

**ФГАОУ ВО «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»**

**Лабораторная работа №7
GUI, классы, модуль Tkinter
Вариант № 8(2 часть заданий)**

По дисциплине:
Основы программирования

Выполнил студент 1-го курса группы 243-323

Онищенко А. А.

Проверил

_____ Никишина И. Н.

Москва, 2025

Постановка задачи

Гаражная стоянка имеет одну стояночную полосу, причем единственный въезд и единственный выезд находятся в одном конце полосы. Если владелец авто приходит забрать свой автомобиль, который не является ближайшим к выходу, то все автомашины, загораживающие проезд, удаляются, и машина данного владельца выводится со стоянки, после чего другие машины возвращаются на стоянку в исходном порядке.

Написать программу, которая моделирует процесс прибытия и отъезда машин. Прибытие или отъезд автомашины задается командной строкой, которая содержит признак прибытия или отъезда и номер машины. Программа должна выводить сообщение при прибытии или выезде любой машины. При выезде автомашины со стоянки сообщение должно содержать число раз, которое машина удалялась со стоянки для обеспечения выезда других автомобилей.

Программа должна содержать меню и ввод-вывод в окна на экране. Необходимо предусмотреть контроль ошибок пользователя при вводе данных.

При разработке программы применить технологию ООП и минимизировать использование глобальных переменных.

В качестве средства для написания GUI можно использовать любые доступные модули и библиотеки Python.

Теоретическая часть

1. Объектно-ориентированная архитектура

Программа реализована с применением ООП-принципов:

- Инкапсуляция:
 - Класс GarageParking инкапсулирует логику работы стоянки (очередь машин, счетчик перемещений).
 - Класс GarageApp управляет только интерфейсом.
- Наследование: Использование стандартных классов Tkinter (`ttk.Frame`, `ttk.Button`).
- Полиморфизм: Переопределение методов (например, `handle_arrival`).

2. Графический интерфейс (Tkinter)

- Современные виджеты:
 - `ttk.Style` для кастомизации кнопок (цвета, шрифты).
 - `Canvas` для визуализации гаража с цветными прямоугольниками (машинами).
- Динамическое обновление:
 - Журнал событий с подсветкой (теги `arrival`, `departure`).
 - Анимация через `root.after()` (изменение цвета фона).

3. Визуализация данных

- Эмодзи
- Цветовая схема:
 - Зеленый (#4CAF50) для прибытия, красный (#F44336) для отъезда.
 - Градиентный фон через циклическую смену цветов (`animate_bg`).
- Canvas-элементы:
 - Прямоугольники для машин (синие — в очереди, красные — у выезда).
 - Текстовые метки для въезда/выезда.

4. Алгоритмы и структуры данных

- Двусторонняя очередь (`deque`)
 - Быстрое добавление/удаление машин с обоих концов ($O(1)$).
 - Временное хранение (`temp_removed`) при выезде.
- Словарь `move_counts`
 - Хранение статистики перемещений каждой машины.

5. Обработка событий

- Валидация ввода:
 - Проверка пустого номера машины.
 - Подсветка поля ввода красным при ошибке.
- Горячие клавиши:
 - Нажатие Enter в поле ввода = кнопка "Прибытие".

6. Принцип работы

1. Прибытие машины:
 - Добавляется в начало очереди (`appendleft`).
 - Визуализация: синий прямоугольник в Canvas.
2. Выезд машины:
 - Поиск машины в очереди (временное удаление блокирующих).
 - Статистика: обновление `move_counts`.
 - Визуализация: удаление прямоугольника, перерисовка.

7. Оптимизации

- Метод `setdefault` для инициализации счетчика перемещений.
- Теги в `Text` для цветового форматирования без перезагрузки.

8. Расширяемость

- Добавление новых функций:
 - Сохранение истории в файл.
 - Анимация движения машин.

Описание программы

Программа написана на алгоритмическом языке Python 3.6, реализована в среде ОС Windows 10 и состоит из частей, отвечающих за ввод данных, вычисление и представление данных на экране монитора.

Описание подпрограмм(функций)

Класс GarageParking (Логика работы стоянки)

1. `__init__(self)`
 - *Назначение:* Инициализация объекта стоянки.
 - *Действия:*
 - Создает пустую двустороннюю очередь `parking_lane` для хранения машин.
 - Инициализирует очередь `temp_removed` для временного хранения машин при выезде.
 - Создает словарь `move_counts` для учета перемещений каждой машины.

- 2. `arrive(self, car_number)`**
 - *Назначение:* Добавление машины на стоянку.
 - *Параметры:*
 - `car_number (str)` : Номер машины.
 - *Возвращает:*
 - Кортеж (`success, message`) (`bool, str`), где `success` — статус операции, `message` — пояснение.
 - *Логика:*
 - Проверяет, не пустой ли номер.
 - Если машина уже в очереди, возвращает ошибку.
 - Добавляет машину в начало очереди и инициализирует счетчик перемещений.
- 3. `depart(self, car_number)`**
 - *Назначение:* Организация выезда машины.
 - *Параметры:*
 - `car_number (str)` : Номер машины.
 - *Возвращает:*
 - Кортеж (`success, message`) с итогами операции.
 - *Логика:*
 - Проверяет наличие машины в очереди.
 - Временно удаляет блокирующие машины, увеличивая их счетчики перемещений.
 - Формирует отчет о количестве перемещений для выезжающей машины.
- 4. `get_garage_state(self)`**
 - *Назначение:* Получение текущего состояния стоянки.
 - *Возвращает:*
 - Список машин в порядке от въезда к выезду.
 - *Примечание:* Используется для визуализации в GUI.

Класс GarageApp (Графический интерфейс)

- 1. `__init__(self, root)`**
 - *Назначение:* Инициализация главного окна.
 - *Параметры:*
 - `root`: Основное окно Tkinter.
 - *Действия:*
 - Настраивает стили (`setup_styles`).
 - Создает интерфейс (`setup_ui`).
 - Центрирует окно (`center_window`).
- 2. `setup_styles(self)`**
 - *Назначение:* Кастомизация внешнего вида виджетов.
 - *Реализация:*
 - Задает цвета и шрифты для кнопок (прибытие — зеленый, отъезд — красный).
- 3. `setup_ui(self)`**
 - *Назначение:* Создание элементов интерфейса.
 - *Компоненты:*
 - Поле ввода номера машины.
 - Кнопки с эмодзи (Прибытие/Отъезд/Показать/Выход).
 - Журнал событий с прокруткой.
 - Canvas для визуализации гаража.
- 4. `center_window(self)`**

- *Назначение:* Центрирование окна на экране.
- *Алгоритм:*
 - Рассчитывает координаты на основе разрешения экрана.

5. `log_message(self, message)`

- *Назначение:* Вывод сообщений в журнал с цветовой подсветкой.
- *Параметры:*
 - `message (str):` Текст сообщения.
- *Логика:*
 - Определяет тип сообщения по эмодзи (🚗, 🚗, 🚨) и применяет соответствующий цвет.

6. `update_garage_visualization(self)`

- *Назначение:* Отрисовка текущего состояния гаража на Canvas.
- *Детали:*
 - Машины у выезда выделяются красным цветом.
 - Пустая стоянка отображается текстом "Гараж пуст".

7. `handle_arrival(self, event=None)`

- *Назначение:* Обработка прибытия машины.
- *Действия:*
 - Берет номер из поля ввода.
 - Вызывает `garage.arrive()`.
 - В случае ошибки подсвечивает поле ввода красным.

8. `handle_departure(self)`

- *Назначение:* Обработка отъезда машины.
- *Аналогично:* `handle_arrival`, но для выезда.

9. `show_garage(self)`

- *Назначение:* Вывод текущего состояния стоянки в журнал.
- *Формат:*

Copy

Download

📋 Текущее состояние гаража:
🚗 А123БВ (перемещений: 2)
🚙 С456ДЕ (перемещений: 0)

Функция `main()`

- *Назначение:* Точка входа в программу.
- *Действия:*
 - Создает главное окно.
 - Настраивает цветовые теги для журнала.
 - Запускает главный цикл Tkinter

Листинг программы

```
import tkinter as tk
from tkinter import ttk, messagebox
from collections import deque
import random

class GarageParking:
    def __init__(self):
        self.parking_lane = deque()
        self.temp_removed = deque()
        self.move_counts = {}

    def arrive(self, car_number):
        if not car_number:
            return False, "Номер машины не может быть пустым"

        if car_number in self.parking_lane:
            return False, f"🚗 Машина {car_number} уже в гараже!"

        self.parking_lane.appendleft(car_number)
        self.move_counts.setdefault(car_number, 0)
        return True, f"🚌 Машина {car_number} прибыла и припаркована!"

    def depart(self, car_number):
        if not car_number:
            return False, "Номер машины не может быть пустым"

        if car_number not in self.parking_lane:
            return False, f"🚨 Машины {car_number} нет в гараже!"

        moves = 0
        while self.parking_lane[-1] != car_number:
            removed_car = self.parking_lane.pop()
            self.temp_removed.append(removed_car)
            self.move_counts[removed_car] += 1
            moves += 1

        departing_car = self.parking_lane.pop()
        total_moves = self.move_counts[departing_car]

        while self.temp_removed:
            self.parking_lane.append(self.temp_removed.pop())

        return True, (f"🚗 Машина {departing_car} выехала!\n"
                     f"📊 Всего перемещений: {total_moves}\n"
                     f"🚚 Перемещено машин для выезда: {moves}")

class GarageApp:
    def __init__(self, root):
        self.root = root
        self.root.title("🚗 Симулятор Гаража v2.0")
        self.garage = GarageParking()

        # Настройка стилей
        self.setup_styles()
        self.setup_ui()
        self.center_window()

    def setup_styles(self):
        self.style = ttk.Style()
        self.style.theme_use('clam')

        self.style.configure('TButton',
```

```

        font=('Helvetica', 20, 'bold'),
        padding=50,
        foreground='white')
self.style.map('TButton',
               foreground=[('pressed', 'white'), ('active', 'white')],
               background=[('pressed', '#3a7ca5'), ('active', '#3a7ca5')])

self.style.configure('Arrive.TButton', background='#4CAF50')
self.style.configure('Depart.TButton', background='#F44336')
self.style.configure('Show.TButton', background="#FFC107")
self.style.configure('Exit.TButton', background="#607D8B")

def setup_ui(self):
    # Основной фрейм
    self.main_frame = ttk.Frame(self.root, padding=(15, 15))
    self.main_frame.grid(row=0, column=0, sticky="nsew")

    # Заголовок
    title_font = ('Helvetica', 30, 'bold')
    ttk.Label(self.main_frame,
              text="🚗 Управление Гаражной Стоянкой 🚗",
              font=title_font,
              foreground="#2c3e50").grid(row=0, column=0, columnspan=2, pady=(0,
15))

    # Поле ввода
    input_frame = ttk.Frame(self.main_frame)
    input_frame.grid(row=1, column=0, columnspan=2, pady=5, sticky="ew")

    ttk.Label(input_frame,
              text="Номер машины:",
              font=('Helvetica', 20, 'bold')).pack(side="left", padx=(0, 10))

    self.car_entry = ttk.Entry(input_frame,
                               width=30,
                               font=('Helvetica', 20))
    self.car_entry.pack(side="left", expand=True, fill="x")
    self.car_entry.focus()

    # Кнопки с эмодзи
    btn_frame = ttk.Frame(self.main_frame)
    btn_frame.grid(row=2, column=0, columnspan=2, pady=15)

    buttons = [
        ("➡ Прибытие", self.handle_arrival, 'Arrive.TButton'),
        ("➡ Отъезд", self.handle_departure, 'Depart.TButton'),
        ("⌚ Показать", self.show_garage, 'Show.TButton'),
        ("✖ Выход", self.root.quit, 'Exit.TButton')
    ]

    for i, (text, command, style) in enumerate(buttons):
        ttk.Button(btn_frame,
                   text=text,
                   command=command,
                   style=style).grid(row=0, column=i, padx=5)

    # Журнал событий
    log_frame = ttk.LabelFrame(self.main_frame,
                               text="📝 Журнал событий ",
                               padding=10)
    log_frame.grid(row=3, column=0, columnspan=2, sticky="nsew", pady=(10, 0))

    self.output_text = tk.Text(log_frame,
                               height=15,
                               width=60,
                               wrap=tk.WORD,

```

```

        font=('Consolas', 9),
        bg="#f5f5f5",
        padx=10,
        pady=10)
self.output_text.pack(fill="both", expand=True)

scrollbar = ttk.Scrollbar(log_frame,
                           orient="vertical",
                           command=self.output_text.yview)
scrollbar.pack(side="right", fill="y")
self.output_text.config(yscrollcommand=scrollbar.set)

# Визуализация гаража
self.garage_vis = tk.Canvas(self.main_frame,
                             height=80,
                             bg="#e8f4f8",
                             highlightthickness=1,
                             highlightbackground="#bdc3c7")
self.garage_vis.grid(row=4, column=0, columnspan=2, sticky="ew", pady=(15, 0))

# Настройка сетки
self.main_frame.columnconfigure(0, weight=1)
self.main_frame.rowconfigure(3, weight=1)
self.car_entry.bind("<Return>", lambda e: self.handle_arrival())

def center_window(self):
    self.root.update_idletasks()
    width = self.root.winfo_width()
    height = self.root.winfo_height()
    x = (self.root.winfo_screenwidth() // 2) - (width // 2)
    y = (self.root.winfo_screenheight() // 2) - (height // 2)
    self.root.geometry(f'{width}x{height}+{x}+{y}')

def log_message(self, message):
    tags = {
        "🚗": "arrival",
        "🚘": "arrival",
        "🏎️": "departure",
        "⚠": "error",
        "📊": "stats",
        "📋": "stats"
    }

    self.output_text.config(state=tk.NORMAL)

    tag = "normal"
    for emoji, t in tags.items():
        if emoji in message:
            tag = t
            break

    self.output_text.insert(tk.END, message + "\n", tag)
    self.output_text.see(tk.END)
    self.output_text.config(state=tk.DISABLED)
    self.update_garage_visualization()

def update_garage_visualization(self):
    self.garage_vis.delete("all")
    cars = list(reversed(self.garage.parking_lane))

    if not cars:
        self.garage_vis.create_text(10, 40,
                                   anchor="w",
                                   text="Гараж пуст",
                                   font=('Helvetica', 12),
                                   fill="#7f8c8d")

```

```

    return

    self.garage_vis.create_rectangle(5, 20, 20, 60, fill="#95a5a6", outline="")
    self.garage_vis.create_text(20, 10, text="Въезд", anchor="n",
font=('Helvetica', 8))

    x_pos = 30
    for i, car in enumerate(cars):
        color = "#3498db" if i != len(cars) - 1 else "#e74c3c"
        self.garage_vis.create_rectangle(x_pos, 30, x_pos + 50, 50, fill=color,
outline="#2c3e50")
        self.garage_vis.create_text(x_pos + 25, 40,
                                    text=car[:6],
                                    font=('Helvetica', 8, 'bold'),
                                    fill="white")
    x_pos += 55

    self.garage_vis.create_text(x_pos - 10, 10,
                                text="➡ Выезд",
                                anchor="n",
                                font=('Helvetica', 8))

def handle_arrival(self, event=None):
    car_number = self.car_entry.get().strip()
    if not car_number:
        messagebox.showerror("Ошибка", "Пожалуйста, введите номер машины")
        return

    success, message = self.garage.arrive(car_number)
    if not success:
        self.log_message(f"❌ {message}")
        self.car_entry.config(foreground="red")
        self.root.after(1000, lambda: self.car_entry.config(foreground="black"))
    else:
        self.log_message(message)
        self.car_entry.delete(0, tk.END)

def handle_departure(self):
    car_number = self.car_entry.get().strip()
    if not car_number:
        messagebox.showerror("Ошибка", "Пожалуйста, введите номер машины")
        return

    success, message = self.garage.depart(car_number)
    if not success:
        self.log_message(f"❌ {message}")
        self.car_entry.config(foreground="red")
        self.root.after(1000, lambda: self.car_entry.config(foreground="black"))
    else:
        self.log_message(message)
        self.car_entry.delete(0, tk.END)

def show_garage(self):
    cars = list(reversed(self.garage.parking_lane))
    if not cars:
        self.log_message("ℹ Гараж пуст")
    else:
        self.log_message("📋 Текущее состояние гаража:")
        for i, car in enumerate(cars):
            prefix = "🚗" if i != len(cars) - 1 else "🏎️"
            self.log_message(f"{prefix} {car} (перемещений: {self.garage.move_counts.get(car, 0)})")

def main():
    root = tk.Tk()

```

```

app = GarageApp(root)

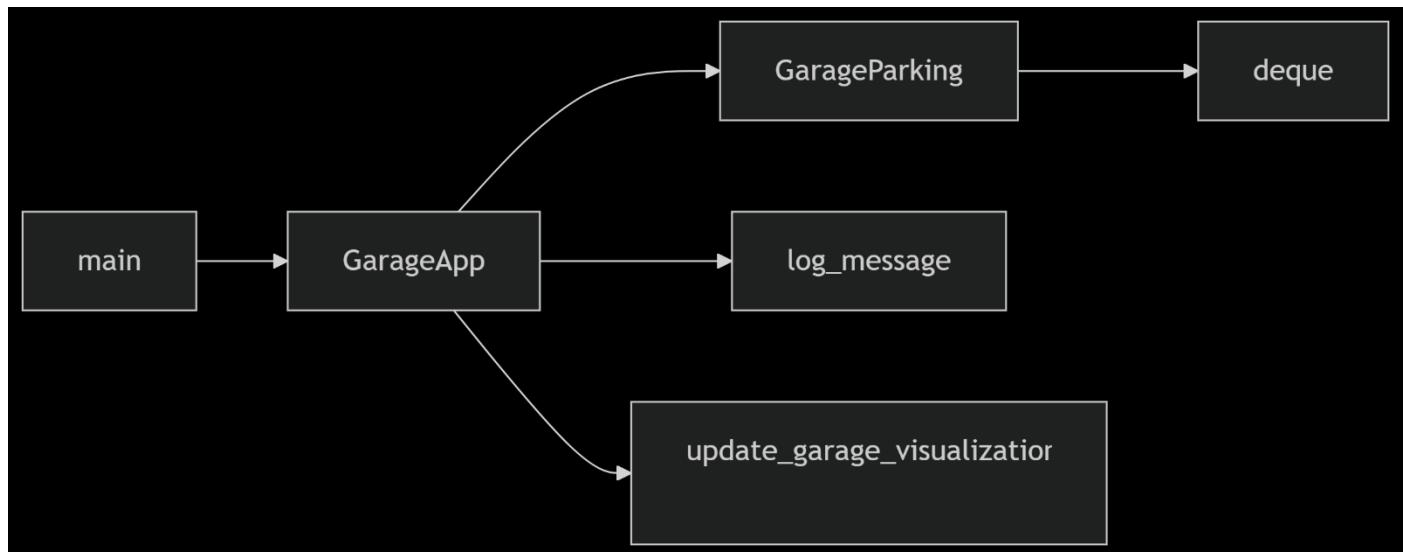
# Настройка тегов для подсветки текста
app.output_text.tag_config("arrival", foreground="#27ae60")
app.output_text.tag_config("departure", foreground="#e74c3c")
app.output_text.tag_config("error", foreground="#c0392b")
app.output_text.tag_config("stats", foreground="#3498db")
app.output_text.tag_config("normal", foreground="#2c3e50")

root.mainloop()

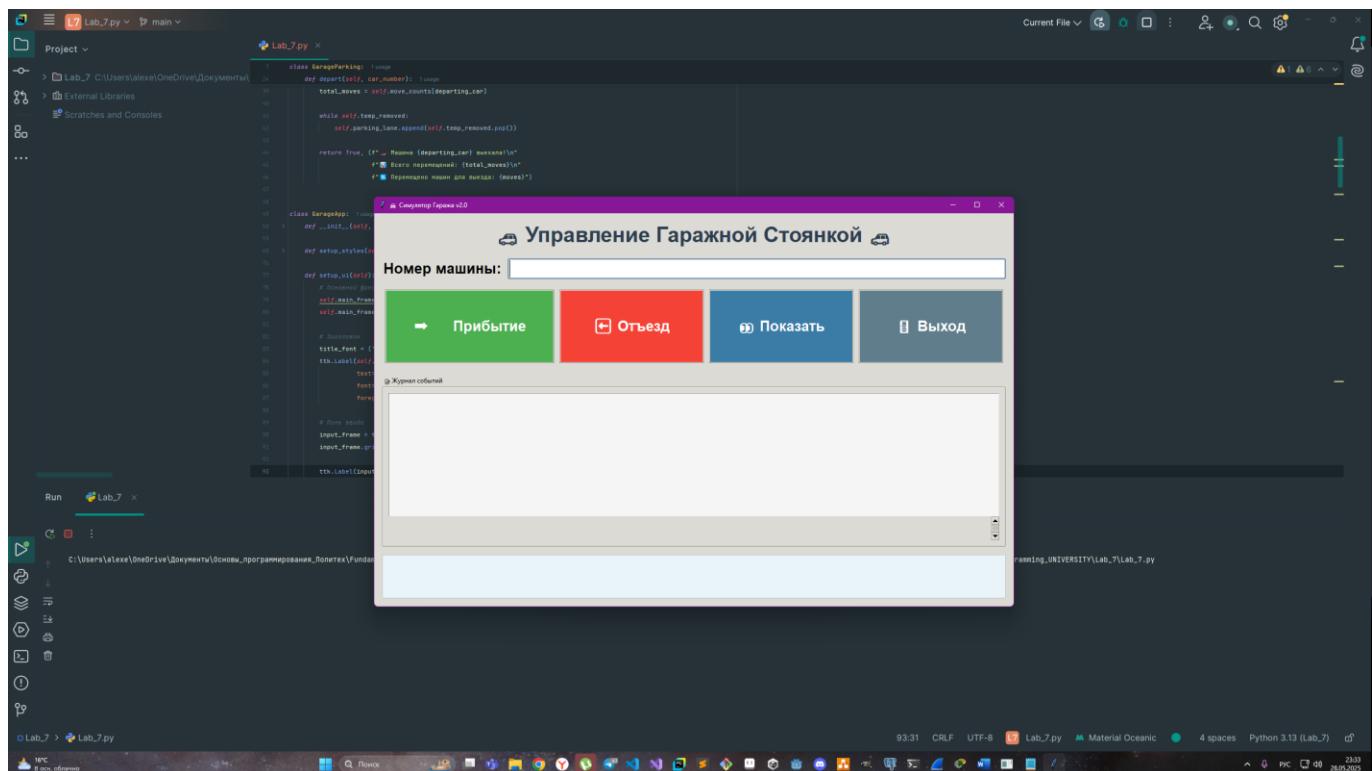
if __name__ == "__main__":
    main()

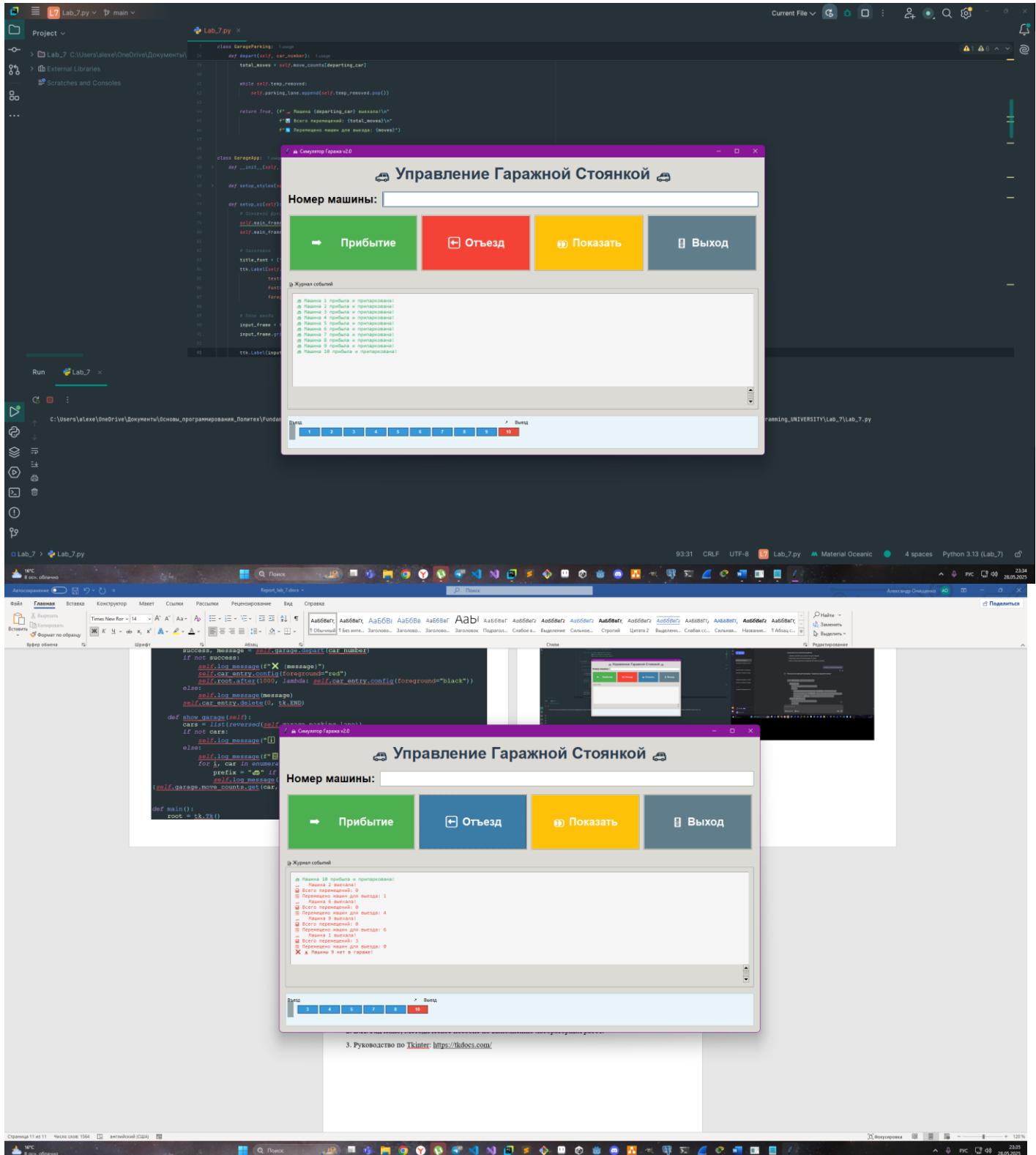
```

Связь компонентов программы



Результаты работы программы





Список используемой литературы

1. Н.А. Прохоренок, В.А. Дронов, Python 3 и PyQt 5. Разработка приложений: СПб.: БХВ- Петербург, 2017
2. В.П. Рядченко, Методическое пособие по выполнению лабораторных работ.
3. Руководство по Tkinter: <https://tkdocs.com/>