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

ОТЧЕТ

по учебной практике

Тема: Генетические алгоритмы

Студент гр. 0304 _____ Люлин Д.В.

Студент гр. 0304 _____ Максименко Е.М.

Студентка гр. 0304 _____ Говорющенко А.В.

Студент гр. 0304 _____ Алексеев Р.В.

Руководитель _____ Жангиров Т.Р.

Санкт-Петербург
2022

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Люлин Д.В. группы 0304

Студент Максименко Е.М. группы 0304

Студентка Говорющенко А.В. группы 0304

Студент Алексеев Р.В. группы 0304

Тема практики: Генетические алгоритмы

Задание на практику:

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

Задача:

Задача раскроя

Дана полубесконечная лента ткани фиксированной ширины, из нее необходимо вырезать прямоугольные участки ткани заданных размеров. Необходимо разместить прямоугольники таким образом чтобы минимизировать длину используемой ленты и количество отходов.

Сроки прохождения практики: 29.06.2022 – 12.07.2022

Дата сдачи отчета: 08.07.2022

Дата защиты отчета: 08.07.2022

Студент	_____	Люлин Д.В.
Студент	_____	Максименко Е.М.
Студентка	_____	Говорющенко А.В.
Студент	_____	Алексеев Р.В.
Руководитель	_____	Жангиров Т.Р.

АННОТАЦИЯ

Цель практики заключается в реализации программы, решающей задачу раскроя при помощи генетического алгоритма. Программа должны иметь графический интерфейс (GUI), визуализировать работу алгоритма, иметь возможность ввода данных через GUI или файл.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является реализация программы, решающей задачу раскроя при помощи генетического алгоритма, программа должна быть написана на языке C++. Программа должна иметь: графический интерфейс (GUI), возможность ввода данных через GUI или файл, пошаговую визуализацию работы ГА с возможностью перехода к концу алгоритма, настройку параметров ГА через GUI, отображение особей популяции с выделением лучшей особи, визуализацию кроссинговера и мутаций, текстовые логи с пояснениями к ГА. После выполнения программа не должна сразу закрываться, а должна давать возможность провести ГА заново, либо ввести другие данные.

1. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС

1.1. Скетч GUI.

Скетч графического интерфейса (GUI), который планируется реализовать представлен на рис. 1.

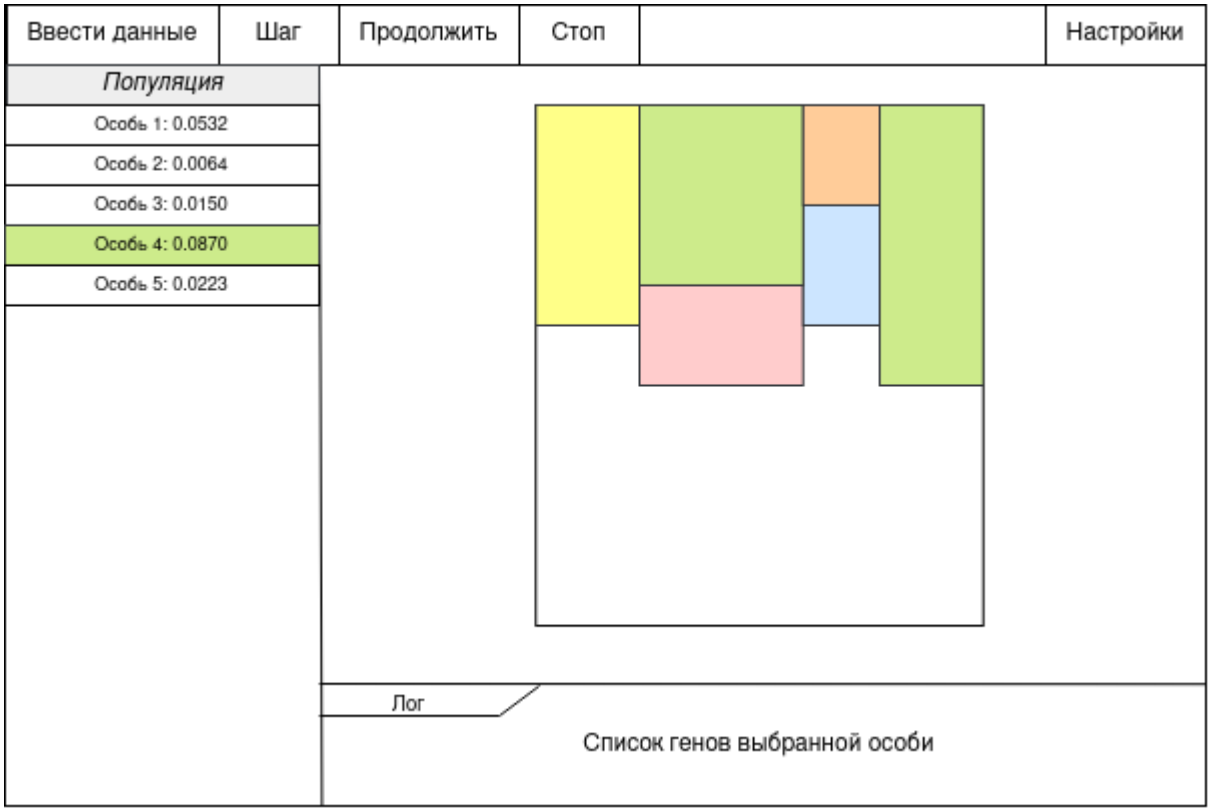

Рисунок 1. Скетч GUI

В левой части окна отображается список особей из текущей популяции. Для каждой особи написано значение функции пригодности, особь с максимальным значение выделена зелёным цветом.

В центральной части окна отображается лента с расположенными на ней прямоугольниками.

В нижней части окна находятся вкладки с логами и информацией о выбранной особи. Во вкладке с логами отображаются все сообщения логов. Во вкладке с информацией описаны все гены выбранной особи, а именно информация о каждом прямоугольнике: номер, координаты, поворот.

В верхней части окна расположен ряд кнопок:

- Ввести данные - открывается небольшое всплывающее окошко, в котором пользователь вводит ширину ленты и может добавлять прямоугольники разных размеров. При вводе прямоугольника ему автоматически присваивается случайный цвет, который сохраняется между особями. Это позволяет наблюдать за тем, как меняется положение данного прямоугольника в разных особях.
- Шаг - выполнить один шаг алгоритма. Кнопки "Ввести данные" и "Настройки" становятся недоступными.
- Продолжить - продолжить выполнение алгоритма с текущего момента до конца. Кнопки "Ввести данные" и "Настройки" становятся недоступными.
- Стоп - прервать выполнение алгоритма. Популяция очищается, ввод данных и настройки становится доступными.
- Настройки - отображение меню настроек. Меню представлено на рис. 2.

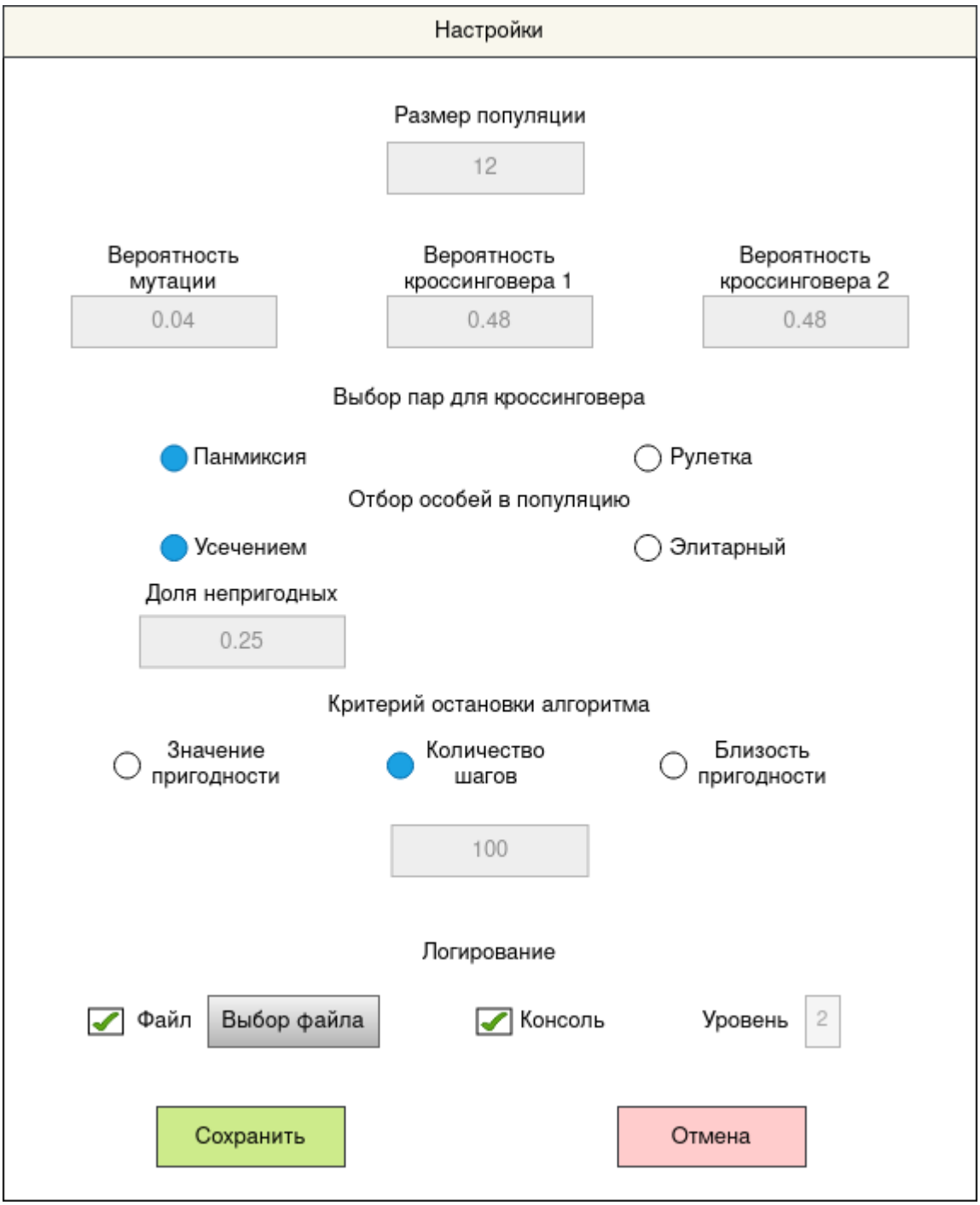

Рисунок 2. Скетч меню настроек ГА в GUI

1.2. Описание сценариев взаимодействия с GUI.

Основной сценарий взаимодействия пользователя с GUI:

1. Запустив программу, пользователь может выбрать опцию "Ввести данные". Всплывает окно, в котором он вводит ширину ленты, а также добавляет произвольное количество прямоугольников с заданной длиной и шириной. После ввода данных пользователь закрывает окно.
2. Перед началом работы алгоритма пользователь может настроить параметры генетического алгоритма, выбрав опцию "Настройки". Можно настроить следующие параметры: способ выбора родителей, способ отбора особей в новую популяцию, критерий остановки алгоритма, вероятности мутации и кроссинговеров, размер популяции. Также пользователь может настроить логирование: уровень подробности логов, запись логов в консоль и файл (и выбор файла).
3. Пользователь может запустить один шаг алгоритма, выбрав опцию "Шаг". С помощью этой опции можно подробно исследовать работу алгоритма. Если делать шаги во время кроссинговера, то синим цветом в левой части экрана будут отображаться пары родителей.
4. В любой момент пользователь может выбрать опцию "Продолжить", чтобы запустить алгоритм с текущего места не пошагово, а в обычном режиме.
5. В любой момент пользователь может выбрать опцию "Стоп", чтобы прервать работу алгоритма. Популяция удаляется, программа восстанавливает состояние из пункта 2. Становится доступным изменение настроек и входных данных, запуск алгоритма с начала.
6. При возникновении исключительных ситуаций всплывает окно, информирующее об ошибке, и программа делает то же, что при выборе опции "Стоп". Сообщение об ошибке записывается в лог.
7. При удачном завершении алгоритма популяция не очищается, пользователь может выделить особь с лучшим решением и посмотреть расположение прямоугольников. При этом кнопки "Шаг" и "Продолжить"

не дают эффекта. Чтобы начать алгоритм заново, необходимо нажать "Стоп".

2. ОПИСАНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Генетический алгоритм:

Основные понятия ГА:

- Ген - информация об одном прямоугольнике, представленная в виде тройки чисел: номер прямоугольника, координата X (координата Y берётся минимально возможной при данном X, и на алгоритм не влияет), поворот (0 или 90 градусов).
- Особь (хромосома) — какое-то решение задачи, т. е. расположение прямоугольников на ленте. За расположение каждого прямоугольника в данном решении отвечают гены. В каждом гене записана информация, по какому значению X разместить данный прямоугольник, номер данного прямоугольника, флаг, отвечающий за поворот на 90 градусов. Важен порядок генов, т. к. прямоугольники, за которые они отвечают, размещаются на ленте с минимально возможным значением Y в порядке расположения генов в массиве.
- Популяция - набор особей, в общем случае не упорядоченный, но во время отбора популяции может потребоваться сортировка.
- Функция пригодности (fitness) - функция от особи, значение функции в диапазоне [0;1]. Показывает качество особи, т.е. качество ответа на задачу. Качество особи считается как $1/h$, где h — длина использованной ленты, необходимой для расположения прямоугольников в данной конфигурации.

Генетический алгоритм можно настраивать, изменяя его параметры, а именно:

1. Способ выбора родителей.
2. Способ отбора особей в новую популяцию.
3. Критерий остановки алгоритма.
4. Вероятности мутации и кроссинговеров.

5. Размер популяции.

Процедуры кроссинговера и мутации не параметризуются, они фиксированы алгоритмом. Существует 2 вида кроссинговера. Происходит или один из них с какой-то вероятностью, или мутация. Эти вероятности может настраивать пользователь.

Операторы кроссинговера:

1. В гены потомка попадают значения координаты X и поворота из генов родителя 2, порядок не изменяется. Далее выбирается промежуток $0 \le l \le r \le n$, где n — количество генов особи. В гены потомка с индексами из интервала попадают номера прямоугольников из генов с этими же индексами генов родителя 2. В гены не из промежутка попадают номера прямоугольников родителя 1. Если возникает конфликт номеров прямоугольников, т. е. возникает попытка повторного добавления номера прямоугольника, который уже попал в гены потомка от родителя 2, выбирается другой номер: если данный номер прямоугольника у родителя 2 находится в гене с индексом k, берем номер прямоугольника из гена родителя 1 с индексом k, повторяем пока есть конфликты.
2. Также как и в предыдущем случае выбирается пара индексов $0 \le l \le r \le n$. Далее гены родителя 1 из промежутка полностью копируются по данным индексам в гены потомка. Гены не из промежутка будут копировать гены с такими же индексами родителя 2. Если возникают конфликты, они решаются также, как и в предыдущем операторе.

Оператор мутации:

Оператор мутации один, он заключается в том, что есть один родитель, выбираются два различных случайных гена, которые меняются местами. Также у данных генов может быть изменено значение поворота (с шансом в 50%).

Способ выбора родителей:

1. Панмиксия:

   Для каждой особи популяции случайным образом с равной вероятностью выбирается пара, с которой будет скрещивание. При этом допускается участие одной особи в нескольких скрещиваниях и скрещивание особи самой с собой.

2. Рулеточный отбор:

   Процедура выбора такая же, как в панмиксии, но разница в том, что парная особь выбирается не равновероятно, а пропорционально пригодности f каждой из N особей:

$$P(i)=\frac{f(i)}{\sum_{i=1}^{N} f(i)}$$

   Инбридинг и аутбридинг неприменимы к задаче, т.к. нет правильного способа подсчёта расстояния между родителями. В связи с тем, что при турнирном отборе во время скрещивания будет много повторяющихся особей, что не улучшает работу алгоритма, решено не использовать этот способ.

Способ отбора особей в новую популяцию:

1. Отбор усечением:

   Пусть N - фиксированный ранее размер популяции. Текущая популяция после кроссинговера (т.е. родители + дети, всего $2*N$) сортируется по возрастанию значения f пригодности особей. Задаётся параметр T в промежутке $(0;1]$, представляющий долю непригодных особей. Из особей, не попавших в непригодную долю, случайно выбирается одна. Этот процесс повторяется N раз. Данный параметр требует дополнительного ввода значения T.

2. Элитарный отбор:

Из текущей популяции после кроссинговера выбирается N особей с лучшим значением f пригодности, а остальные — отбрасываются.

Критерий остановки алгоритма:

1. Предельное значение пригодности:
   Остановка, если в текущей популяции есть хотя бы одна особь, значение функции пригодности которой больше или равно заданному.
2. Ограничение на количество шагов.
3. Критерий близости функции пригодности:
   Остановка, если для любой пары особей значения функции пригодности не превосходят заданную величину.

3. НАЧАЛО РЕАЛИЗАЦИИ

3.1. Реализация GUI

Согласно скетчу GUI из п. 1.1 с помощью фреймворка Qt был реализован графический интерфейс программы. На текущий момент реализованы основное окно (см. рис. 3), окно настроек (см. рис. 4) и окно ввода данных (см. рис. 5).

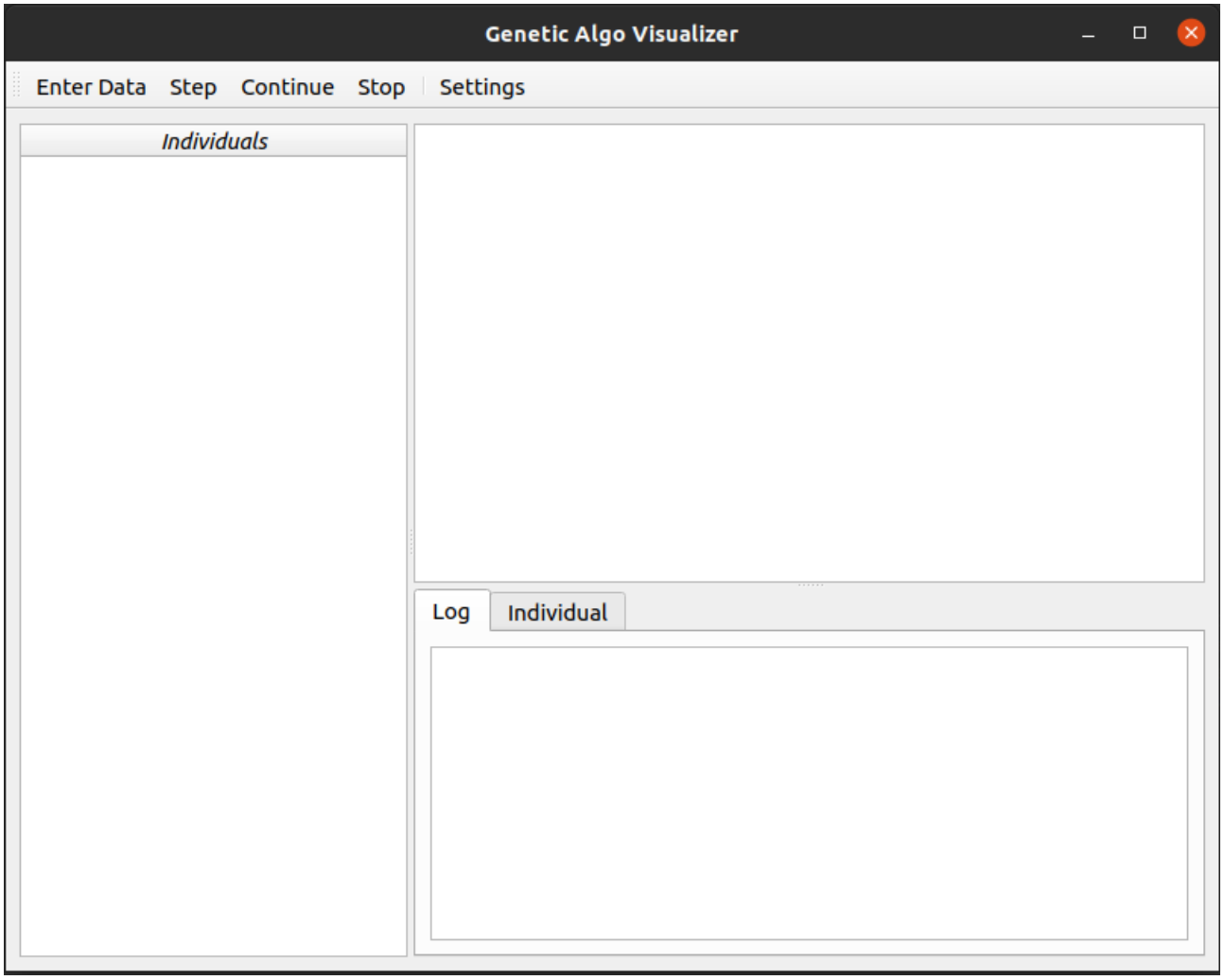

Рисунок 3. Основное окно программы

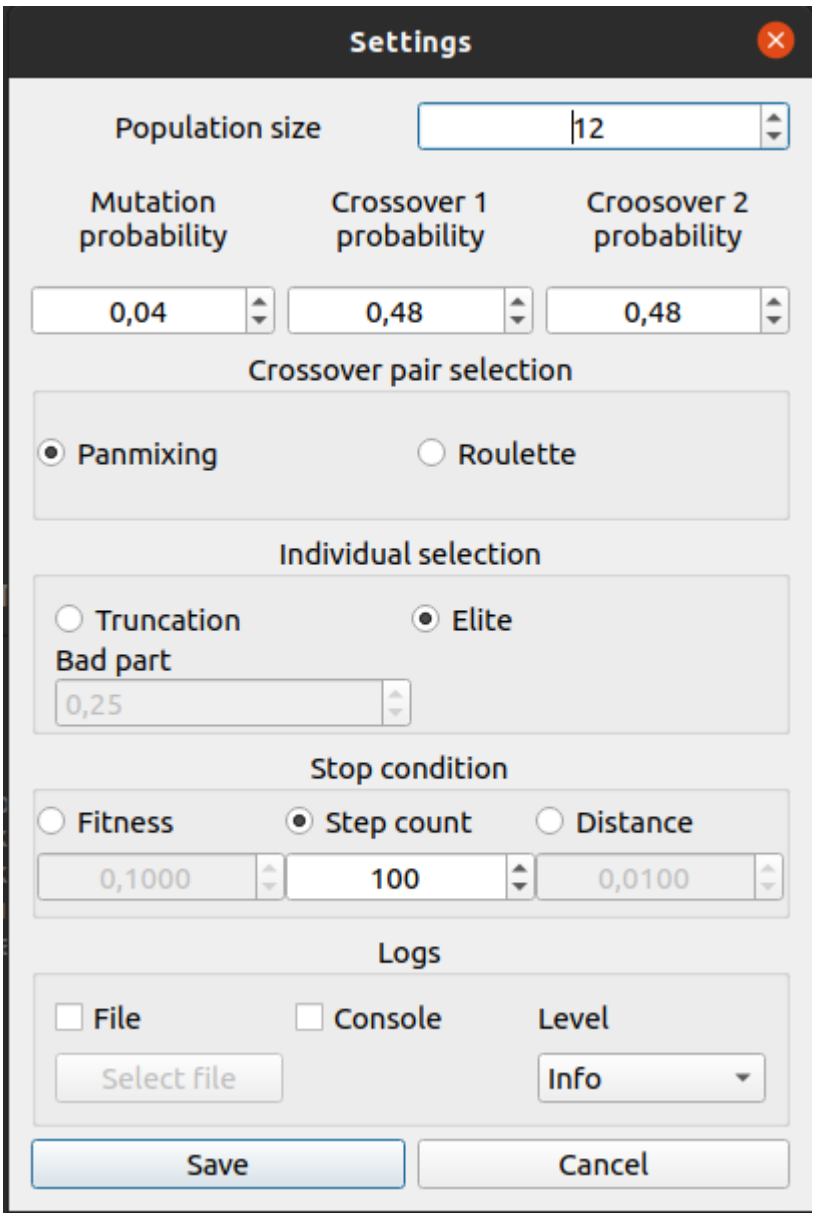

Рисунок 4. Окно настройки программы

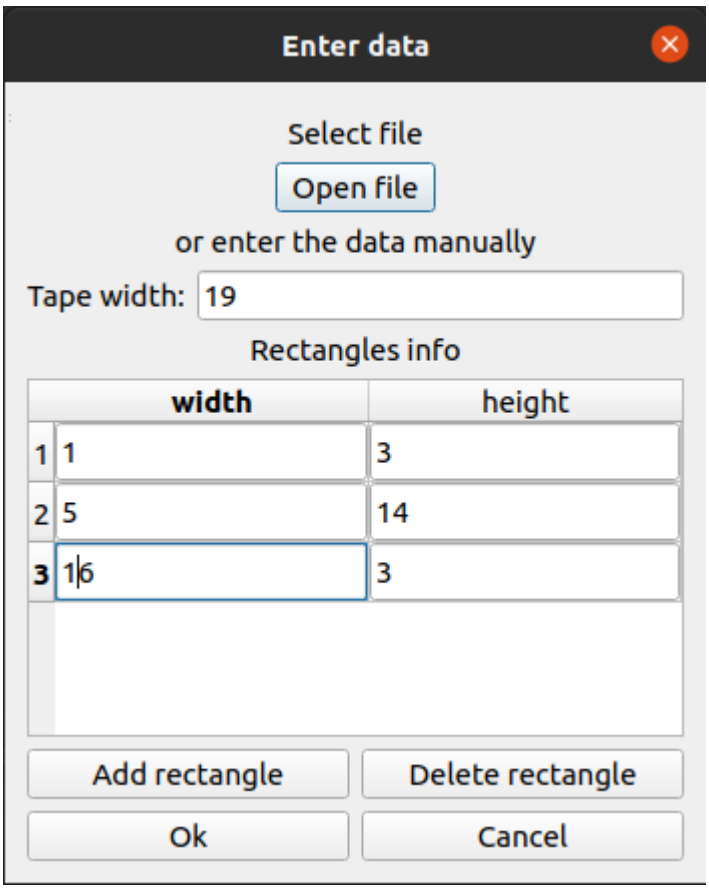

Рисунок 5. Окно ввода данных в программе

После ввода данных и настроек и нажатия кнопки «Ок» введенные данные или настройки сохраняются, и, при повторном открытии окна, поля окна будут заполнены ранее введенными данными.

3.2. Реализация генетического алгоритма

Согласно п.2 был реализован функционал генетического алгоритма для решения поставленной задачи. Для этого были написаны классы гена, особи и класса, отвечающего за сам генетический алгоритм. Для гибкой параметризации алгоритма были написаны интерфейсы операторов кроссинговера, мутации, выбора родителей, отбора особей в новое поколение. Также был написан интерфейс для различных критериев остановки алгоритма.

Были реализованы описанные в п.2 операторы: два оператора кроссинговера, оператор мутации, операторы выбора родителя методом панмиксии и рулетки, оператор отбора методом усечения и элитарным методом. При создании класса генетического алгоритма пользователь может

задать нужный оператор выбора родителя и отбора особей, а также размер популяции и вероятности кроссинговеров и мутации.

На данный момент взаимодействие между генетическим алгоритмом и графическим интерфейсом не реализовано.

3.3. Реализация логирования

В программе будет 4 уровня логов:

1. Нет логов;
2. Критический — самые важные сообщения и сообщения об ошибках, например, ввода пользователя;
3. Информация — основная информация о работе алгоритма;
4. Отладка — наиболее подробные сообщения.

Для управления логами создан класс, являющийся «синглтоном», *Log* со статическим методом *get_log()* для получения экземпляра класса. Класс Log имеет методы *critical()*, *info()*, *debug()*, принимающие на вход строки и передающие их непосредственно логгерам, если установлен уровень логирования не ниже заданного. Уровень логирования задается методом *set_log_level()*, принимающим *enum LogLevel*. Установка логгера происходит методом *add_logger()*, удаление - *delete_logger()*. При установке и удалении передаётся строковое имя логгера.

Каждый логгер реализует интерфейс *ILogger* с методом *write()*, принимающим строку и выводящим ее. Классы логгеров:

1. *TabLogger* - записывает лог в специальную вкладку в программе. В конструкторе принимает указатель на виджет, в который будет выводиться лог;
2. *ConsoleLogger* - выводит лог в поток *std::clog*;
3. *FileLogger* - выводит лог в файл. В конструкторе принимает имя файла, работа с файлом согласно *RAII*.

4. ЗАВЕРШЕНИЕ РЕАЛИЗАЦИИ

4.1 Конечная версия GUI

На данной итерации была закончена разработка графического интерфейса программы. Графический интерфейс полностью функционирует. Также была добавлена связь графического интерфейса и генетического алгоритма. Конечную версию основного окна программы см. на рис. 6.

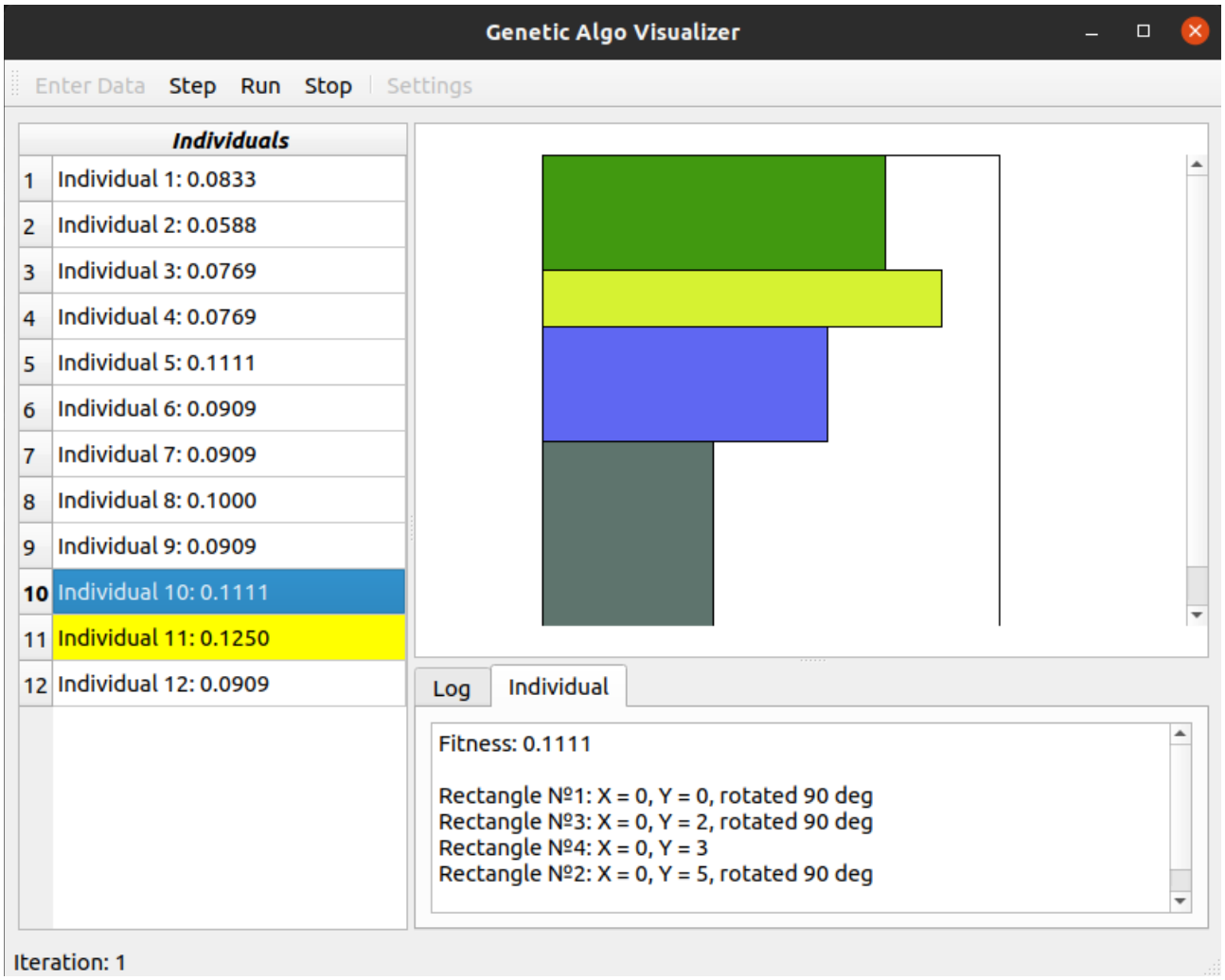

Рисунок 6. Конечная версия основного окна программы

Работой алгоритма можно управлять с помощью кнопок «Step» и «Run». По нажатию кнопки «Step» выполняется один шаг алгоритма: генерация начальной популяции, генерация новой особи, отбор особей в популяцию. Для демонстрации работы кроссинговера и мутации на этапе генерации новой особи в левом сайдбаре зеленым цветом подсвечиваются родители (родитель, в случае

мутации) особи, а сама новая особь подсвечивается бирюзовым цветом (см. рис. 7).

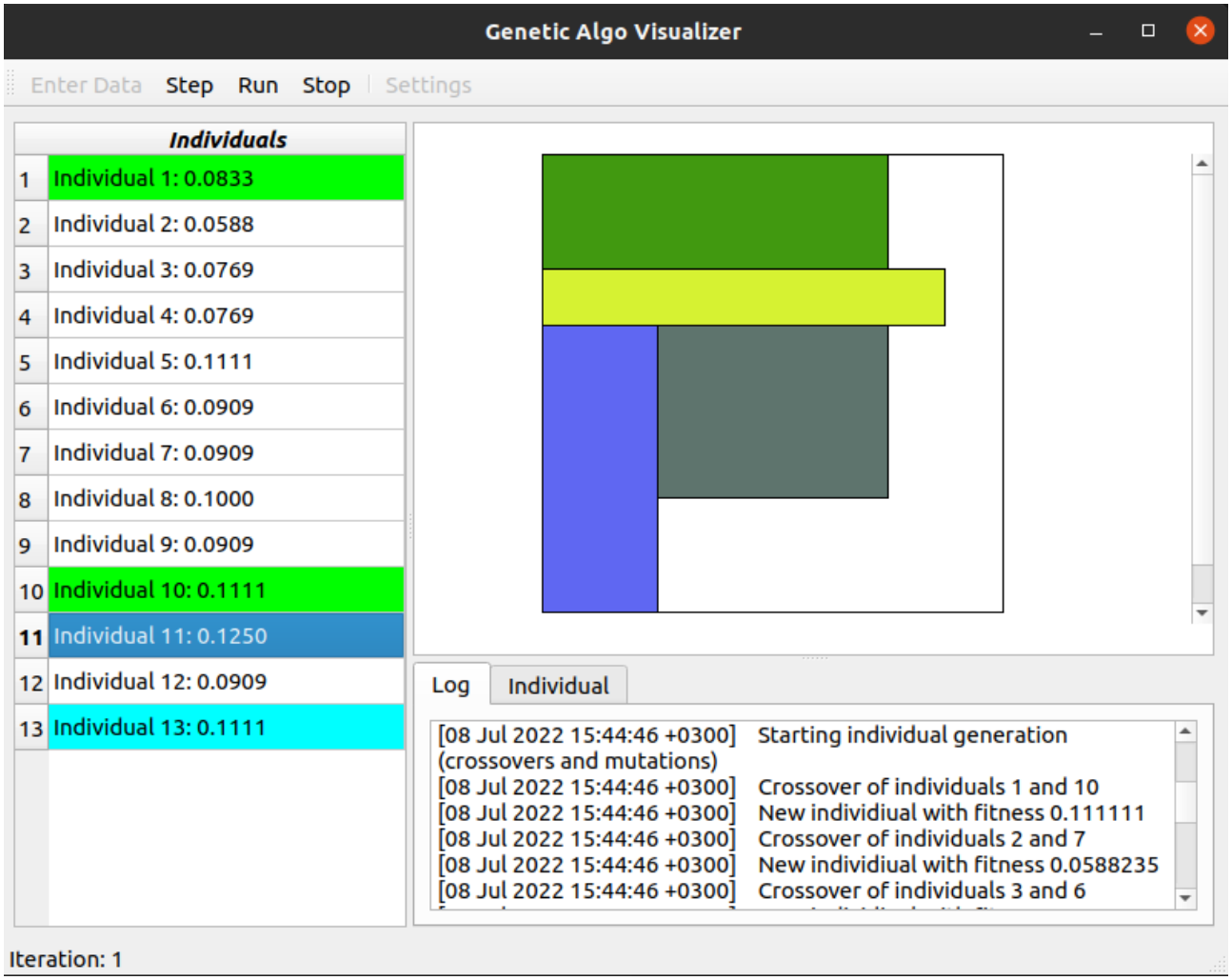

Рисунок 7. Демонстрация работы кроссинговера и мутаций в алгоритме

Если необходимо прогнать алгоритм до самого конца, то можно воспользоваться кнопкой «Run». По нажатию на данную кнопку алгоритм выполняется до тех пор, пока не будет выполнено условие остановки, указанное в настройках.

В случае, если пользователь ввел файл с некорректными данными или указал неверный файл для записи логов, программа выведет ошибку, а также сообщение об этом действии попадет в лог с уровнем «critical». Если пользователь не введет данные и попробует запустить алгоритм с помощью кнопок «Step» и «Run», ему также будет выведено сообщение о необходимости

ввести данные. Если какое-то решение недопустимо, при нажатии на него пользователь вместо ленты увидит соответствующее сообщение.

Также был добавлен перевод графического интерфейса на русский язык (см. рис. 8).

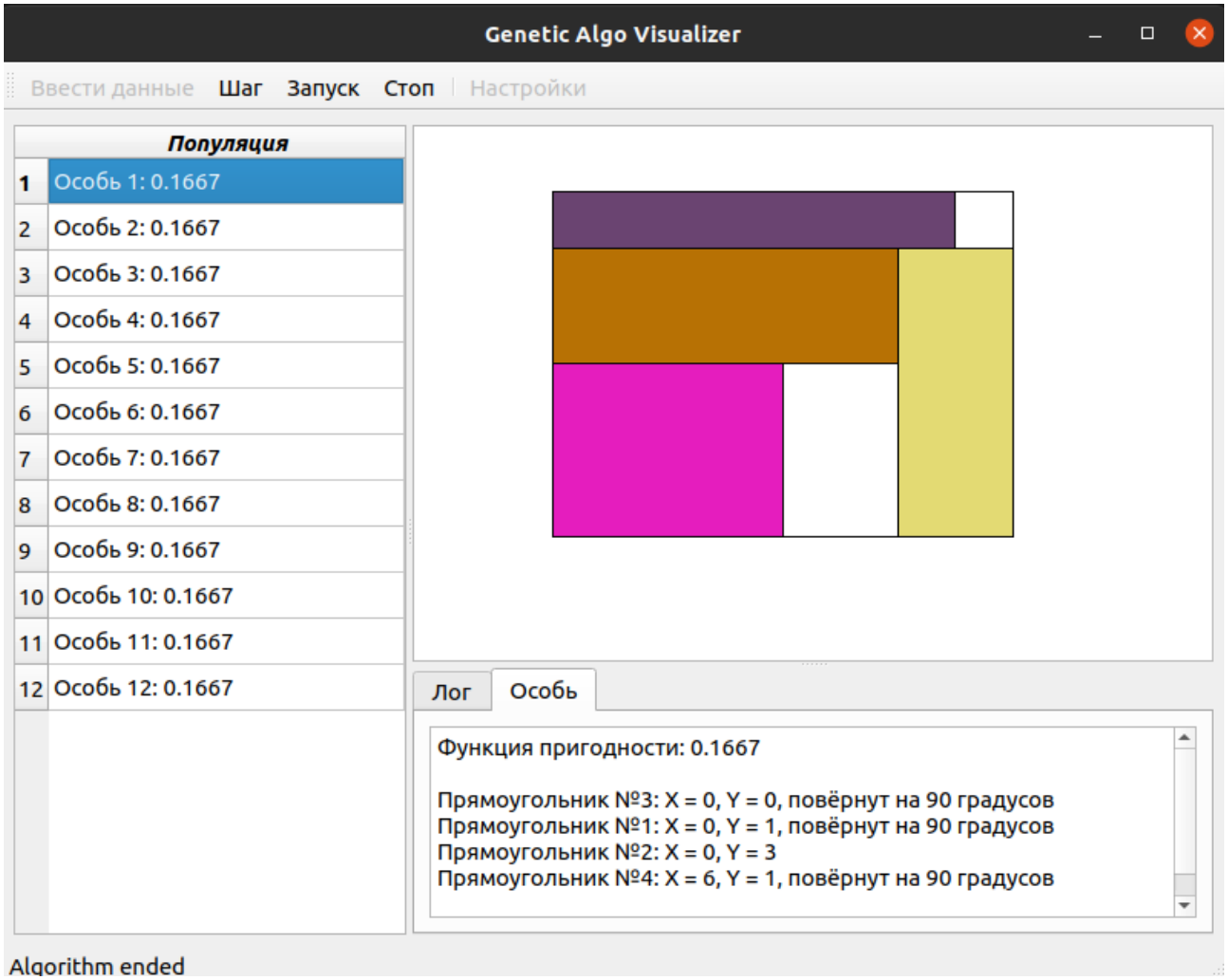

Рисунок 8. Перевод графического интерфейса на русский язык

4.2 Конечная версия генетического алгоритма

По части генетического алгоритма на данной итерации было произведено незначительное количество изменений. В частности, были исправлены некоторые баги: бесконечное выполнение алгоритма при введении в данных только одного прямоугольника, падение программы при попытке запуска алгоритма без введенных данных. Также были добавлены сообщения о работе алгоритма. Для взаимодействия с графическим интерфейсом было также

добавлено представление особи, которое содержит информацию о координатах расположения каждого прямоугольника и их повороте.

5. КЛАССЫ ПРОГРАММЫ

В данном разделе описаны только классы генетического алгоритма, гена и особи. Остальные классы имеют простое строение и их функционал не нуждается в пояснении.

5.1. Класс генетического алгоритма Packer

Класс Packer содержит поля:

- *std::vector<std::pair<size_t, size_t>> rectangles_* - информация о размерах прямоугольников из условия задачи;
- *std::vector<Individual> population_* - текущая популяция;
- *size_t iteration_count_* - количество итераций алгоритма;
- *size_t tape_width_* - ширина ленты из условия задачи;
- *size_t population_size_* - размер популяции (задается пользователем);
- *double probabilities_[3]* — вероятности выбора операторов кроссовера и мутации;
- *std::unique_ptr<ICrossover> crossover1_operator_* - первый оператор кроссинговера;
- *std::unique_ptr<ICrossover> crossover2_operator_* - второй оператор кроссинговера;
- *std::unique_ptr<IMutator> mutation_operator_* - оператор мутации;
- *std::unique_ptr<IParentSelector> parent_selection_operator_* - оператор выбора родителя (задается пользователем);
- *std::unique_ptr<ISelector> selection_operator_* - оператор отбора в популяцию;
- *std::unique_ptr<IStopCondition> stop_condition_* - условие остановки работы алгоритма.
- *Result::State current_state_* - текущее состояние алгоритма

Класс Packer содержит методы:

- *Packer(*

  const std::vector<std::pair<size_t, size_t>>& rectangles,

  size_t tape_width,

std::unique_ptr<IParentSelector>&& parent_selection_operator,
std::unique_ptr<ISelector>&& selection_operator,
std::unique_ptr<IStopCondition>&& stop_condition,
size_t population_size = 20,
double crossover1_probability = 0.45,
double crossover2_probability = 0.45,
double mutation_probability = 0.1
) - конструктор класса Packer, принимающий на вход размеры прямоугольников из условия задачи, размер ленты из условия задачи, оператор выбора родителей и отбора в новое поколение, условие остановки алгоритма, размер популяции, а также вероятности кроссинговеров и мутации. Задает настройки генетического алгоритма, после чего вызывает генерацию начальной популяции;

- *Result step()* - выполнение одного шага алгоритма и получение результата;
- *const std::vector<Individual>& get_population() const* — получение текущей популяции;
- *size_t get_iteration_count() const* — получение количества пройденных итераций алгоритма;
- *Result init_population()* - случайная генерация начального поколения и получение результата;
- *Result generate_new_breed()* - генерация нового поколения (без отбора);
- *Result selection()* - отбор особей в новую популяцию.

5.2. Класс гена Gene

Класс Gene содержит поля:

- *size_t coordinate_* - координату X расположения прямоугольника, за который отвечает данный ген;
- *size_t index_* - номер прямоугольника, за который отвечает данный ген;

- *bool rotate_* - флаг поворота на 90 градусов прямоугольника, за который отвечает данный ген.

Класс Gene содержит методы:

- *Gene(size_t coordinate, size_t index, bool rotate = false)* — конструктор класса, принимающий на вход координату X прямоугольника, его номер, а также флаг поворота на 90 градусов. Данными значениями инициализируются поля класса;
- *size_t get_coordinate() const* — получение координаты X прямоугольника;
- *size_t get_index() const* — получение номера прямоугольника;
- *bool get_rotation()* - получение флага поворота прямоугольника.

5.3. Класс особи Individual

Класс Individual содержит поля:

- *std::vector<Gene> genes_* - список генов данной особи;
- *bool is_feasible_* - флаг, является ли решение допустимым (решение является допустимым, если ни один из прямоугольников не выходит за границы ленты);
- *float fitness* — значение функции приспособленности данной особи.

Класс Individual содержит методы:

- *Individual(const std::vector<Gene>& genes, const std::vector<std::pair<size_t, size_t>> rectangles, size_t tape_width)* — конструктор класса, который принимает на вход список генов особи, размеры прямоугольников и ширину ленты из условия задачи. Гены записываются в соответствующее поле, вычисляется допустимость решения и значение функции приспособленности (если решение не допустимо, значение равно 0);
- *size_t genes_count() const* — количество генов у особи;
- *const Gene& get_gene(size_t gene_index) const* — получение конкретного гена особи по индексу;

- *const std::vector<Gene>& get_genes() const* — получение списка всех генов особи;
- *float get_fitness() const* — получение значения функции приспособленности особи;
- *bool is_feasible() const* — получение флага допустимости решения
- *bool check_is_feasible(const std::vector<std::pair<size_t, size_t>> rectangles, size_t tape_width) const* — проверка решения на допустимость;
- *float calculate_fitness(const std::vector<std::pair<size_t, size_t>> rectangles, size_t tape_width) const* — вычисление значения функции приспособленности особи;
- *const IndividualRepresentation& representation() const* — получение представления особи, т. е. информации о координатах расположения и поворотах всех прямоугольников. Метод применяется только для допустимых решений.

6. СБОРКА И ЗАПУСК ПРОГРАММЫ

Для сборки программы необходимо установить Qt версии 5 или выше. Также должен быть установлен компилятор с поддержкой стандарта C++17 или выше.

Для сборки программы необходимо перейти в корень проекта и прописать следующие команды:

```
qmake genetic_algorithm.pro
make
```

После этого программа соберется в директорию *build*.

Для запуска программы необходимо открыть полученный после сборки файл в директории *build*.

ЗАКЛЮЧЕНИЕ

Был создан скетч графического интерфейса программы и описан один из сценариев взаимодействия пользователя с графическим интерфейсом. Также были описаны и обоснованы модификации ГА, используемые для решения задачи раскроя.

Был частично реализован графический интерфейс программы согласно скетчу: основное окно программы, окно настроек, окно ввода данных. Также был реализован генетический алгоритм для решения задачи раскроя.

На четвертой итерации был полностью реализован графический интерфейс, который взаимодействует с генетическим алгоритмом. Можно посмотреть любое промежуточное решение (особь), а также кроссинговер и мутацию особей. В процессе работы программы и, в частности, алгоритма, происходящее логируется. В процессе проверки работоспособности алгоритма багов обнаружено не было.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Панченко Т.В. Учебно-методическое пособие «Генетические алгоритмы».