МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “ Парадигмы и конструкции языков программирования”

Отчет по домашнему заданию

Выполнила:
Студент группы ИУ5-34Б
Глозман В.А.
Преподаватель:
Гапанюк Ю. Е.

Москва 2025

Задание:

1. Выберите язык программирования (который Вы ранее не изучали) и (1) напишите по нему реферат с примерами кода или (2) реализуйте на нем небольшой проект (с детальным текстовым описанием).
2. Реферат (проект) может быть посвящен отдельному аспекту (аспектам) языка или содержать решение какой-либо задачи на этом языке.
3. Необходимо установить на свой компьютер компилятор (интерпретатор, транспилятор) этого языка и произвольную среду разработки.
4. В случае написания реферата необходимо разработать и откомпилировать примеры кода (или модифицировать стандартные примеры).
5. В случае создания проекта необходимо детально комментировать код.
6. При написании реферата (создании проекта) необходимо изучить и корректно использовать особенности парадигмы языка и основных конструкций данного языка.
7. Приветствуется написание черновика статьи по результатам выполнения ДЗ. Черновик статьи может быть подготовлен группой студентов, которые исследовали один и тот же аспект в нескольких языках или решили одинаковую задачу на нескольких языках.

Описание программы

Целью работы является освоение языка программирования Python и библиотеки Tkinter для создания графических пользовательских интерфейсов. В рамках задания реализован проект «Тренажер таблицы умножения» — интерактивное приложение, предназначенное для изучения и закрепления навыков умножения через игровой процесс.

multiplication_game.py

```
"""
Логика игры для тренажера таблицы умножения
Данный модуль содержит класс MultiplicationGame, который управляет всей
логикой игры:
- генерация вопросов на умножение
- проверка ответов пользователя
- подсчет статистики и очков
- управление временем игры
"""
import random  # Для генерации случайных чисел
from datetime import datetime  # Для отслеживания времени начала игры


class MultiplicationGame:
```

```
    """
    Основной класс игры. Отвечает за:
    - хранение текущего состояния игры (очки, статистика)
    - генерацию новых вопросов в зависимости от уровня сложности
    - проверку правильности ответов пользователя
    - подсчет времени игры
    """

    def __init__(self):
        """
        Инициализация нового экземпляра игры.
        Устанавливает начальные значения всех переменных.
        """
        self.score = 0  # Текущее количество очков
        self.correct_answers = 0  # Количество правильных ответов
        self.wrong_answers = 0  # Количество неправильных ответов
        self.current_question = None  # Текущий активный вопрос (словарь с
данными)
        self.start_time = None  # Время начала текущей игры
        self.questions_answered = 0  # Общее количество отвеченных вопросов

        # Настройки уровней сложности.
        # Каждый уровень определяет диапазон чисел и лимит времени (хотя
время не используется активно)
        self.levels = {
            "легкий": {"min": 1, "max": 5, "time_limit": 30},      # Простые
примеры: числа 1-5
            "средний": {"min": 1, "max": 10, "time_limit": 25},    # Средние
примеры: числа 1-10
            "сложный": {"min": 5, "max": 15, "time_limit": 20}     # Сложные
примеры: числа 5-15
        }

        self.current_level = "средний"  # Текущий уровень сложности по
умолчанию

    def start_new_game(self, level="средний"):
        """
        Начинает новую игру с заданным уровнем сложности.
        Сбрасывает всю статистику и генерирует первый вопрос.

        Аргументы:
            level (str): уровень сложности ("легкий", "средний", "сложный")
        """
        # Сброс всех показателей игры к начальным значениям
        self.score = 0
        self.correct_answers = 0
        self.wrong_answers = 0
        self.questions_answered = 0

        self.current_level = level  # Установка выбранного уровня сложности
        self.start_time = datetime.now()  # Запись времени начала игры
        self.generate_question()  # Генерация первого вопроса

    def generate_question(self):
        """
        Генерирует новый вопрос на умножение на основе текущего уровня
сложности.
        Возвращает словарь с данными вопроса.
```

```
        Возвращает:
            dict: словарь с полями:
                - "text": текст вопроса в формате "a × b = ?"
                - "answer": правильный ответ (a * b)
                - "a": первое число
                - "b": второе число
        """
        # Получение настроек для текущего уровня сложности
        level_settings = self.levels[self.current_level]

        # Генерация двух случайных чисел в заданном диапазоне
        a = random.randint(level_settings["min"], level_settings["max"])
        b = random.randint(level_settings["min"], level_settings["max"])

        # Создание объекта вопроса с текстом и правильным ответом
        self.current_question = {
            "text": f"{a} × {b} = ?",  # Текст вопроса для отображения
            "answer": a * b,            # Правильный ответ
            "a": a,                     # Первое число (для подсказок)
            "b": b                      # Второе число (для подсказок)
        }

        return self.current_question  # Возврат созданного вопроса

    def check_answer(self, user_answer):
        """
        Проверяет ответ пользователя на текущий вопрос.
        Обновляет статистику и начисляет очки.

        Аргументы:
            user_answer (str или int): ответ, введенный пользователем

        Возвращает:
            tuple: (is_correct, points) где:
                - is_correct (bool): True если ответ правильный, иначе False
                - points (int): количество начисленных очков
        """
        # Преобразование ответа пользователя в целое число
        try:
            user_answer = int(user_answer)
        except ValueError:
            # Если преобразование не удалось (пользователь ввел не число)
            return False, 0  # Ответ неправильный, 0 очков

        # Проверка совпадения ответа пользователя с правильным ответом
        is_correct = user_answer == self.current_question["answer"]
        self.questions_answered += 1  # Увеличение счетчика отвеченных
вопросов

        if is_correct:
            # Если ответ правильный
            self.correct_answers += 1  # Увеличение счетчика правильных
ответов

            # Начисление очков в зависимости от уровня сложности
            if self.current_level == "легкий":
                points = 5      # 5 очков за правильный ответ на легком
уровне
```

```
            elif self.current_level == "средний":
                points = 10     # 10 очков за правильный ответ на среднем
уровне
            else:  # сложный
                points = 15     # 15 очков за правильный ответ на сложном
уровне

            self.score += points  # Добавление очков к общему счету
        else:
            # Если ответ неправильный
            self.wrong_answers += 1  # Увеличение счетчика неправильных
ответов
            points = 0  # За неправильный ответ очки не начисляются

        # Возврат результата проверки и количества начисленных очков
        return is_correct, points

    def get_stats(self):
        """
        Собирает и возвращает текущую статистику игры.

        Возвращает:
            dict: словарь с текущей статистикой:
                - "score": общее количество очков
                - "correct": количество правильных ответов
                - "wrong": количество неправильных ответов
                - "accuracy": процент правильных ответов (округляется до 1
знака)
                - "total_questions": общее количество отвеченных вопросов
        """
        # Расчет точности (процента правильных ответов)
        accuracy = 0
        if self.questions_answered > 0:
            # Точность = (правильные ответы / все ответы) * 100%
            accuracy = (self.correct_answers / self.questions_answered) * 100

        # Возврат статистики в виде словаря
        return {
            "score": self.score,
            "correct": self.correct_answers,
            "wrong": self.wrong_answers,
            "accuracy": round(accuracy, 1),  # Округление до одного знака
после запятой
            "total_questions": self.questions_answered
        }

    def get_time_elapsed(self):
        """
        Вычисляет, сколько секунд прошло с начала текущей игры.

        Возвращает:
            int: количество секунд, прошедших с начала игры
            Если игра не начата, возвращает 0
        """
        if self.start_time:
            # Вычисление разницы между текущим временем и временем начала
игры
            elapsed = datetime.now() - self.start_time
            return int(elapsed.total_seconds())  # Преобразование в целые
```

```
секунды
        return 0  # Игра еще не начата
```

multiplication_gui.py

```
"""
Графический интерфейс для тренажера таблицы умножения
Данный модуль реализует графический интерфейс приложения с использованием
библиотеки tkinter.
Он предоставляет пользователю интерактивный интерфейс для взаимодействия с
игровой логикой,
отображения вопросов, ввода ответов и просмотра статистики.
"""

import tkinter as tk  # Основная библиотека для создания GUI
from tkinter import ttk, messagebox, font  # Модули для виджетов, диалоговых
окон и шрифтов
from multiplication_game import MultiplicationGame  # Импорт игровой логики


class MultiplicationTrainerGUI:
    """
    Класс графического интерфейса приложения.
    Создает окно приложения, организует виджеты и управляет взаимодействием
    между пользователем и игровой логикой.
    """

    def __init__(self, root):
        """
        Инициализация графического интерфейса.

        Аргументы:
            root (tk.Tk): корневое окно приложения
        """
        self.root = root  # Сохранение ссылки на главное окно
        self.root.title("Тренажер таблицы умножения")  # Заголовок окна
        self.root.geometry("600x550")  # Установка размеров окна (ширина x
высота)
        self.root.configure(bg='#f0f0f0')  # Установка фонового цвета окна

        # Создание экземпляра игры - связь между GUI и логикой игры
        self.game = MultiplicationGame()

        # Настройка шрифтов для различных элементов интерфейса
        self.title_font = font.Font(family="Arial", size=16, weight="bold")
        self.question_font = font.Font(family="Arial", size=24,
weight="bold")
        self.stats_font = font.Font(family="Arial", size=12)

        # Переменные для хранения данных, связанных с tkinter
        self.user_answer = tk.StringVar()  # Переменная для хранения ответа
пользователя
        self.time_var = tk.StringVar(value="Время: 0 сек")  # Переменная для
отображения времени

        # Создание всех элементов интерфейса
        self.create_widgets()
```

```
        # Запуск таймера для обновления времени игры
        self.update_timer()

    def create_widgets(self):
        """
        Создает и размещает все элементы графического интерфейса в окне.
        Метод организует интерфейс в виде иерархии фреймов и виджетов.
        """
        # Основной фрейм - контейнер для всех остальных элементов
        main_frame = ttk.Frame(self.root, padding="20")
        # Размещение основного фрейма в окне с растягиванием во всех
направлениях
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # 1. ЗАГОЛОВОК ПРИЛОЖЕНИЯ
        title_label = ttk.Label(
            main_frame,
            text="ТРЕНАЖЕР ТАБЛИЦЫ УМНОЖЕНИЯ",  # Текст заголовка
            font=self.title_font,  # Шрифт заголовка
            foreground="darkblue"  # Цвет текста
        )
        # Размещение заголовка в первой строке, занимающего 3 колонки
        title_label.grid(row=0, column=0, columnspan=3, pady=(0, 20))

        # 2. ФРЕЙМ С УПРАВЛЕНИЕМ И ВРЕМЕНЕМ
        info_frame = ttk.Frame(main_frame)  # Фрейм для элементов управления
        info_frame.grid(row=1, column=0, columnspan=3, pady=(0, 20))

        # Метка для выбора уровня сложности
        level_label = ttk.Label(info_frame, text="Уровень:")
        level_label.grid(row=0, column=0, padx=(0, 10))

        # Выпадающий список (Combobox) для выбора уровня сложности
        self.level_combo = ttk.Combobox(
            info_frame,
            values=["легкий", "средний", "сложный"],  # Доступные уровни
            state="readonly",  # Только для чтения (пользователь не может
вводить свой текст)
            width=15  # Ширина виджета
        )
        self.level_combo.set("средний")  # Установка значения по умолчанию
        self.level_combo.grid(row=0, column=1, padx=(0, 20))

        # Кнопка для начала новой игры
        new_game_btn = ttk.Button(
            info_frame,
            text="Новая игра",  # Текст на кнопке
            command=self.start_new_game  # Функция, вызываемая при нажатии
        )
        new_game_btn.grid(row=0, column=2, padx=(20, 0))

        # Метка для отображения времени игры
        self.time_label = ttk.Label(
            info_frame,
            textvariable=self.time_var,  # Привязка к переменной времени
            font=self.stats_font
        )
        self.time_label.grid(row=0, column=3, padx=(20, 0))
```

```
        # 3. ФРЕЙМ С ВОПРОСОМ
        question_frame = ttk.LabelFrame(main_frame, text="Вопрос",
padding="20")
        question_frame.grid(row=2, column=0, columnspan=3, pady=(0, 20),
sticky=(tk.W, tk.E))

        # Метка для отображения текущего вопроса
        self.question_label = ttk.Label(
            question_frame,
            text="Нажмите 'Новая игра' для начала",  # Текст по умолчанию
            font=self.question_font,
            foreground="darkgreen",
            anchor="center"  # Выравнивание текста по центру
        )
        self.question_label.pack(expand=True, fill='both')  # Размещение с
растягиванием

        # 4. ФРЕЙМ ДЛЯ ВВОДА ОТВЕТА
        answer_frame = ttk.Frame(main_frame)
        answer_frame.grid(row=3, column=0, columnspan=3, pady=(0, 20))

        # Метка для поля ввода ответа
        answer_label = ttk.Label(answer_frame, text="Ваш ответ:")
        answer_label.grid(row=0, column=0, padx=(0, 10))

        # Поле для ввода ответа пользователем
        self.answer_entry = ttk.Entry(
            answer_frame,
            textvariable=self.user_answer,  # Привязка к переменной ответа
            width=20,  # Ширина поля
            font=("Arial", 14)  # Шрифт для текста в поле
        )
        self.answer_entry.grid(row=0, column=1, padx=(0, 10))
        # Привязка клавиши Enter к функции проверки ответа
        self.answer_entry.bind('<Return>', lambda event: self.check_answer())

        # Кнопка для проверки ответа
        check_btn = ttk.Button(
            answer_frame,
            text="Проверить",
            command=self.check_answer
        )
        check_btn.grid(row=0, column=2, padx=(10, 0))

        # Кнопка для показа подсказки
        hint_btn = ttk.Button(
            answer_frame,
            text="Подсказка",
            command=self.show_hint
        )
        hint_btn.grid(row=0, column=3, padx=(10, 0))

        # 5. ФРЕЙМ СО СТАТИСТИКОЙ ТЕКУЩЕЙ ИГРЫ
        stats_frame = ttk.LabelFrame(main_frame, text="Текущая статистика",
padding="15")
        stats_frame.grid(row=4, column=0, columnspan=3, pady=(0, 20),
sticky=(tk.W, tk.E))
```

```
        # Словарь для хранения ссылок на метки статистики
        self.stats_labels = {}
        # Данные для статистики: (текст метки, ключ в статистике)
        stats_data = [
            ("Очки:", "score"),
            ("Правильно:", "correct"),
            ("Неправильно:", "wrong"),
            ("Точность:", "accuracy"),
            ("Всего вопросов:", "total_questions")
        ]

        # Создание меток статистики в две колонки
        for i, (label_text, key) in enumerate(stats_data):
            row = i // 2  # Определение номера строки (целочисленное деление)
            col = (i % 2) * 2  # Определение номера столбца

            # Метка с названием статистического показателя
            name_label = ttk.Label(stats_frame, text=label_text,
font=self.stats_font)
            name_label.grid(row=row, column=col, padx=(0, 5), pady=2,
sticky=tk.W)

            # Метка с текущим значением статистического показателя
            value_label = ttk.Label(stats_frame, text="0",
font=self.stats_font)
            value_label.grid(row=row, column=col+1, padx=(5, 20), pady=2,
sticky=tk.W)
            self.stats_labels[key] = value_label  # Сохранение ссылки для
обновления

        # 6. ФРЕЙМ С ИНФОРМАЦИЕЙ ОБ УРОВНЯХ СЛОЖНОСТИ
        levels_frame = ttk.LabelFrame(main_frame, text="Уровни сложности",
padding="15")
        levels_frame.grid(row=5, column=0, columnspan=3, pady=(0, 10),
sticky=(tk.W, tk.E))

        # Информация о каждом уровне сложности
        levels_info = [
            ("Легкий:", "Числа от 1 до 5, 5 очков за ответ"),
            ("Средний:", "Числа от 1 до 10, 10 очков за ответ"),
            ("Сложный:", "Числа от 5 до 15, 15 очков за ответ")
        ]

        # Создание меток с информацией об уровнях
        for i, (level_name, level_desc) in enumerate(levels_info):
            name_label = ttk.Label(levels_frame, text=level_name,
                                   font=self.stats_font, foreground="blue")
            name_label.grid(row=i, column=0, padx=(0, 5), pady=2,
sticky=tk.W)

            desc_label = ttk.Label(levels_frame, text=level_desc,
font=self.stats_font)
            desc_label.grid(row=i, column=1, padx=(5, 0), pady=2,
sticky=tk.W)

        # 7. ФРЕЙМ С КНОПКАМИ УПРАВЛЕНИЯ
        buttons_frame = ttk.Frame(main_frame)
        buttons_frame.grid(row=6, column=0, columnspan=3, pady=(10, 0))
```

```
        # Кнопка помощи
        help_btn = ttk.Button(
            buttons_frame,
            text="Помощь",
            command=self.show_help
        )
        help_btn.grid(row=0, column=0, padx=(0, 10))

        # Кнопка для показа таблицы умножения
        table_btn = ttk.Button(
            buttons_frame,
            text="Таблица умножения",
            command=self.show_multiplication_table
        )
        table_btn.grid(row=0, column=1, padx=10)

        # Кнопка выхода из приложения
        exit_btn = ttk.Button(
            buttons_frame,
            text="Выход",
            command=self.root.quit  # Метод закрытия главного окна
        )
        exit_btn.grid(row=0, column=2, padx=(10, 0))

        # Настройка веса строк и столбцов для правильного растягивания
        for i in range(7):
            main_frame.rowconfigure(i, weight=1)  # Растягивание строк

        for i in range(3):
            main_frame.columnconfigure(i, weight=1)  # Растягивание столбцов

    def start_new_game(self):
        """
        Начинает новую игру с выбранным уровнем сложности.
        Сбрасывает интерфейс и инициализирует новую игровую сессию.
        """
        level = self.level_combo.get()  # Получение выбранного уровня
сложности
        self.game.start_new_game(level)  # Начало новой игры в логике

        # Сброс интерфейса
        self.user_answer.set("")  # Очистка поля ввода
        # Установка текста первого вопроса
        self.question_label.config(text=self.game.current_question["text"])
        self.answer_entry.focus()  # Установка фокуса на поле ввода
        self.update_stats()  # Обновление статистики

    def check_answer(self):
        """
        Проверяет ответ пользователя на текущий вопрос.
        Обрабатывает результат проверки и обновляет интерфейс.
        """
        # Проверка, что игра начата
        if not self.game.current_question:
            messagebox.showinfo("Внимание", "Сначала начните новую игру!")
            return

        # Получение ответа пользователя
        user_answer = self.user_answer.get().strip()
```

```
        if not user_answer:  # Проверка на пустой ответ
            messagebox.showwarning("Внимание", "Введите ответ!")
            return

        # Проверка ответа через игровую логику
        is_correct, points = self.game.check_answer(user_answer)

        if is_correct:
            # Показ сообщения об успехе
            messagebox.showinfo("Правильно!", f"Верно! +{points} очков")
            # Генерация нового вопроса
            self.game.generate_question()
            # Обновление текста вопроса

self.question_label.config(text=self.game.current_question["text"])
        else:
            # Показ сообщения об ошибке с правильным ответом
            correct_answer = self.game.current_question["answer"]
            messagebox.showerror("Неправильно", f"Неверно! Правильный ответ:
{correct_answer}")

        # Сброс поля ввода
        self.user_answer.set("")
        # Обновление статистики
        self.update_stats()
        # Возврат фокуса на поле ввода
        self.answer_entry.focus()

    def show_hint(self):
        """
        Показывает подсказку для текущего вопроса.
        Отображает таблицу умножения для первого числа из вопроса.
        """
        if not self.game.current_question:
            return  # Если вопрос не задан, ничего не делаем

        # Получение данных текущего вопроса
        question = self.game.current_question
        a = question["a"]  # Первое число
        b = question["b"]  # Второе число

        # Формирование текста подсказки
        hint_text = f"{a} × {b} = "

        # Добавление таблицы умножения для первого числа
        hint_text += f"\n\nТаблица умножения на {a}:"
        for i in range(1, 11):
            hint_text += f"\n{a} × {i} = {a * i}"

        # Показ подсказки в диалоговом окне
        messagebox.showinfo("Подсказка", hint_text)

    def update_stats(self):
        """
        Обновляет отображение статистики на экране.
        Получает актуальные данные из игровой логики и обновляет метки.
        """
        stats = self.game.get_stats()  # Получение текущей статистики
```

```
        # Обновление каждой метки статистики
        for key, label in self.stats_labels.items():
            if key == "accuracy":
                # Для точности добавляем знак процента
                label.config(text=f"{stats[key]}%")
            else:
                # Для остальных показателей просто значение
                label.config(text=str(stats[key]))

    def update_timer(self):
        """
        Обновляет отображение времени игры.
        Вызывается периодически для обновления таймера.
        """
        if self.game.start_time:
            # Получение прошедшего времени в секундах
            elapsed = self.game.get_time_elapsed()
            # Обновление текста метки времени
            self.time_var.set(f"Время: {elapsed} сек")

        # Планирование следующего обновления через 1000 мс (1 секунда)
        self.root.after(1000, self.update_timer)

    def show_help(self):
        """
        Показывает справочную информацию об игре.
        Отображает правила игры и инструкции по использованию.
        """
        help_text = """
        КАК ИГРАТЬ:

        1. Выберите уровень сложности:
           - Легкий: числа от 1 до 5
           - Средний: числа от 1 до 10
           - Сложный: числа от 5 до 15

        2. Нажмите "Новая игра"

        3. Решайте примеры на умножение

        4. Вводите ответ в поле и нажимайте "Проверить" или Enter

        5. Используйте "Подсказку" если нужно

        СИСТЕМА ОЧКОВ:
        - Легкий уровень: 5 очков за правильный ответ
        - Средний уровень: 10 очков за правильный ответ
        - Сложный уровень: 15 очков за правильный ответ

        ПРАВИЛА:
        - Статистика сохраняется только во время текущей игры
        - При нажатии "Новая игра" статистика сбрасывается
        - Игра не имеет конца - играйте сколько хотите!

        Удачи!
        """

        # Показ справочной информации в диалоговом окне
        messagebox.showinfo("Помощь", help_text)
```

```
    def show_multiplication_table(self):
        """
        Открывает новое окно с таблицей умножения.
        Позволяет пользователю просматривать таблицу умножения для справки.
        """
        # Создание нового окна (дочернего по отношению к главному)
        table_window = tk.Toplevel(self.root)
        table_window.title("Таблица умножения")
        table_window.geometry("400x500")
        table_window.configure(bg='white')

        # Создание вкладок для разных диапазонов таблицы умножения
        notebook = ttk.Notebook(table_window)

        # Вкладка с таблицей умножения для чисел 1-5
        frame1 = ttk.Frame(notebook)
        self.create_table_frame(frame1, 1, 5)
        notebook.add(frame1, text="1-5")

        # Вкладка с таблицей умножения для чисел 6-10
        frame2 = ttk.Frame(notebook)
        self.create_table_frame(frame2, 6, 10)
        notebook.add(frame2, text="6-10")

        # Размещение набора вкладок в окне
        notebook.pack(expand=True, fill='both')

    def create_table_frame(self, parent, start, end):
        """
        Создает фрейм с таблицей умножения для заданного диапазона чисел.

        Аргументы:
            parent: родительский виджет для размещения элементов
            start (int): начальное число диапазона
            end (int): конечное число диапазона
        """
        # Создание текстового виджета с прокруткой
        text_widget = tk.Text(parent, wrap=tk.WORD, font=("Courier", 12),
bg='white')
        scrollbar = ttk.Scrollbar(parent, orient=tk.VERTICAL,
command=text_widget.yview)
        text_widget.configure(yscrollcommand=scrollbar.set)

        # Размещение полосы прокрутки и текстового виджета
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        text_widget.pack(side=tk.LEFT, expand=True, fill='both')

        # Генерация текста таблицы умножения
        table_text = ""
        for i in range(start, end + 1):
            table_text += f"\nТаблица умножения на {i}:\n"
            # Строки таблицы умножения для числа i
            for j in range(1, 11):
                table_text += f"{i:2} × {j:2} = {i*j:3}\n"
            table_text += "-" * 20 + "\n"  # Разделитель между таблицами

        # Вставка сгенерированного текста
        text_widget.insert(1.0, table_text)
```

```
        # Установка режима "только для чтения"
        text_widget.config(state=tk.DISABLED)


def main():
    """
    Основная функция запуска приложения.
    Создает главное окно, экземпляр приложения и запускает главный цикл
обработки событий.
    """
    root = tk.Tk()  # Создание главного окна приложения
    app = MultiplicationTrainerGUI(root)  # Создание экземпляра приложения
    root.mainloop()  # Запуск главного цикла обработки событий
```

main.cpp

```
"""
Основной файл для запуска тренажера таблицы умножения
"""
# Импорт функции main из модуля multiplication_gui
from multiplication_gui import main

# Вызов функции main() для запуска приложения
if __name__ == "__main__":
    main()
```

Скриншоты работы приложения: