МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

по теме «Генетические алгоритмы»

Тема: Задача о кратчайшем пути.

Злобин А.С.

Студенты гр. 0382 _____ Андрющенко К.С.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2022

Цель работы

Разработать и реализовать программу, решающую одну из оптимизационных задач (файл “Варианты”) с использованием генетических алгоритмов (ГА), а также визуализирующая работу алгоритма.

Задача

Вариант 9

Задача о кратчайшем пути (задача поиска самого короткого пути (цепи) между двумя точками (вершинами) на графе, в которой минимизируется сумма весов рёбер, составляющих путь).

Необходимо реализовать вариант о нахождении кратчайшего пути между всеми парами вершин в графе, т.е. вывести матрицу кратчайших путей Входные данные:

- Список вершин
- Список ребер

Выполнение работы

Используемый язык программирования: Python 3.

Выполнение работы

1. Скетч GUI. Взаимодействие пользователя с программой.

   Предварительный набросок интерфейса программы:

   - Основное окно программы

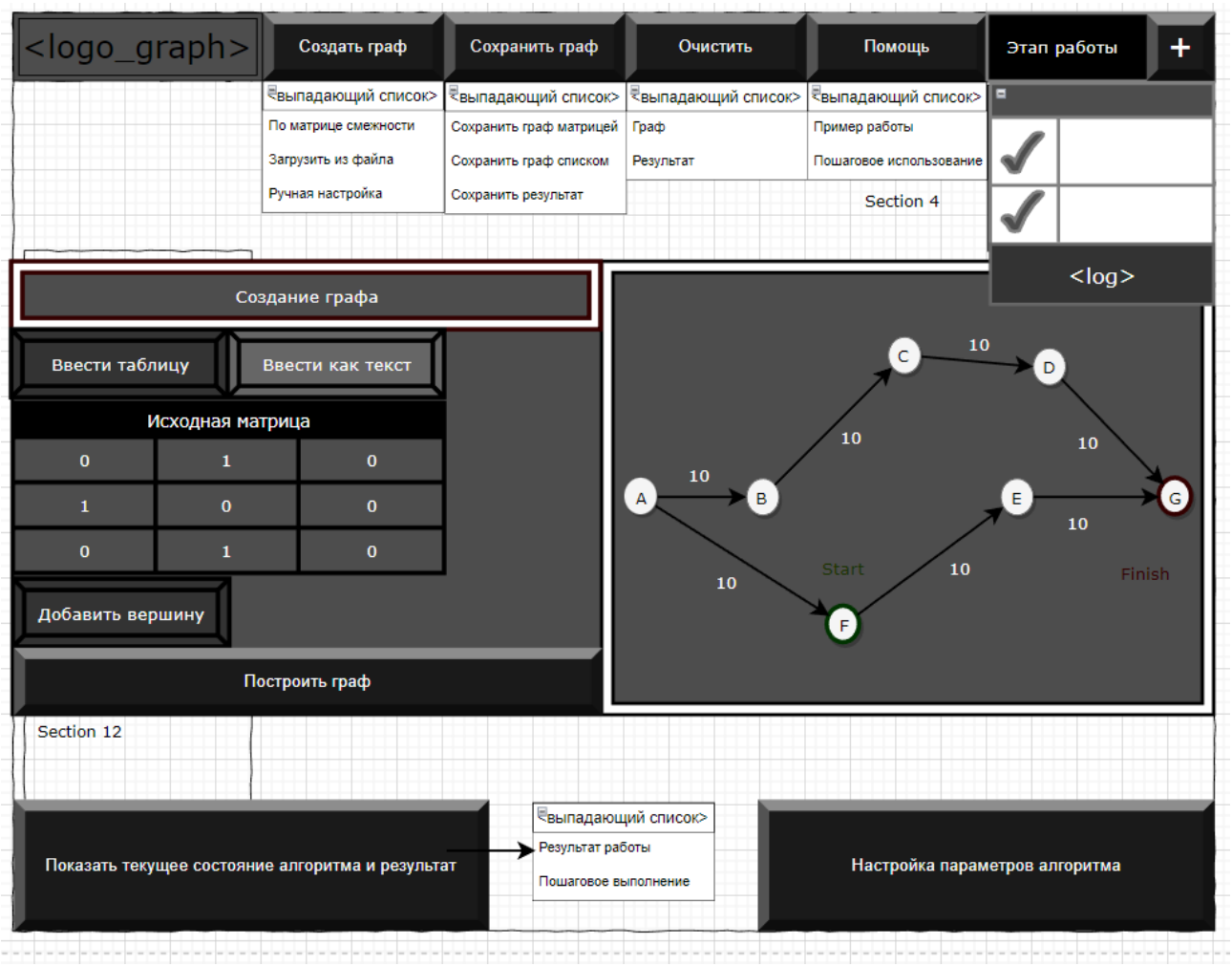

Рисунок 1 - Основное окно программы

- Меню настроек

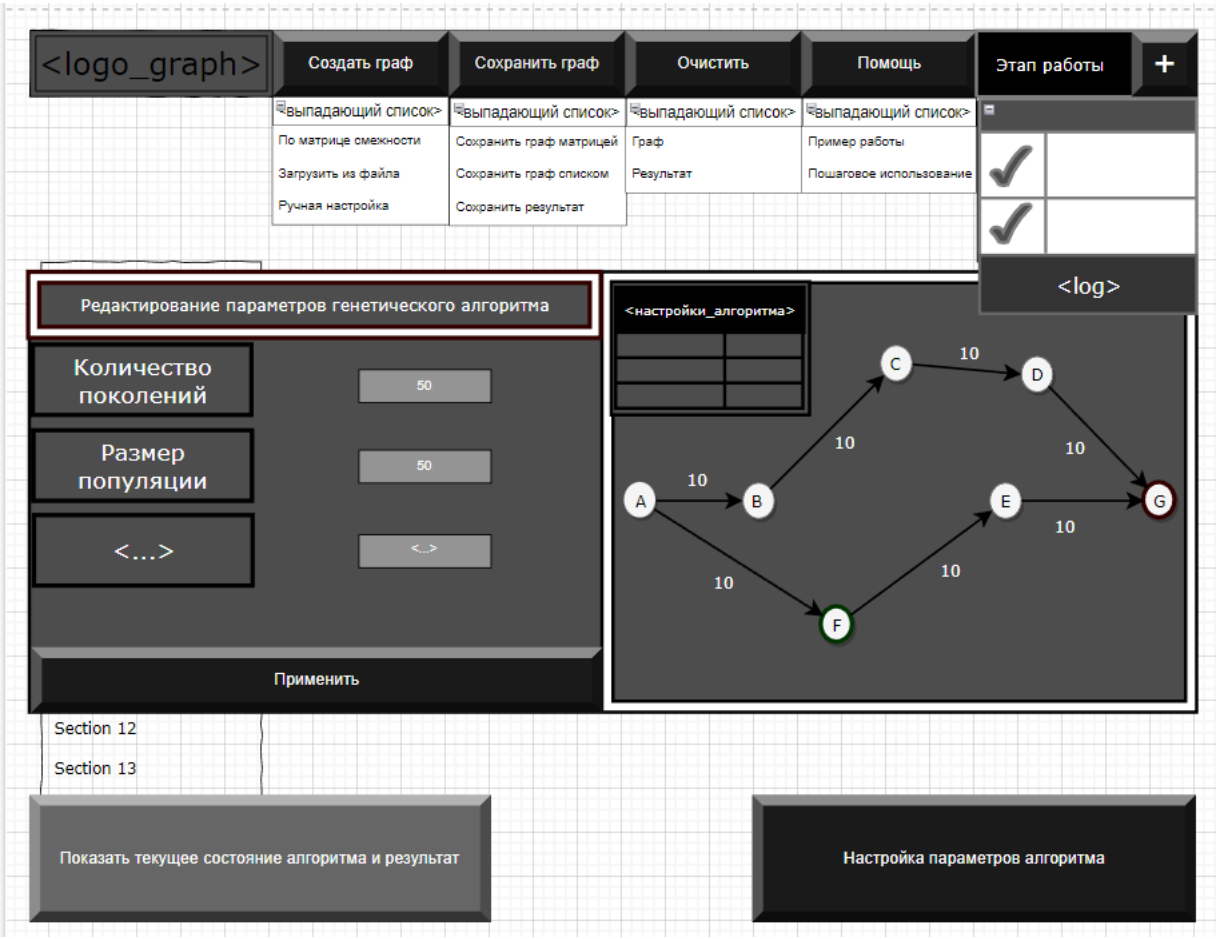

Рисунок 2 - Меню настроек параметров генетического алгоритма

- Визуализация работы алгоритма

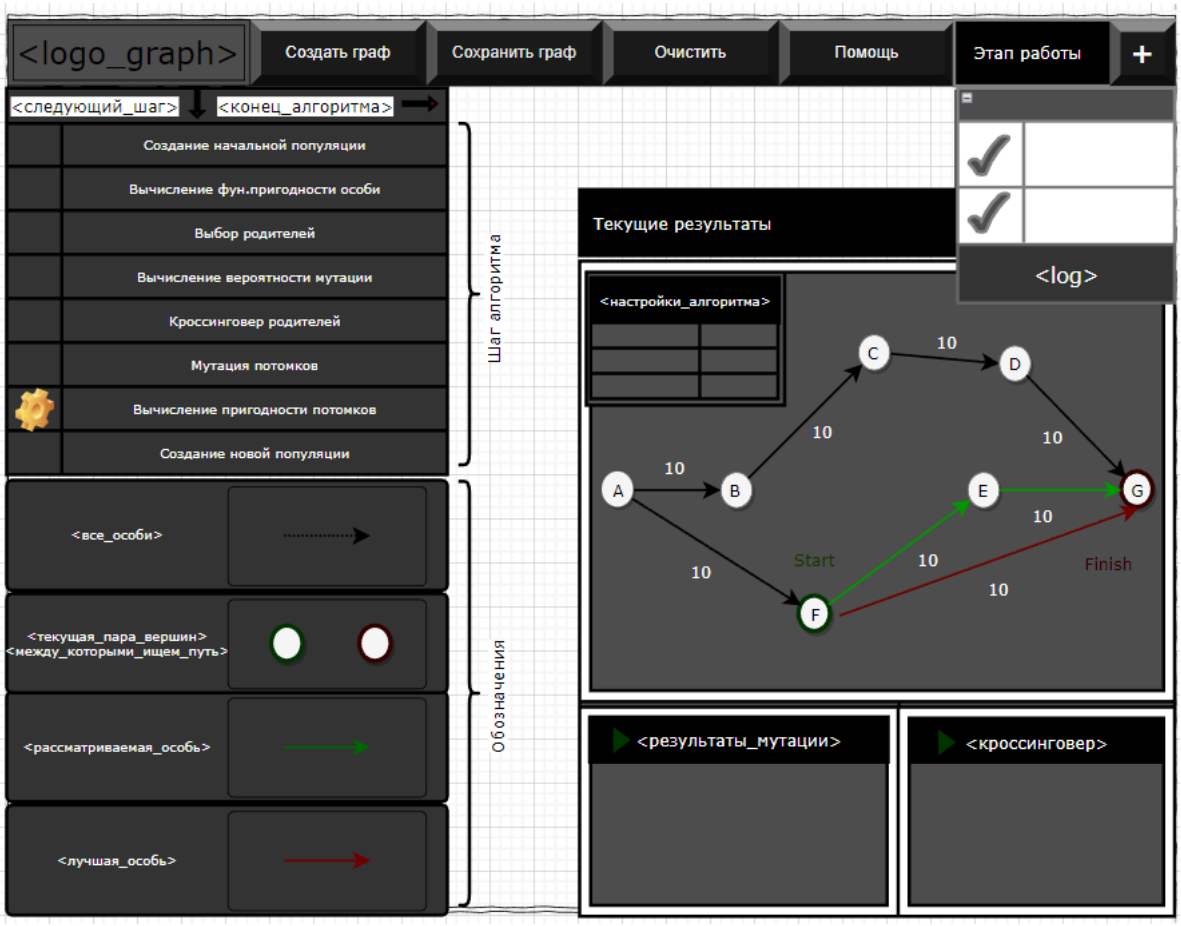

Рисунок 3 - Визуализация работы алгоритма

Описание сценария взаимодействия пользователь с программой.

«Основное окно программы»

При запуске программы открывается основное окно.

«Меню настроек»

В этом окне будут присутствовать область настроек параметров алгоритма (см. Рисунок 2), такие как размер популяции, количество поколений, вероятность кроссинговера и т. д. Эти параметры пользователю нужно будет указать (либо оставить значения по умолчанию) перед запуском алгоритма.

«Визуализация работы алгоритма»

Демонстрация работы алгоритма запускается по нажатию кнопки «показать текущее состояние алгоритма и результат». В правом окне отображаются пути между двумя вершинами, которые представляют собой

особей поколения. Ниже этого окна будут отображаться результаты мутаций и кроссинговера для текущего поколения. В левом окне находятся две кнопки для управления визуализацией алгоритма. Кнопка «Следующий шаг» переносит нас на следующий шаг алгоритма, список которых перечислен ниже. Текущий шаг отмечен соответствующим символом. Кнопка «Конец алгоритма» пропускает визуализацию и переносит нас сразу к результату выполнения алгоритма. Ниже списка шагов приведены условные обозначения, использующиеся при визуализации особей поколения.

«Ввод данных»

В качестве входных данных используется список вершин и список рёбер. Пользователь будет иметь возможность задать граф, для которого будут вычисляться кратчайшие пути, следующими способами:

- С помощью матрицы смежности

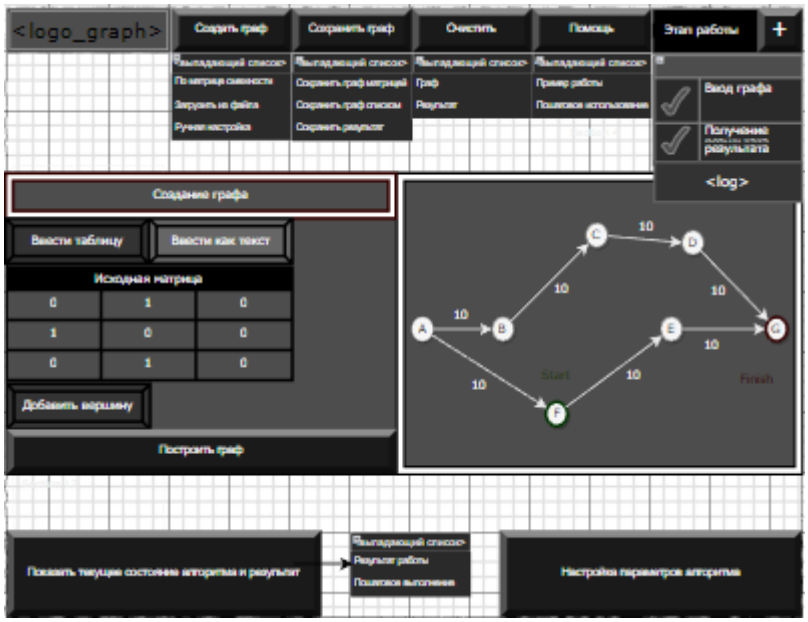

Рисунок 4 – Ввод графа с помощью матрицы смежности заполняя таблицу

Пользователь заполняет ячейки таблицы указывая вес ребра (ячейка пуста или 0 – нет пути между вершинами), в окне справа отрисовывается текущий граф, после нажатия кнопки «Построить граф». В случае нехватки ячеек, пользователь может добавить вершину через соответствующую кнопку

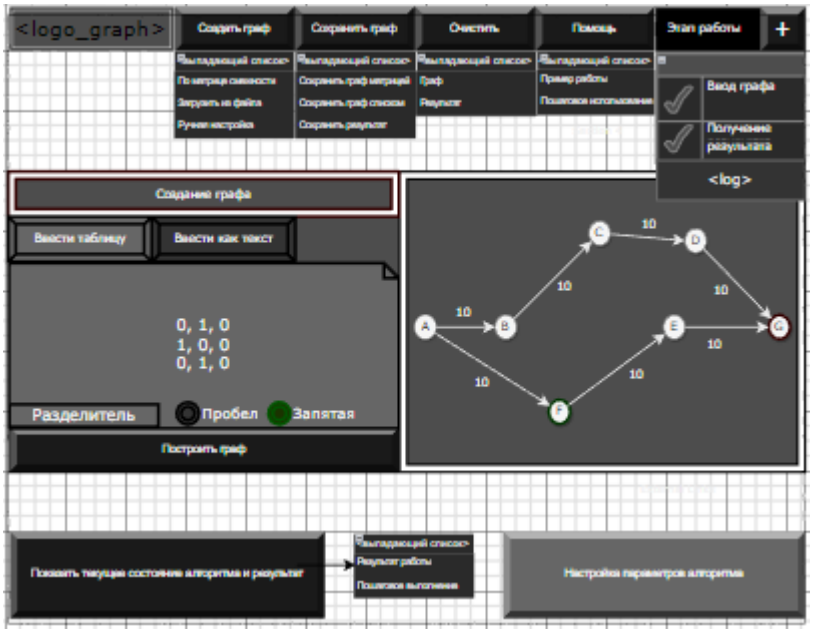

Рисунок 5 – Текстовый ввод графа с помощью матрицы смежности

Пользователь вводит матрицу в строку, указав один из предложенных разделителей между элементами матрицы.

- «Рисуя» граф в представленном окне с помощью инструментов «создать вершину», «создать направленное ребро» и «создать ненаправленное ребро».

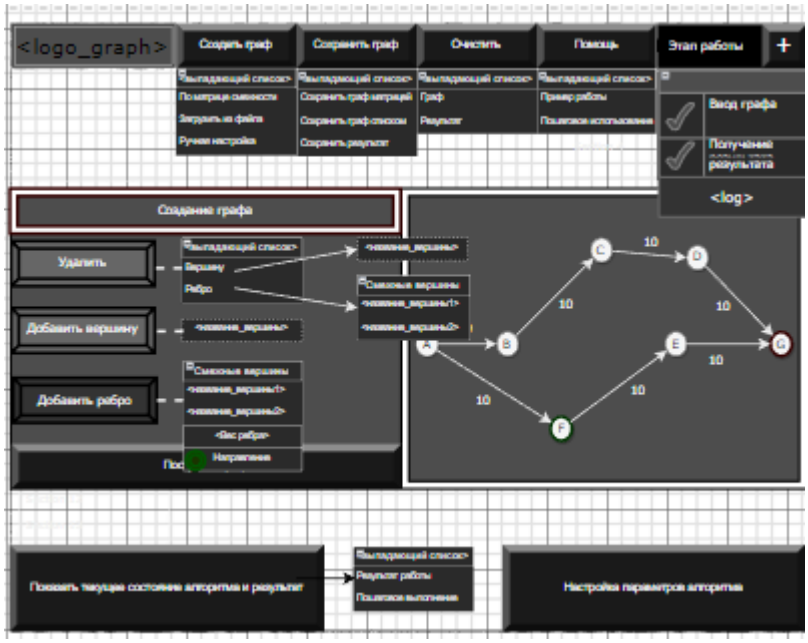

Рисунок 6 – Ручная настройка графа

После нажатия кнопки удалить появляется выпадающий список с возможными для удаления элементами. В случае удаления вершины пользователь должен ввести ее имя. При удалении ребра необходимо ввести вершину начало и вершину конец. В случае ненаправленного ребра повторить данный пункт 2 раза, изменяя порядок вершин.

- Загрузка из файла.

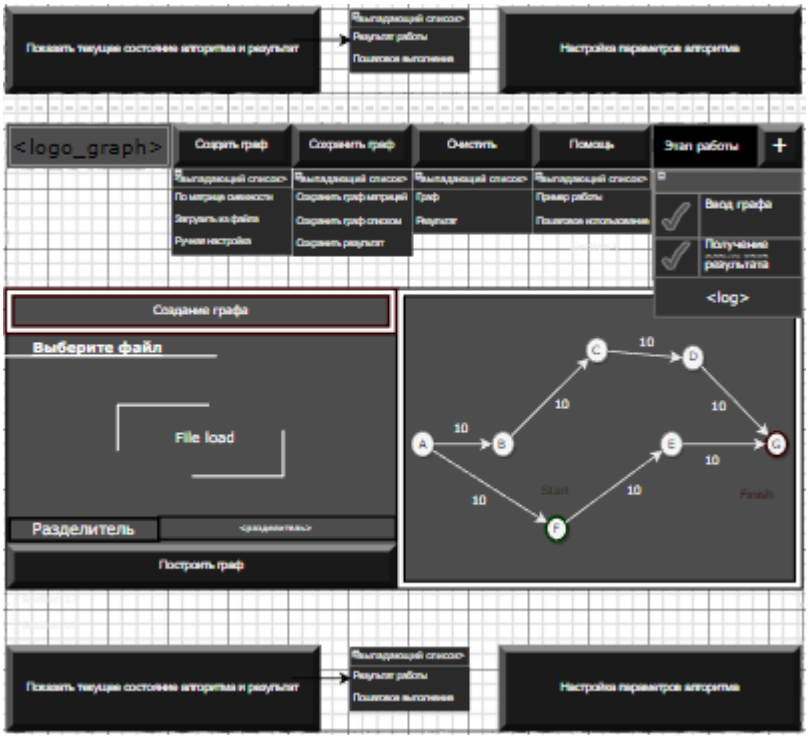

Рисунок 7 – Ввод графа из файла

Пользователю необходимо выбрать нужный файл.

При любом варианте ввода можно посмотреть текущий вид графа в окне справа.

В меню с выпадающим списками вверху окна можно выбрать параметры очистки полей, ввода и сохранения графа. Под кнопкой помощь можно выбрать из списка вариант, в котором все поля будут заполнены и показан результат (пример работы), или получение набора инструкций по работе с GUI.

Этапы работы можно скрыть, в выпадающем окне будут показаны результаты процесса работы приложения.

Описание генетического алгоритма.

Первым делом нам нужно формализовать задачу и определиться со способом хранения информации в хромосомах. Здесь могут быть самые разные варианты, однако остановимся на следующем. Одна хромосома описывает все возможные маршруты от исходной точки до всех остальных вершин графа.

Узлы, для удобства, пронумерованы от 0 до n - 1 (всего n узлов). Каждый маршрут представлен отдельным списком. Так как длина пути изначально неизвестна, то он заканчивается тогда, когда встречается вершина с номером назначения.

Для хранения графа будем использовать матрицу смежности, которая на пересечении строк и столбцов содержит значения весов дуг между соответствующими вершинами. Те клетки, что не содержат значений, означают отсутствие связей. По идее, здесь можно прописать бесконечные величины, означающие бесконечно длинные маршруты.

Следующим шагом в программе нужно определить, как вычислять приспособленность отдельных особей, как выполнять скрещивание и мутации. Начнем с функции приспособленности. Она должна возвращать кортеж значений, которые являются суммой длин путей до вершин графа.

В функции мутации перебираем списки маршрутов особи и для каждого выполняем перемешивание индексов с некоторой заданной вероятностью.

В качестве функции скрещивания будем использовать алгоритм упорядоченного скрещивания.

2. Начало реализации

   Частичная реализация GUI

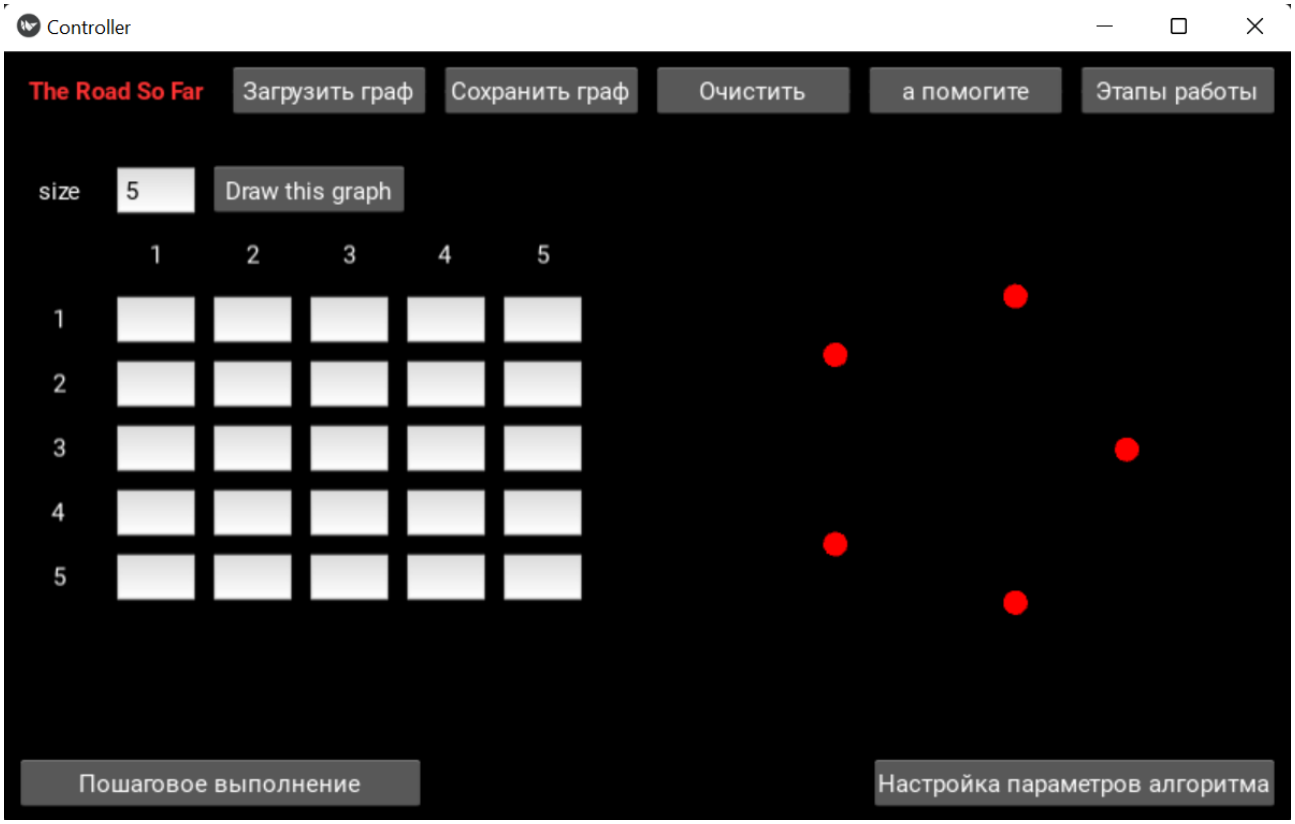

Рисунок 7 – Реализация GUI

Для реализации графического интерфейса будет использоваться фреймворк Kivy.

Был написан класс ControllerApp и Conroller, а так же файл, генерирующий разметку основной страницы программы.

Структура расположения объектов на основной странице представляет собой комбинацию BoxLayout, что позволяет упростить размещение основных элементов.

На основной странице расположен заголовок с управляющими кнопками, такими как загрузка / сохранение графа, «Помощь», с инструкциями к использованию, а также результаты работы. Все пункты заголовка, за исключением загрузки файла, реализованы ввиде выпадающих меню (класс DropDown), в которых перечислены более конкретные действия (например, различные способы сохранения графа, такие как сохранение ввиде матрицы смежности или списка).

Ниже расположены два окна – левое для ввода матрицы смежности, правое для отрисовки графа. Окно для ввода графа представляет собой GridLayout, заполненную полями для ввода текста. Размер матрицы смежности указывается выше, в поле size, после чего автоматически происходит обновление поля для ввода матрицы до указанного размера. Это окно имеет свои полосы прокрутки, на случай, если матрица будет большая. Они реализованы с помощью класса ScrollView.

Отрисовка графа происходит по нажатию кнопки «Draw this graph». При этом используется пакет kivy.graphics. Вершины графа располагаются по окружности. В дальнейшем будут добавлены подписи к вершинам и рёбрам графа.

Окно с настройками пока не реализовано, но оно будет отображаться вместо окна с вводом графа и будет представлять собой набор полей для ввода текста. Окно с ходом алгоритма будет реализовано позже.

Реализовано хранения данных и основные элементы ГА

Для хранения информации о графе и настройках генетического алгоритма были реализованы классы Graph_alg и Settings_alg соответсвенно. Класс Graph_alg содержит два поля: inf и graph. Поле inf отвечает за хранение значения, которым обозначается несуществующий путь (по умолчанию указано 10). Величина пути не может быть равна inf. Поле graph хранит квадратную матрицу ввиде вложенных списков, которая является матрицей смежности. Также в этом классе определены методы get_graph и set_graph, с помощью которых можно получить и задать матрицу смежности. Метод save_data_graph выполняет сохранение графа в файл с расширением txt. Метод load_data_graph, соответственно, загружает граф из такого файла.

Класс Settings_alg отвечает за настройки самого генетического алгоритма. Он хранит поле const_alg, которое является словарём, и хранит основные параметры алгоритма. При инициализации класса задаются значения по умолчанию. Метод set_settings_alg принимает на вход два значения – имя параметра и его новое значение и меняет настройки алгоритма. Метод

set_settings_alg возвращает значение переданного параметра. Метод set_all_settings_alg принимает список, в котором по порядку перечислены значения всех параметров алгоритма и сохраняет эти значения в словарь с настройками. Метод get_all_settings_alg возвращает словарь настроек. Методы save_data_config и load_data_config отвечают за сохранение и загрузку настроек из файла. Формат файла сохранённых настроек имеет вид списка значений параметров в столбик. Метод set_setting_alg_from_file загружает настройки непосредственно в класс настроек.

Основные элементы генетического алгоритма реализованы в файлах sel.py и mutation.py. В файле mutation.py представлены основные функции реализации мутаций генов. За реализацию алгоритма мутации соседних генов отвечает функция mutation_neighb_trans. В ней случайным образом выбирается ген и меняется значением с соседним. В функции mutation_trans случайным образом выбираются два гена и их значения меняются местами. Гауссовская мутация реализована в функции mutation_gauss. Функция mut_gen отвечает за реализацию мутаций. В дальнейшем в ней будет реализована возможность выбора функции мутации.

В файле sel.py реализованы основные функции селекции, такие как: турнирный отбор, отбор по правилу рулетки (пропорциональная приспособленность), стохастическая универсальная выборка и выбор наиболее подходящей половины.

Функция создания индивида принимает вершину начала и конца маршрута для генов в качестве входных данных, строит маршрут, используя случайное число (без повторений) и создает особь.

Функция создания популяции принимает количество особей, каждая из которых описывается по вышеприведённому алгоритму.

Для скрещивания реализовано несколько функций:

- Единая точка. в этом методе гены после единственной точки заменяются генами другого родителя для создания двух потомков.

• Две точки. В этом методе гены между двумя точками заменяются генами другого родителя, чтобы создать двух потомков.
• Единая точка (модификация). Первую часть (до точки разреза) создаем по принципу обмена частей генов родителей, а вторую заполняем так, чтобы гены не повторялись.

3. Работающий прототип

Графический интерфейс был доработан. Корневым виджетом теперь являяется ScreenManager. Создано три класса, наследуемых от класса класса Screen: ScreenMain, ScreenRunAlg и ScreenSettings, отвечающих соответсвенно за главный экран, на котором находятся поля для ввода графа, запуск алгоритма, с визуализацией его пошагового выполнения и окно настроек. Переключение между «экранами» приложения происходит с помощью ScreenManager. Методы по обработки событий были перенесены из класса приложения в классы соответствующих экранов, элементы которого вызывают это событие. На экране настроек расположен список параметров с числовыми значениями, такие как размер популяции и т. д. и список параметров с возможностью выбора одного значения из приведённого выпадающего списка. На главном экране приложения был переписан алгоритм ввода матрицы. Теперь введённые в матрицу значения считываются и сохраняются в классе окна с постоянной периодичностью, что позволяет реализовать отрисовку введённого графа в реальном времени. На экране пошагового выполнения алгоритма расположен список этапов алгоритма и кнопка перехода к следующему этапу. Также, текущий этап выделен меткой ‘*’. В дальнейшем, рядом с этим списком будет интегрирована визуализация работы алгоритма с помощью matplotlib.

Этапы генетического алгоритма:

1. Формируем начальную популяцию - список, содержащий в себе набор заданного размера списков, состоящих из начальной вершины и возможных перестановок остальных вершин графа.

2. Вычисляем пригодность используя функцию fitness, считающую кумулятивную сумму каждой особи - длину маршрута.

3. Выбираем родителей из лучшей половины методом рулетки.

4. Определяем вероятность скрещивания и мутации каждой особи, основываясь на их пригодности, чем лучше особь - тем менее вероятна для нее мутация и более вероятно скрещивание.

5. Оцениваем пригодность мутировавших потомков, в случае хорошей пригодности, оставляем в популяции, убирая наименее пригодную особь, иначе особь не добавляется в популяцию.

6. Для слабой половины родителей проводим турнирный отбор и выполняем пункт 5.

Таким образом формируем новую популяцию и повторяем процесс необходимое количество итераций

4. Финальная версия