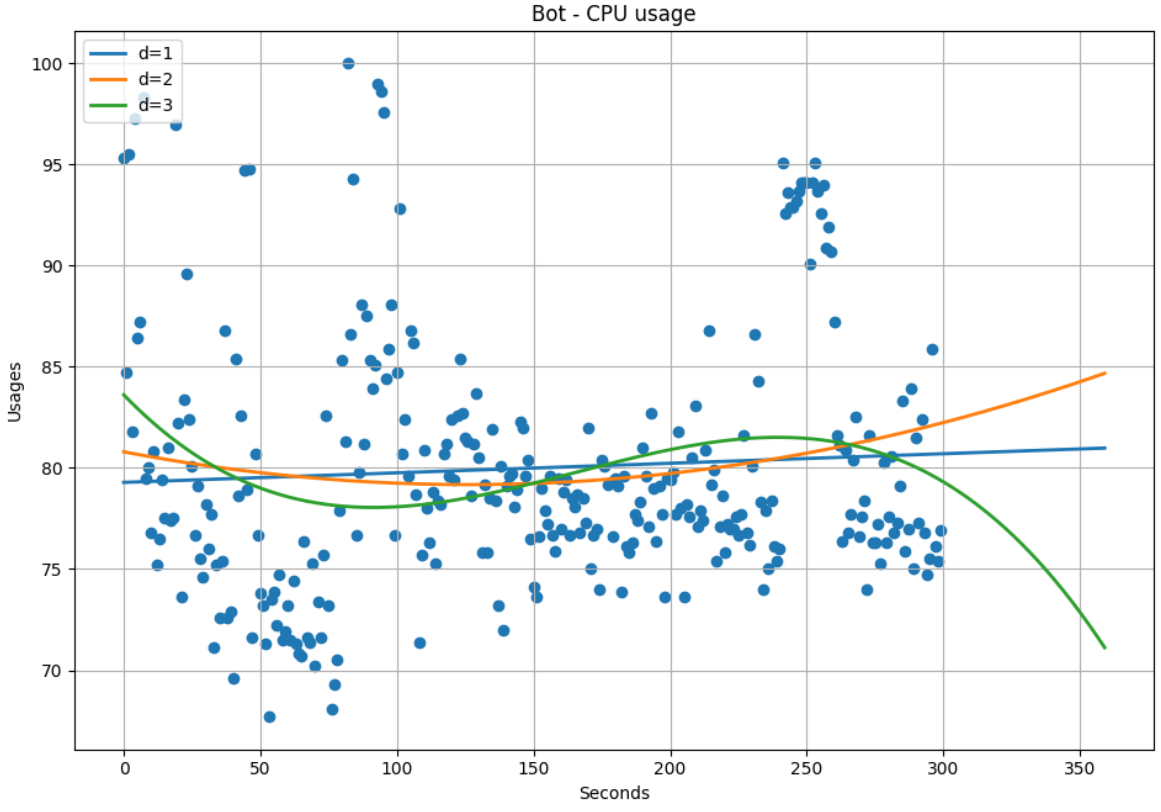

Person - GPU usage
normal
average
Usages
Seconds
40
35
30
25
20
0
50
100
150
200
250
300

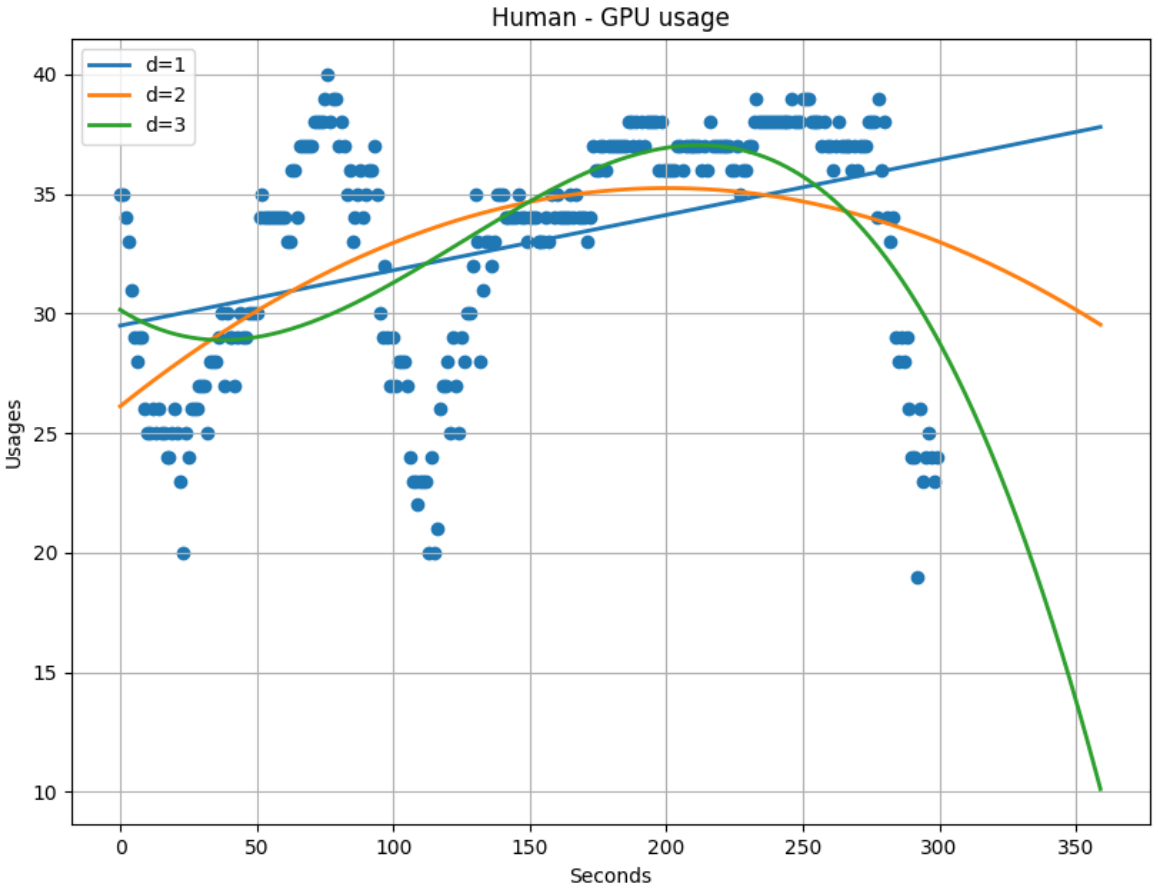

Bot - RAM usage
normal
average
Usages
Seconds
57.0
56.5
56.0
55.5
55.0
54.5
0
50
100
150
200
250
300

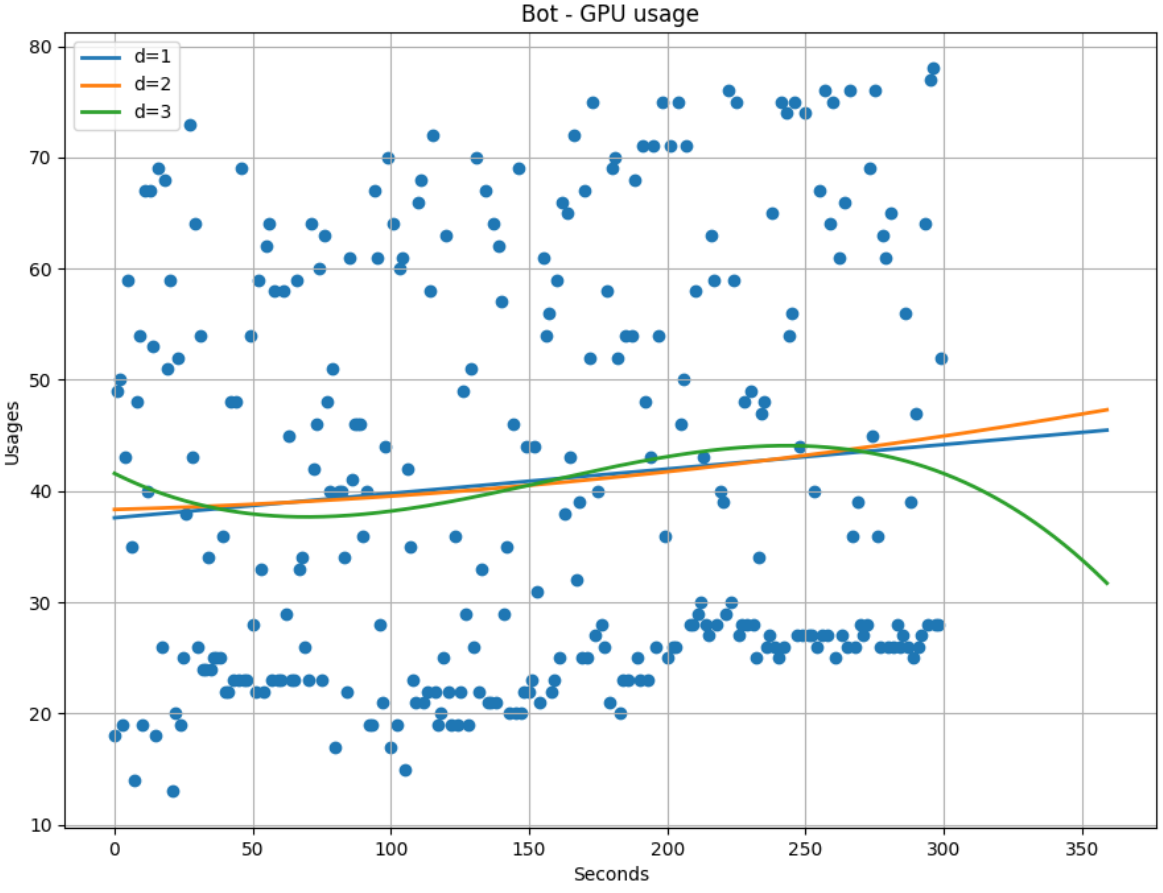

Person - RAM usage
normal
average
Usages
Seconds
50.8
50.6
50.4
50.2
50.0
49.8
0
50
100
150
200
250
300

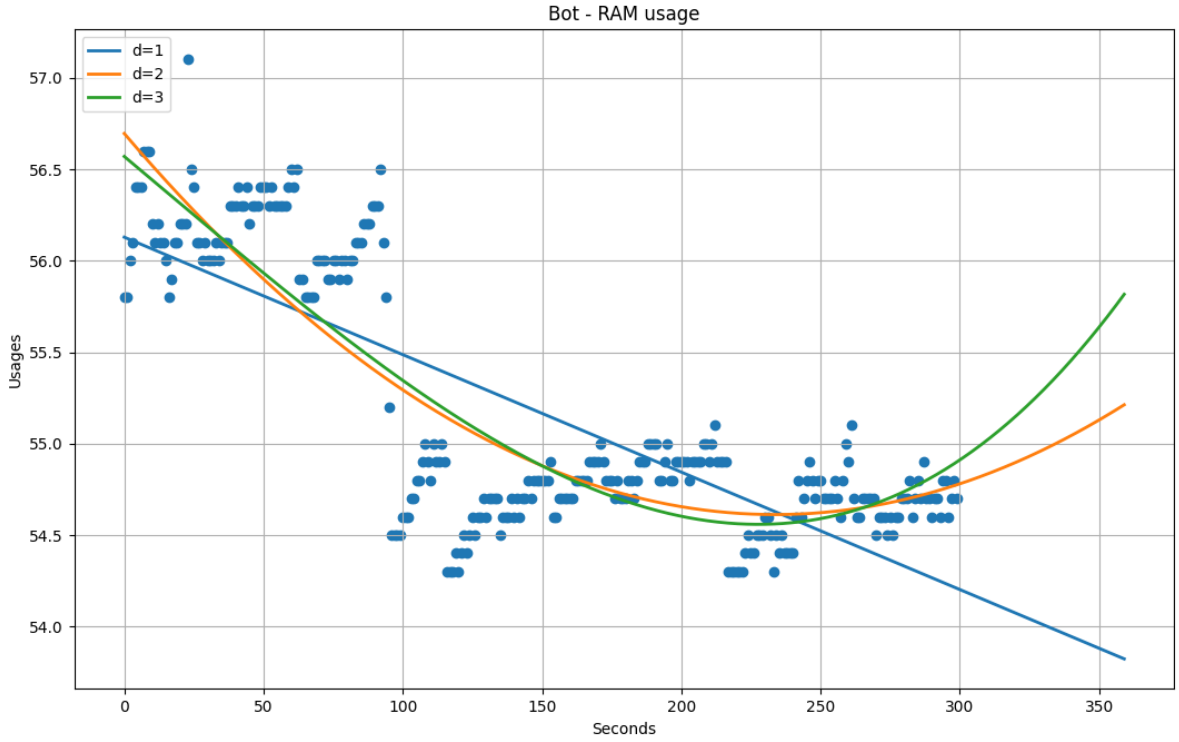
CPU		GPU		RAM	
Human	Bot	Human	Bot	Human	Bot
51.7 %	80.0 %	33.0 %	40.9 %	50.3 %	55.2 %
на 28.3 %		на 7.9		на 1.9 %	

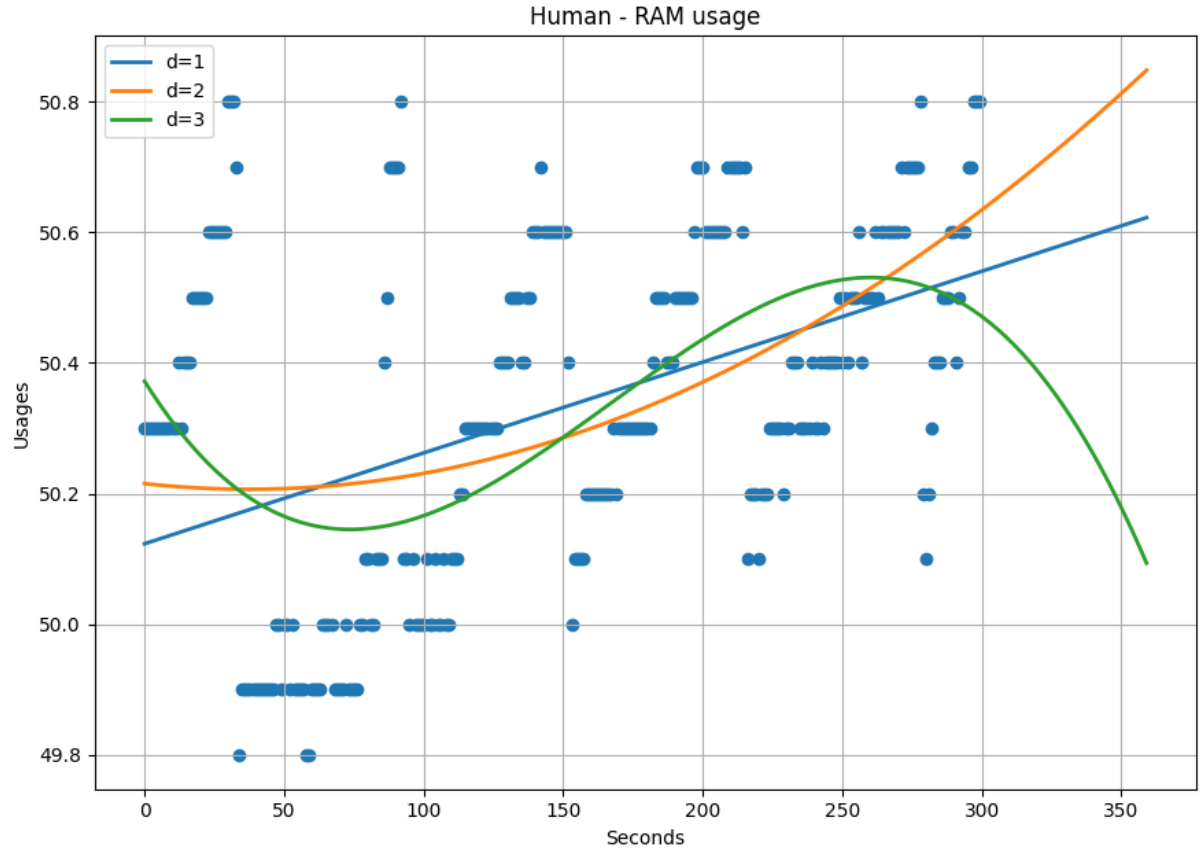


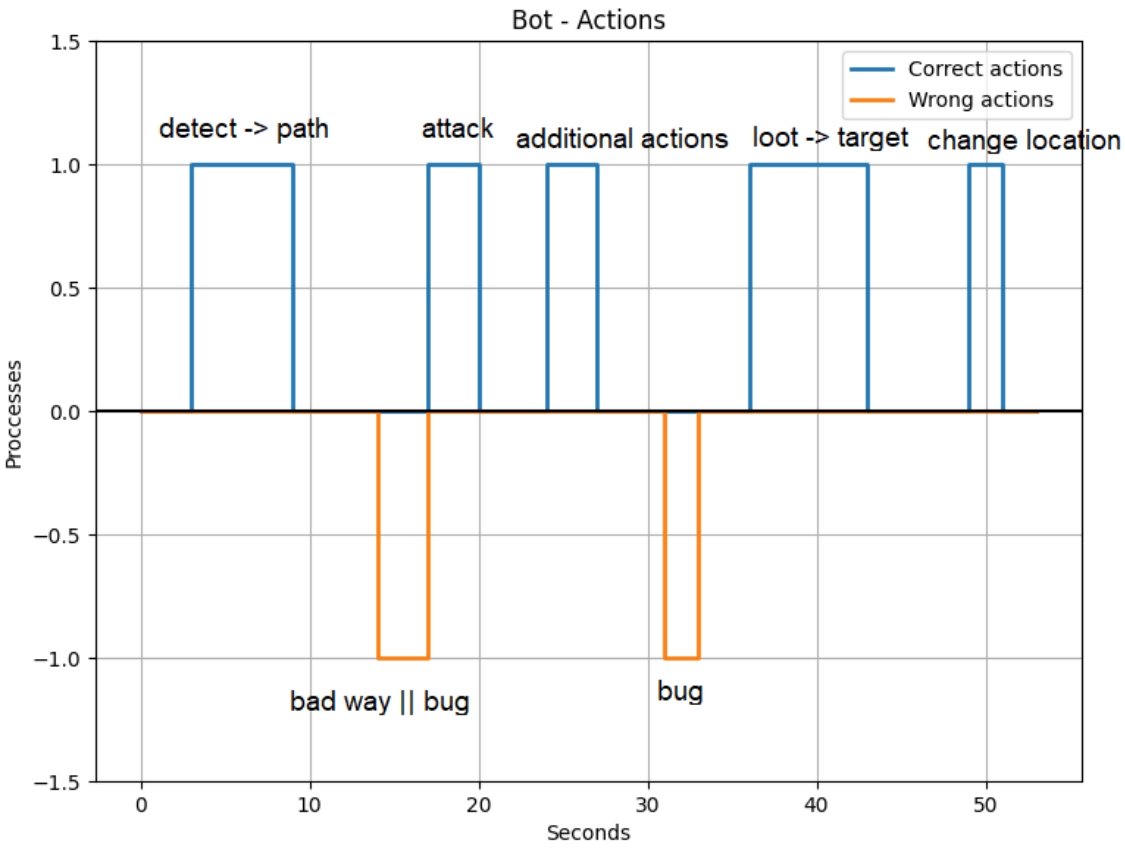



Human - GPU usage
d=1
d=2
d=3
Usages
Seconds
40
35
30
25
20
15
10
0
50
100
150
200
250
300
350


Bot - GPU usage
d=1
d=2
d=3
Usages
Seconds
80
70
60
50
40
30
20
10
0
50
100
150
200
250
300
350


Bot - RAM usage
d=1
d=2
d=3
Usages
Seconds
57.0
56.5
56.0
55.5
55.0
54.5
54.0
0
50
100
150
200
250
300
350


Human - RAM usage
d=1
d=2
d=3
Usages
Seconds
50.8
50.6
50.4
50.2
50.0
49.8
0
50
100
150
200
250
300
350


Bot - Actions
Correct actions
Wrong actions
detect -> path
attack
additional actions
loot -> target
change location
bad way || bug
bug
Proccesses
Seconds
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
10
20
30
40
50