**Файлы и их назначение**

requirements**.**txt**:**

В этом файле перечислены необходимые библиотеки Python и их версии для работы приложения:

* dearpygui**:** Используется для создания графического интерфейса пользователя (GUI).
* psutil**:** Предоставляет информацию о запущенных процессах и использовании системы.
* pywin**32:** Позволяет взаимодействовать с Windows API для управления окнами.

ui**.**py**:**

Этот файл определяет функции для построения элементов интерфейса:

* get\_window\_info**(**target\_hwnd**):** Получает информацию (заголовок, ID процесса, имя процесса) об окне по его дескриптору.
* auto\_center\_cb**(**s**,** d**):** Функция обратного вызова для автоматического центрирования главного окна при изменении размера экрана.
* ui**(**tar\_hwnd**=**None**):** Создает главное окно и отображает информацию о процессе (заголовок, имя, ID) целевого окна.

translator**.**py**:**

Этот файл отвечает за функциональность перевода:

* **Библиотека** argostranslate**:** Используется для оффлайн перевода между языками.
* from\_code **&** to\_code**:** Определяют коды исходного и целевого языков ("en" для английского и "ru" для русского в данном случае).
* **Процесс перевода:** Загружает и устанавливает необходимый пакет перевода, затем переводит заданный текст ("Hello World") и выводит результат.

selection**.**py**:**

Этот файл позволяет пользователю выбрать область экрана для захвата субтитров:

* **Библиотека** tkinter**:** Создает базовое окно с холстом.
* **Библиотека** PIL **(**Pillow**):** Используется для обработки изображений и отображения скриншота.
* **Библиотека** pynput**:** Обнаруживает события клавиатуры, а именно клавишу "Esc" для выхода из процесса выбора.
* **Процесс выбора:**
  + Делает скриншот и отображает его на холсте.
  + Позволяет пользователю щелкнуть и перетащить мышь, чтобы определить прямоугольную область.
  + Сохраняет координаты выбранной области в файл ("data.rect").
  + Запускает основное приложение ("prcl.py") с координатами области в качестве аргументов.

recognition\_on\_img**.**py**:** (Кажется, не связан с основным приложением и потенциально неполный)

Этот файл, похоже, представляет собой отдельный скрипт для распознавания изображений и перевода. Он использует paddleocr для OCR и отображает результаты в окне DearPyGui. Неясно, как он связан с основным потоком приложения.

prcl**.**py**:**

Этот файл перечисляет запущенные процессы и позволяет пользователю выбрать целевое окно для наложения перевода:

* **Библиотека** psutil**:** Получает информацию о процессах.
* **Функция** mwtitle**(**pid**):** Определяет заголовок главного окна процесса по его ID.
* **Функция** get\_mwtitle**(**sender**,** app\_data**,** user\_data**):** Обрабатывает выбор процесса из списка и запускает функцию make\_ovl с информацией о выбранном окне.
* **Функция** make\_ovl**(**t**):** Запускает главный скрипт приложения ("main.py") с заголовком и дескриптором выбранного окна в качестве аргументов.
* **Интерфейс:** Создает окно DearPyGui со списком запущенных процессов и позволяет выбрать целевой процесс.

main**.**py**:**

Этот файл является основным скриптом приложения, который:

* **Обрабатывает аргументы командной строки:**
  + HWND: дескриптор целевого окна.
  + Заголовок окна.
* **Использует** paddleocr **для** OCR**:**
  + Распознает текст с захваченного изображения.
  + Фильтрует результаты с низкой достоверностью.
* **Использует** argostranslate **для перевода:**
  + Переводит распознанный текст с английского на русский.
* **Осуществляет наложение перевода:**
  + Вычисляет расположение перевода над оригинальными субтитрами.
  + Использует DearPyGui для отрисовки текста перевода поверх целевого окна.
* **Функция** get\_screen**():**
  + Выполняет захват экрана, распознавание текста, перевод и отрисовку перевода.
  + Выполняется периодически с помощью threading.Timer.

external\_overlay**.**py**:**

Этот файл содержит класс ExternalOverlay, который:

* **Создает прозрачное окно** DearPyGui **поверх целевого окна.**
* **Отслеживает изменения положения и размера целевого окна и обновляет положение окна наложения.**
* **Запускает функцию** ui**, которая строит интерфейс наложения.**

MAIN**.**PY

import copy

import external\_overlay

import timer

from external\_overlay import ExternalOverlay

import dearpygui.dearpygui as dpg

import random

import string

import win32gui

import sys

import threading

import pyautogui

import paddleocr

import rapidfuzz

import argostranslate.package

import argostranslate.translate

import nltk

**# Загрузка необходимых модулей для обработки естественного языка (**POS tagging)

nltk.download('averaged\_perceptron\_tagger')

**# Путь к файлу с распознанным текстом**

fname = "./recognized.txt"

**# Функция для расчета расстояния Левенштейна (используется для сравнения строк)**

PASTE\_DIST = rapidfuzz.distance.DamerauLevenshtein

**# Глобальная переменная для хранения текста с предыдущего кадра**

PREVIOUS\_FRAME\_TXT = ""

**# Директория для хранения временных файлов**

BIN\_DUMP\_DIR = "./bin\_dumps"

**# Инициализация** OCR-движка PaddleOCR

ocr = paddleocr.PaddleOCR(lang='en',

rec\_model\_dir="\_models/rec/en/en\_PP-OCRv4\_rec\_infer",

det\_model\_dir="\_models/det/en/en\_PP-OCRv4\_det\_infer",

use\_gpu=True,

cls=False, show\_log=False)

**# Глобальные переменные для хранения информации о распознанных строках, индексах, размерах окна и области захвата**

list\_boxes = []

last\_rpl = []

idx = 0

lines = []

WINDOW\_WIDTH = 1000

WINDOW\_HEIGHT = 1000

**# Чтение координат области захвата из файла**

file\_rect = open('./data.rect', 'r')

rp = file\_rect.readlines()

rtopx = int(rp[0])

rtopy = int(rp[1])

rWIDTH = int(rp[2]) - int(rp[0])

rHEIGHT = int(rp[1]) - int(rp[3])

**# Определение области захвата и положения окна**

selection = [[rtopx, rtopy], [rWIDTH, rHEIGHT]]

title\_pos = [[0, 0], [1000, 1000]]

wp = [[0, 0], [500, 300]]

DISPLACE\_X = 0

DISPLACE\_Y = -20

**# Загрузка и установка пакета для перевода с английского на русский**

from\_code = "en"

to\_code = "ru"

argostranslate.package.update\_package\_index()

available\_packages = argostranslate.package.get\_available\_packages()

package\_to\_install = next(filter(lambda x: x.from\_code == from\_code and x.to\_code == to\_code, available\_packages))

argostranslate.package.install\_from\_path(package\_to\_install.download())

**# Функция для изменения расположения строк с учетом смещения и размеров области захвата**

def line\_repos(l):

**# Функция для получения соотношения длин слов в строке**

def get\_word\_ratios(s):

**# Функция для скрытия интерфейса наложения**

def hide\_ui():

**# Функция для отображения интерфейса наложения**

def show\_ui():

**# Функция для очистки интерфейса от отрисованных элементов**

def clear\_ui(a, b):

**# Функция для удаления всех строк и элементов интерфейса**

def subs\_str\_clr():

**# Функция для генерации случайной строки**

def rand\_string():

**# Функция для отрисовки текста с рамкой на интерфейсе наложения**

def outtext(dpg, txt, pos, size):

**# Функция для отрисовки фона для строк (не используется)**

def draw\_back(dpg):

**# Функция для обновления интерфейса с новыми случайными строками**

def repaint(a, b):

**# Функция для вывода информации о событиях**

def \_log(sender, app\_data, user\_data):

**# Функция для создания окна наложения и отрисовки текста**

def ui(tar\_hwnd=None):

**# Функция для получения координат и размеров целевого окна**

def getrect(hwnd):

**# Функция для отображения интерфейса после определенного времени**

def show\_ui\_after(a, b):

**# Функция для сохранения скриншота области захвата и распознавания текста**

def save\_screen():

**# Функция для определения пересечения двух списков**

def intersection(lst1, lst2):

**# Функция для захвата экрана, распознавания текста, перевода и отрисовки**

def get\_screen():

**# Получение дескриптора целевого окна из аргументов командной строки**

HWND = int(sys.argv[2])

getrect(HWND)

**# Создание объекта** ExternalOverlay и запуск приложения

overlay = ExternalOverlay(sys.argv[1], ui)

overlay.start()

PRCL.PY

import subprocess

import psutil

from subprocess import Popen, PIPE

import win32gui

import dearpygui.dearpygui as dpg

**# Список процессов и окон**

processlist = list()

win\_list = []

**# Глобальные переменные для хранения информации о распознанных строках и индексах**

list\_boxes = []

idx = 0

lines = [["sfskv vk njf nds,jfn d,", [0, 0], [500, 30]], ["rewgwrwgs gs grts gh", [0, 40], [500, 70]]]

**# Функция для получения списка окон с их дескрипторами и заголовками**

def callback(hwnd, strings):

if win32gui.IsWindowVisible(hwnd):

window\_title = win32gui.GetWindowText(hwnd)

left, top, right, bottom = win32gui.GetWindowRect(hwnd)

if window\_title and right - left and bottom - top:

strings.append(str(hwnd) + " " + window\_title)

return True

**# Функция для заполнения списка окон**

def enumw(win\_list):

win32gui.EnumWindows(callback, win\_list) **#** populate list

**# Заполнение списка процессов**

for process in psutil.process\_iter():

processlist.append(process)

**# Функция для получения заголовка главного окна процесса по его** ID

def mwtitle(pid):

process = Popen(['powershell', '.\mainwindowtitle.ps1', str(pid)], stdout=PIPE, stderr=PIPE)

stdout, notused = process.communicate()

return stdout.splitlines()[0].decode(encoding="oem")

**# Функция для вывода информации о событиях**

def \_log(sender, app\_data, user\_data):

print(f"sender: {sender}, \t app\_data: {app\_data}, \t user\_data: {user\_data}")

**# Функция для обработки выбора процесса из списка и запуска** make\_ovl

def get\_mwtitle(sender, app\_data, user\_data):

print(app\_data)

make\_ovl(app\_data)

**# Создание контекста и окна** DearPyGui

dpg.create\_context()

dpg.create\_viewport(title="process list", width=500, height=300, decorated=True, always\_on\_top=True)

**# Заполнение списка окон**

enumw(win\_list)

**# Создание окна со списком процессов**

with dpg.window(no\_background=False, no\_title\_bar=False, tag="pwindow", pos=(300, 300), width=500, height=300):

dpg.add\_listbox([p for p in win\_list], tag="prc\_list", label="process to attach", num\_items=12, callback=get\_mwtitle)

**# Регистрация шрифта с поддержкой кириллицы**

with dpg.font\_registry():

with dpg.font("notomono-regular.ttf", 13, default\_font=True, tag="Default font") as f:

dpg.add\_font\_range\_hint(dpg.mvFontRangeHint\_Cyrillic)

**# Регистрация обработчиков событий клавиатуры (не используются)**

with dpg.handler\_registry(show=False, tag="\_\_demo\_keyboard\_handler"):

k\_down = dpg.add\_key\_down\_handler(key=dpg.mvKey\_0, callback=\_log)

k\_release = dpg.add\_key\_release\_handler(key=dpg.mvKey\_0, callback=\_log)

k\_press = dpg.add\_key\_press\_handler(key=dpg.mvKey\_0, callback=\_log)

**# Привязка шрифта**

dpg.bind\_font("Default font")

**# Функция для запуска главного скрипта приложения с информацией о выбранном окне**

def make\_ovl(t):

subprocess.Popen(['python', './main.py', " ".join(t.split(" ")[1:]), t.split(" ")[0]])

**# Настройка и запуск** DearPyGui

dpg.setup\_dearpygui()

dpg.show\_viewport()

dpg.set\_primary\_window("pwindow", True)

dpg.start\_dearpygui()

dpg.destroy\_context()

EXTERNAL\_OVERLAY.PY

from typing import Callable

import dearpygui.dearpygui as dpg

import threading

import time

import win32gui

import win32con

import ctypes

from ctypes import c\_int

**# Импорт необходимых модулей для работы с окнами и стилями**

dwm = ctypes.windll.dwmapi

user32 = ctypes.windll.user32

**# Структура для определения отступов окна**

class MARGINS(ctypes.Structure):

\_fields\_ = [("cxLeftWidth", c\_int), ("cxRightWidth", c\_int), ("cyTopHeight", c\_int), ("cyBottomHeight", c\_int)]

**# Класс для создания внешнего наложения поверх целевого окна**

class ExternalOverlay:

def \_\_init\_\_(self, target\_window: str, ui\_to\_run: Callable, overlay\_name: str = "Overlay"):

**# Заголовок целевого окна**

self.target\_window = target\_window

**# Заголовок окна наложения**

self.overlay\_name = overlay\_name

**# Функция для построения интерфейса наложения**

self.ui\_to\_run = ui\_to\_run

**# Дескрипторы целевого окна и окна наложения**

self.target\_hwnd = None

self.overlay\_hwnd = None

**# Потоки для управления интерфейсом и отслеживания целевого окна**

self.ui\_thread = None

self.hook\_thread = None

**# Функция для запуска наложения**

def start(self, overlay\_delay: float = 0.01):

**# Запуск потока для инициализации интерфейса**

self.ui\_thread = threading.Thread(target=self.\_init\_ui)

self.ui\_thread.start()

time.sleep(0.5) **# Ожидание создания окон**

**# Проверка успешного создания дескрипторов окон**

if (self.target\_hwnd == None or self.overlay\_hwnd == None):

raise Exception("Creating handles failed.")

**# Запуск потока для отслеживания целевого окна**

self.hook\_thread = threading.Thread(target=self.\_hook\_to\_target(overlay\_delay))

self.hook\_thread.start()

**# Функция для получения дескрипторов окон**

def \_set\_handles(self):

**# Поиск целевого окна по заголовку**

self.target\_hwnd = win32gui.FindWindow(None, self.target\_window)

if not self.target\_hwnd:

raise Exception("Target Window not found.")

**# Поиск окна наложения по заголовку**

self.overlay\_hwnd = win32gui.FindWindow(None, self.overlay\_name)

if not self.overlay\_hwnd:

raise Exception("Overlay Window not found.")

**# Функция для установки прозрачности окна наложения**

def \_set\_dpg\_win\_transparent(self):

**# Установка отступов для расширения рамки окна на всю клиентскую область**

margins = MARGINS(-1, -1, -1, -1)

dwm.DwmExtendFrameIntoClientArea(self.overlay\_hwnd, margins)

**# Установка стиля окна для прозрачности и пропуска кликов**

win32gui.SetWindowLong(self.overlay\_hwnd, win32con.GWL\_EXSTYLE, win32gui.GetWindowLong(self.overlay\_hwnd, win32con.GWL\_EXSTYLE) | win32con.WS\_EX\_LAYERED | win32con.WS\_EX\_TRANSPARENT)

**# Функция для отслеживания положения и размера целевого окна**

def \_hook\_to\_target(self, overlay\_delay: float):

**# Получение координат и размеров целевого окна**

tar\_rect = win32gui.GetWindowRect(self.target\_hwnd)

**# Установка положения и размеров окна наложения поверх целевого окна**

win32gui.SetWindowPos(self.overlay\_hwnd, win32con.HWND\_TOPMOST, tar\_rect[0], tar\_rect[1], tar\_rect[2] - tar\_rect[0], tar\_rect[3] - tar\_rect[1], win32con.SWP\_NOZORDER)

while True:

try:

**# Обновление положения и размеров окна наложения при изменении целевого окна**

tar\_rect = win32gui.GetWindowRect(self.target\_hwnd)

win32gui.SetWindowPos(self.overlay\_hwnd, win32con.HWND\_TOPMOST, tar\_rect[0], tar\_rect[1], tar\_rect[2] - tar\_rect[0], tar\_rect[3] - tar\_rect[1], win32con.SWP\_NOZORDER)

time.sleep(overlay\_delay)

except:

**# Закрытие окна наложения и остановка** DearPyGui **при возникновении ошибки**

win32gui.CloseWindow(self.overlay\_hwnd)

dpg.stop\_dearpygui()

**# Функция для инициализации интерфейса наложения**

def \_init\_ui(self):

**# Создание контекста и окна** DearPyGui

dpg.create\_context()

dpg.create\_viewport(title=self.overlay\_name, width=1000, height=1000, decorated=False, always\_on\_top=True, clear\_color=[0.0, 0.0, 0.0, 0.0]),

dpg.setup\_dearpygui()

dpg.show\_viewport()

**# Получение дескрипторов окон**

self.\_set\_handles()

**# Установка прозрачности окна наложения**

self.\_set\_dpg\_win\_transparent()

**# Запуск функции для построения интерфейса наложения**

self.ui\_to\_run(self.target\_hwnd)

**# Запуск** DearPyGui

dpg.start\_dearpygui()

SELECTION**.**PY

import subprocess

import sys

import tkinter as tk

from pynput.keyboard import Key, Listener

import pyautogui

import keyboard

import os

from PIL import Image, ImageTk

from functools import partial

**# Размеры окна для выбора области**

WIDTH, HEIGHT = 1600, 900

**# Координаты начальной и конечной точек прямоугольной области**

topx, topy, botx, boty = 0, 0, 0, 0

**# Идентификатор прямоугольника выбора на холсте**

rect\_id = None

**# Создание скриншота**

pyautogui.screenshot("./straight\_to\_disk.png")

**# Путь к файлу с изображением**

path = "mtr.jpg"

**# Функция для получения координат начальной точки при нажатии левой кнопки мыши**

def get\_mouse\_posn(event):

global topy, topx

topx, topy = event.x, event.y

**# Функция для обновления координат прямоугольника выбора при движении мыши с зажатой левой кнопкой**

def update\_sel\_rect(event):

global rect\_id

global topy, topx, botx, boty

botx, boty = event.x, event.y

canvas.coords(rect\_id, topx, topy, botx, boty) **# Обновление координат прямоугольника**

**# Создание окна** tkinter

window = tk.Tk()

window.title("Select Area")

window.geometry('%sx%s' % (WIDTH, HEIGHT))

window.configure(background='grey')

**# Загрузка скриншота и создание холста**

img = ImageTk.PhotoImage(Image.open("./straight\_to\_disk.png"))

canvas = tk.Canvas(window, width=img.width(), height=img.height(), borderwidth=0, highlightthickness=0)

canvas.pack(expand=True)

canvas.img = img **# Сохранение ссылки на изображение**

canvas.create\_image(0, 0, image=img, anchor=tk.NW)

**# Создание прямоугольника выбора (изначально невидимый)**

rect\_id = canvas.create\_rectangle(topx, topy, topx, topy, dash=(2, 2), fill='', outline='red')

**# Привязка событий мыши к функциям**

canvas.bind('<Button-1>', get\_mouse\_posn)

canvas.bind('<B1-Motion>', update\_sel\_rect)

**# Функция для создания обработчика события нажатия клавиши**

def exit\_on\_key(keyname):

def callback(event):

**# Проверка нажатия клавиши "**Esc"

if event.name == keyname:

**# Расчет координат и размеров выбранной области**

tcX = min(topx, botx)

tcY = min(topy, boty)

rWIDTH = max(topx, botx) - tcX

rHEIGHT = max(topy, boty) - tcY

**# Сохранение координат и размеров в файл**

fname = "./data.rect"

file\_rect = open(fname, 'w')

file\_rect.write('\n'.join([str(tcX), str(tcY), str(rWIDTH), str(rHEIGHT)]))

file\_rect.close()

**# Запуск приложения "**prcl**.**py**" с координатами области в качестве аргументов**

subprocess.Popen(['python', './prcl.py', str(tcX), str(tcY), str(rWIDTH), str(rHEIGHT)])

sys.exit()

return callback

**# Привязка обработчика события нажатия клавиши "**Esc**"**

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

keyboard.hook(exit\_on\_key('esc'))

window.mainloop()

UI.PY

import dearpygui.dearpygui as dpg

import psutil

import win32gui

import win32process

**# Функция для получения информации об окне**

def get\_window\_info(target\_hwnd):

**# Получение заголовка окна**

window\_title = win32gui.GetWindowText(target\_hwnd)

**# Получение** ID процесса, связанного с окном

process\_id = win32process.GetWindowThreadProcessId(target\_hwnd)[1]

**# Поиск имени процесса по** ID

process\_name = None

for proc in psutil.process\_iter(['name', 'pid']):

if proc.info['pid'] == process\_id:

process\_name = proc.info['name']

break

**# Возврат словаря с информацией об окне**

return {

'window\_title': window\_title,

'process\_id': process\_id,

'process\_name': process\_name,

}

**# Функция обратного вызова для автоматического центрирования окна**

def auto\_center\_cb(s, d):

**# Получение ширины области просмотра и ширины окна**

viewport\_w = dpg.get\_viewport\_width()

win\_w = dpg.get\_item\_width("main\_win")

**# Установка позиции окна по центру с небольшим отступом справа**

dpg.set\_item\_pos("main\_win", [(viewport\_w - win\_w) - 10, 80])

**# Функция для создания интерфейса с информацией о процессе**

def ui(tar\_hwnd=None):

**# Установка функции обратного вызова для автоматического центрирования окна**

dpg.set\_viewport\_resize\_callback(auto\_center\_cb)

**# Получение информации о целевом окне**

window\_info = get\_window\_info(tar\_hwnd)

**# Создание главного окна**

with dpg.window(tag="main\_win", no\_background=False, no\_move=False, no\_resize=True, no\_title\_bar=True,

autosize=True):

**# Добавление текста с информацией о процессе**

dpg.add\_text("Process Information:")

dpg.add\_text("Process Window: " + window\_info['window\_title'])

dpg.add\_text("Process Name: " + window\_info['process\_name'])

dpg.add\_text("Process ID: " + str(window\_info['process\_id']))