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. 2304 _____ Деменев К.О.

Студентка гр. 2304 _____ Иванова М.А.

Студент гр. 2304 _____ Шумилов А.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Изучить генетические алгоритмы, научиться применять их на практике. Разработать генетический алгоритм и GUI. Создать программу, решающую задачу о неограниченном рюкзаке с помощью генетического алгоритма.

Задание. Вариант 1.

Задача о рюкзаке (1 рюкзак)

Дано N вещей, каждая i-я имеет вес W_i и стоимость C_i. Необходимо заполнить рюкзак с максимальной вместимостью по весу W_{max} вещами так, чтобы суммарная стоимость вещей в рюкзаке была максимальной. Можно класть несколько копий одной вещи в рюкзак.

Распределение ролей в команде.

- Деменев К.О. – разработка и реализация GUI;
- Иванова М.А. – написание отчета, частичная реализация алгоритма;
- Шумилов А.В. – организация работы в команде, разработка структуры проекта, частичная реализация алгоритма.

Генетический алгоритм был разработан совместно.

Выполнение работы.

Генетический алгоритм

Способы представления генома, отбора родителей и особей в следующее поколение, скрещивания и мутации, а также работа генетического алгоритма были описаны в предыдущем отчете.

К текущей итерации была снова изменена система *штрафов*. Теперь помимо мягкого штрафа существует жесткий штраф. Таким образом, если суммарный вес вещей не превышает максимально допустимого, то функция приспособленности равняется суммарной стоимостей вещей. Если перевес есть, но он меньше самое тяжелой вещи, суммарная стоимость вещей помножается на коэффициент, который вычисляется как $1 - \frac{\text{перевес}}{\max \text{допустимый вес}}$. Если перевес превышает вес самой тяжелой вещи, которая может быть добавлена в рюкзак, то функция приспособленности равняется нулю. Таким образом, значения с большим перевесом штрафуются сильнее, при этом совсем неудачные решения отбрасываются.

Кроме того, были добавлены новые способы отбора родителей: рулетка и инбридинг; способы скрещивания: дискретная и промежуточная рекомбинация; способы мутации: мутация перестановкой, мутация случайной заменой; способы отбора особей в следующее поколение: отбор вытеснением и отбор усечением.

Организация кода

Была изменена структура кода: теперь для каждой составляющей работы генетического алгоритма был создан интерфейс и его реализация.

- Класс-интерфейс отбора родителей *ParentSelectionStrategy*с абстрактным методом *selectParent,* который принимает поколение, параметры работы алгоритма и возвращает список отобранных в родители особей.

  Существуют следующие классы-реализации данного интерфейса:

- *TournamentSelection* – турнирный обор;
- *RouletteSelection* – отбор рулеткой;
- *InbreedingSelection* – инбридинг.

- Класс-интерфейс скрещивания *CrossingStrategy* с абстрактным методом *crossing,* который принимает список отобранных в родители особей, параметры работы алгоритма и возвращает список полученных детей.

  Существуют следующие классы-реализации данного интерфейса:
  - *UniformCrossing* – равномерное скрещивание;
  - *DiscreteRecombination* – дискретная рекомбинация;
  - *IntermediateRecombination* – промежуточная рекомбинация
- Класс-интерфейс мутации *MutationStrategy* с абстрактным методом *mutation,* который принимает список детей, параметры работы алгоритма и меняет (мутирует) некоторые детские особи.

  Существуют следующие классы-реализации данного интерфейса:
  - *DensityMutation* – плотность мутации;
  - *PermutationMutation* – мутация перестановкой;
  - *ExchangeMutation* – мутация заменой.
- Класс-интерфейс отбора в следующее поколение *GenerationSelectionStrategy* с абстрактным методом *select,* который принимает старое поколение, список полученных детей, параметры работы алгоритма и возвращает новое поколение особей.

  Существуют следующие классы-реализации данного интерфейса:
  - *EliteSelection* – элитарный обор;
  - *TruncationSelection* – отбор усечением;
  - *ExclusionSelection* – отбор вытеснением.

Исходный код программы расположен в Приложении А.

Тестирование расположено в Приложении Б.

Реализация графического интерфейса

Структура GUI была незначительно дополнена: название кнопки, отвечающей за сохранение параметров работы алгоритма, было исправлено на «изменить». Также были добавлены списки для выбора параметров отбора родителей, скрещивания, мутации и отбора в следующее поколение (рис. 1-2)

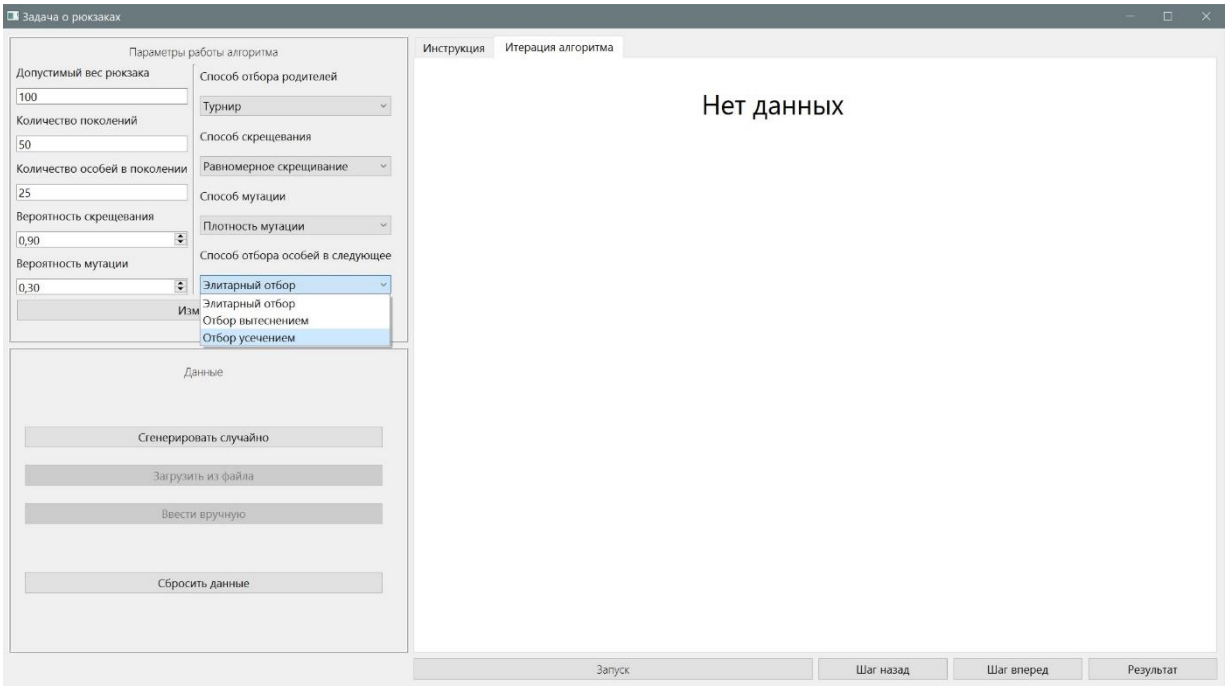

Рисунок 1 - Окно при выборе параметров работы алгоритма

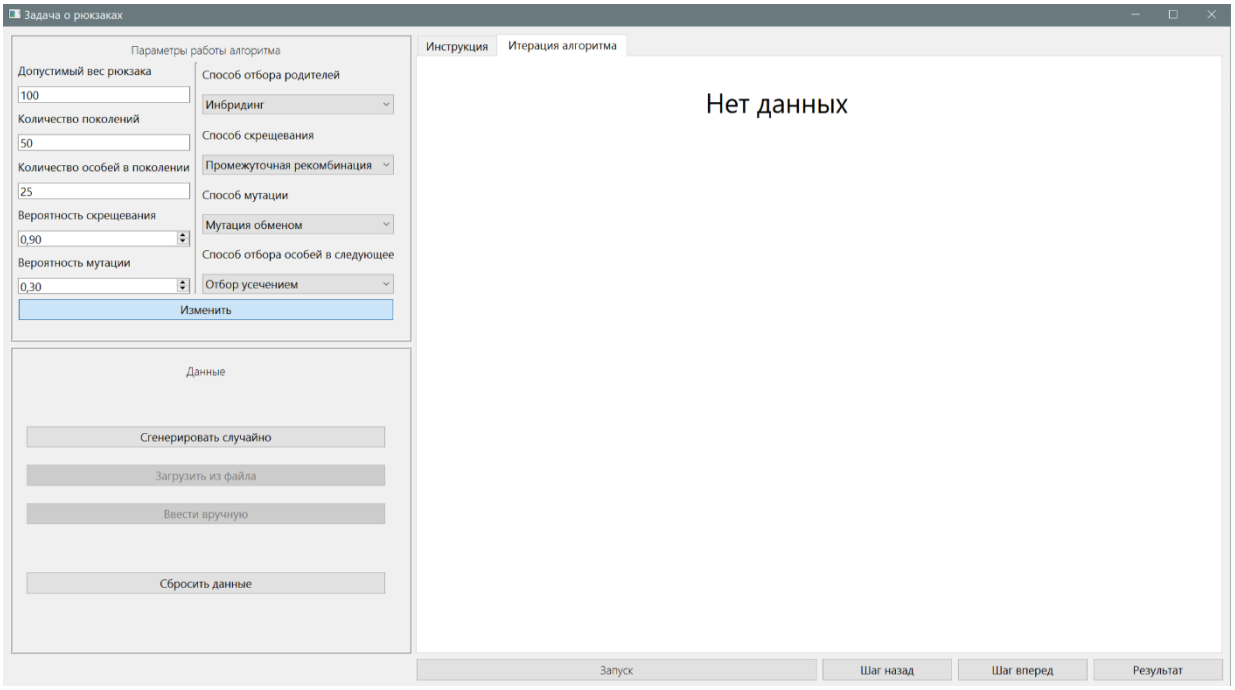

Рисунок 2 - Окно при сохранении выбранных параметров работы алгоритма

Вывод.

Изучены генетические алгоритмы. Разработан генетический алгоритм и GUI. Создана программа, решающая задачу о неограниченном рюкзаке с помощью генетического алгоритма.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src/libs/GA_methods/crossing.py

```
import random
from abc import ABC, abstractmethod

from src.libs.algorithm_parameters import *


class CrossingStrategy(ABC):
    @abstractmethod
    def crossing(self, selectedParents: list[Backpack],
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        pass


class UniformCrossing(CrossingStrategy):
    def crossingForTwoParents(self, parents: list[Backpack, Backpack])
-> list[Backpack]:
        children = [[], []]
        for j in range(len(parents[0].genome)):
            i = random.choice([0, 1])
            children[0].append(parents[i].genome[j])
            children[1].append(parents[1 - i].genome[j])
        return list(map(Backpack, children))

    def crossing(self, selectedParents: list[Backpack],
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        producedChildren = []
        while len(producedChildren) <
algorithmParameters.amountOfIndividsPerGeneration:
            parents = random.sample(selectedParents, 2)
            global generationNum
            global discreteRecomb
            if log and len(producedChildren) < 2 and generationNum ==
1 and discreteRecomb:
                if discreteRecomb:
                    print("\nДИСКРЕТНАЯ РЕКОМБИНАЦИЯ")
                else:
                    print("\nРАВНОМЕРНОЕ СКРЕЩИВАНИЕ")
                print(f"Два выбранных родителя:")
                print(f"\t1) {parents[0]}")
                print(f"\t2) {parents[1]}")
            if random.random() <
algorithmParameters.crossingProbability:
                producedChildren +=
self.crossingForTwoParents(parents)
                if log and len(producedChildren) == 2 and
generationNum == 1 and discreteRecomb:
                    if discreteRecomb:
                        print(f"Полученный ребенок:")
                        print(f"\t{producedChildren[-1]}")
                    else:
                        print(f"Полученные дети:")
                        print(f"\t1) {producedChildren[-1]}")
```

```
                        print(f"\t1) {producedChildren[-2]}")
            else:
                if log and not len(producedChildren) and generationNum
== 1 and discreteRecomb:
                    print("Скрещивание не проводится")
        return producedChildren


class DiscreteRecombination(CrossingStrategy):
    def crossing(self, selectedParents: list[Backpack],
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        producedChildren = []
        global discreteRecomb
        for i in range(2):
            children = UniformCrossing().crossing(selectedParents,
algorithmParameters)
            producedChildren += [children[j] for j in range(0,
len(children), 2)]
            discreteRecomb = 0
        return producedChildren


class IntermediateRecombination(CrossingStrategy):
    def crossingForTwoParents(self, parents: list[Backpack, Backpack])
-> Backpack:
        child = []
        for i in range(len(parents[0].genome)):
            parameter = random.uniform(-0.25, 1.25)
            child.append(int(parents[0].genome[i] + parameter *
(parents[1].genome[i] - parents[0].genome[i])))
            if child[i] < 0:
                child[i] = 0

        child = Backpack(child)
        return child

    def crossing(self, selectedParents: list[Backpack],
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        producedChildren = []
        while len(producedChildren) <
algorithmParameters.amountOfIndividsPerGeneration:
            parents = random.sample(selectedParents, 2)
            global generationNum
            if log and len(producedChildren) < 1 and generationNum ==
1:
                print("\nПРОМЕЖУТОЧНАЯ РЕКОМБИНАЦИЯ")
                print(f"Два выбранных родителя:")
                print(f"\t1) {parents[0]}")
                print(f"\t2) {parents[1]}")
            if random.random() <
algorithmParameters.crossingProbability:

producedChildren.append(self.crossingForTwoParents(parents))
                if log and len(producedChildren) == 1 and
generationNum == 1:
                    print(f"Полученный ребенок:")
                    print(f"\t{producedChildren[-1]}")
```

```
            else:
                if log and not len(producedChildren) and generationNum
== 1:
                    print("Скрещивание не проводится")
        return producedChildren
```

Название файла: src/libs/GA_methods/generation_selection.py

```
import random
from abc import ABC, abstractmethod

from src.libs.algorithm_parameters import *


class GenerationSelectionStrategy(ABC):
    @abstractmethod
    def select(self, oldGeneration: Generation, producedChildren:
list[Backpack],
               algorithmParameters: AlgorithmParameters) ->
Generation:
        pass

    def outputBackpacks(self, backpacks: list[Backpack],
algorithmParameters: AlgorithmParameters) -> None:
        for i, backpack in enumerate(backpacks):
            print(f"{i + 1}) {backpack.genome}")
            print(f"\tСуммарная стоимость вещей: {backpack.cost}")
            print(
                f"\tСуммарный вес вещей: {backpack.weight}, дельта =
{algorithmParameters.maxBackpackWeight - backpack.weight}")


class EliteSelection(GenerationSelectionStrategy):
    def select(self, oldGeneration: Generation, producedChildren:
list[Backpack],
               algorithmParameters: AlgorithmParameters) ->
Generation:
        allCandidates = oldGeneration.backpacks + producedChildren
        generation = sorted(allCandidates, key=lambda x: x.cost,
reverse=True)[
                     :int(0.1 *
algorithmParameters.amountOfIndividsPerGeneration)]
        global generationNum
        if log and generationNum == 1:
            print(f"\nЭЛИТАРНЫЙ ОТБОР")
            print(f"Лучшие 10% родительских и детских особей:")
            self.outputBackpacks(generation, algorithmParameters)
        while len(generation) !=
algorithmParameters.amountOfIndividsPerGeneration:
            generation.append(random.choice(allCandidates))
        if log and generationNum == 1:
            print(f"Остальные 90% выбираются случайно")
            print(f"\nИтоговое новое поколение:")
            self.outputBackpacks(generation, algorithmParameters)
            print()
        return Generation(generation)
```

```
class TruncationSelection(GenerationSelectionStrategy):
    def select(self, oldGeneration: Generation, producedChildren:
list[Backpack],
               algorithmParameters: AlgorithmParameters) ->
Generation:
        sortedOldGeneration = sorted(oldGeneration.backpacks +
producedChildren, key=lambda x: x.cost, reverse=True)[
                              :int(0.5 *
algorithmParameters.amountOfIndividsPerGeneration)]
        newGeneration = []
        while len(newGeneration) !=
algorithmParameters.amountOfIndividsPerGeneration:
            newGeneration.append(random.choice(sortedOldGeneration))

        global generationNum
        if log and generationNum == 1:
            print(f"\nОТБОР УСЕЧЕНИЕМ")
            print(f"Выбираем среди 50% лучших родительских и детских
особей:")
            self.outputBackpacks(sortedOldGeneration,
algorithmParameters)
            print(f"\nИтоговое новое поколение:")
            self.outputBackpacks(newGeneration, algorithmParameters)
            print()

        return Generation(newGeneration)


class ExclusionSelection(GenerationSelectionStrategy):
    def select(self, oldGeneration: Generation, producedChildren:
list[Backpack],
               algorithmParameters: AlgorithmParameters) ->
Generation:
        sortedOldGeneration = sorted(oldGeneration.backpacks +
producedChildren, key=lambda x: x.cost, reverse=True)
        newGeneration = []
        addedToNewGenerationGenomes = []

        for individ in sortedOldGeneration:
            if len(newGeneration) ==
algorithmParameters.amountOfIndividsPerGeneration:
                break
            if individ.genome not in addedToNewGenerationGenomes:
                newGeneration.append(individ)
                addedToNewGenerationGenomes.append(individ.genome)

        # if len(newGeneration) !=
algorithmParameters.amountOfIndividsPerGeneration:
        #     print("ПУПУПУ")
        #     exit(0)

        global generationNum
        if log and generationNum == 1:
            print(f"\nОТБОР ВЫТЕСНЕНИЕМ")
            print(f"Выбираем в новое поколение лучшие уникальные
родительские и детские особи")
```

```
            print(f"\nИтоговое новое поколение:")
            self.outputBackpacks(newGeneration, algorithmParameters)
            print()

        return Generation(newGeneration)
```

Название файла: src/libs/GA_methods/mutation.py

```
import random
from abc import ABC, abstractmethod

from src.libs.algorithm_parameters import *


class MutationStrategy(ABC):
    @abstractmethod
    def mutation(self, children: list[Backpack], algorithmParameters:
AlgorithmParameters, items: list[Item]) -> None:
        pass


class DensityMutation(MutationStrategy):
    def mutationOneChild(self, child: Backpack, algorithmParameters:
AlgorithmParameters) -> None:
        global mutationNum
        if log and mutationNum == 1:
            print("\nПЛОТНОСТЬ МУТАЦИИ")
            print(f"Геном до мутации:")
            print(f"{child}")

        parameter = 20
        for i in range(len(child.genome)):
            if log and i == mutationNum == 1:
                print(f"\tПервый ген до мутации: {child.genome[i]}")
            if random.random() <
algorithmParameters.mutationProbability * 1.25:
                delta = 0
                for j in range(parameter):
                    randVal = random.choices([1, 0], weights=[1 /
parameter, 1 - 1 / parameter])[0]
                    delta += randVal * 2 ** (-i)
                sign = random.choice([-1, 1])
                child.genome[i] = int(child.genome[i] + sign * delta *
2)
                if child.genome[i] < 0:
                    child.genome[i] = 0
                if log and i == mutationNum == 1:
                    print(f"\tСлучайно полученное значение, на которое
мутирует ген: {int(2 * delta)}")
                    print(f"\tЗнак мутации: {sign}")
                    print(f"\tПервый ген после мутации:
{child.genome[i]}")
            else:
                if log and i == mutationNum == 1:
                    print(f"\tПервый ген не мутирует")
        if log and mutationNum == 1:
            print(f"Геном после мутации:")
```

```
            print(f"{child}")
        mutationNum = 2

    def mutation(self, children: list[Backpack], algorithmParameters:
AlgorithmParameters, items: list[Item]) -> None:
        for i in range(len(children)):
            if random.random() <
algorithmParameters.mutationProbability:
                self.mutationOneChild(children[i],
algorithmParameters)


class PermutationMutation(MutationStrategy):
    def mutation(self, children: list[Backpack], algorithmParameters:
AlgorithmParameters, items: list[Item]) -> None:
        global mutationNum
        for i in range(len(children)):
            if random.random() <
algorithmParameters.mutationProbability:
                if log and mutationNum == 1:
                    print("\nМУТАЦИЯ ПЕРЕСТАНОВКОЙ")
                    print(f"Геном до мутации:")
                    print(f"{children[i]}")
                num_of_recomb = random.randint(1, int(len(items) *
algorithmParameters.mutationProbability + 1))
                if log and mutationNum == 1:
                    print(f"Количество перестановок: {num_of_recomb}")
                for j in range(num_of_recomb):
                    ind1, ind2 = random.sample(range(len(items)), 2)
                    children[i].genome[ind1], children[i].genome[ind2]
= (
                        children[i].genome[ind2],
children[i].genome[ind1])
                    if log and mutationNum == 1 and j == 0:
                        print(f"\tСлучайно выбранные индексы генов для
первой перестановки: {ind1}, {ind2}")
                        print(f"\tГеном после первой перестановки:
{children[i]}")
                if log and mutationNum == 1:
                    print(f"Геном после мутации:")
                    print(f"{children[i]}")
                mutationNum = 2


class ExchangeMutation(MutationStrategy):
    def mutation(self, children: list[Backpack], algorithmParameters:
AlgorithmParameters, items: list[Item]) -> None:
        global mutationNum
        for i in range(len(children)):
            if random.random() <
algorithmParameters.mutationProbability:
                if log and mutationNum == 1:
                    print("\nМУТАЦИЯ СЛУЧАЙНОЙ ЗАМЕНОЙ")
                    print(f"Геном до мутации:")
                    print(f"{children[i]}")
                numOfChanges = random.randint(1,
```

```
                                                int(len(children[i]) *
algorithmParameters.mutationProbability + 1))
                if log and mutationNum == 1:
                    print(f"Количество замен: {numOfChanges}")
                for j in range(numOfChanges):
                    i = random.choice(range(len(children[i])))
                    value = random.randint(0,
algorithmParameters.maxBackpackWeight // items[i].cost)
                    children[i].genome[i] = value
                    if log and mutationNum == 1 and j == 0:
                        print(f"\tСлучайно выбранный индекс гена для
первой замены: {i}")
                        print(f"\tНовое значение гена для первой
замены: {value}")
                        print(f"\tГеном после первой замены:
{children[i]}")
                if log and mutationNum == 1:
                    print(f"Геном после мутации:")
                    print(f"{children[i]}")
                mutationNum = 2
```

Название файла: src/libs/GA_methods/parent_selection.py

```
import random
from abc import ABC, abstractmethod

from src.libs.algorithm_parameters import *


class ParentSelectionStrategy(ABC):
    @abstractmethod
    def selectParent(self, generation: Generation,
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        pass


class TournamentSelection(ParentSelectionStrategy):
    def selectParent(self, generation: Generation,
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        selectedParents = []
        while len(selectedParents) !=
algorithmParameters.amountOfIndividsPerGeneration:
            indexes = [i for i in range(len(generation))]
            tournamentIndexes = random.sample(indexes, 2)
            selectedParents.append(max([generation[i] for i in
tournamentIndexes]))

            global generationNum
            if log and len(selectedParents) < 2 and generationNum ==
1:
                print("\nОТБОР ТУРНИРОМ")
                individ1 = generation[tournamentIndexes[0]]
                individ2 = generation[tournamentIndexes[1]]
                print(f"Две случайно выбранные особи:")
                print(f"\t1) {individ1}")
                print(f"\t2) {individ2}")
                print(f"\tВыбираем лучшую из них: {selectedParents[-
1].genome}")
```

```
        return selectedParents


class RouletteSelection(ParentSelectionStrategy):
    def selectParent(self, generation: Generation,
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        selectedParents = []
        sumFitness = sum([individ.cost for individ in generation])
        probabilities = [individ.cost / sumFitness for individ in
generation]

        table = PrettyTable(['№', 'Особь', 'Приспособленность',
'Вероятность выбора'])
        for i in range(len(generation)):
            table.add_row([i, generation[i].genome,
generation[i].cost, round(probabilities[i], 4)])

        while len(selectedParents) !=
algorithmParameters.amountOfIndividsPerGeneration:
            selectedParents.append(random.choices(generation,
weights=probabilities, k=1)[0])

            global generationNum
            if log and len(selectedParents) < 2 and generationNum ==
1:
                print("\nОТБОР РУЛЕТКОЙ")
                print(table)
                print(f"Случайно выбранная особь:")
                print(f"\t{selectedParents[0]}")

        return selectedParents


class InbreedingSelection(ParentSelectionStrategy):
    def selectTwoParents(self, generation: Generation) ->
list[Backpack]:
        firstParentInd = random.choice([i for i in
range(len(generation))])
        selectedParents =
[generation.descendingSortedBackpacks[firstParentInd]]
        if firstParentInd == 0:
            secondParentInd = firstParentInd + 1
        elif (firstParentInd == len(generation) - 1 or
              (abs(generation.descendingSortedBackpacks[firstParentInd
- 1].cost -

generation.descendingSortedBackpacks[firstParentInd].cost) <
               abs(generation.descendingSortedBackpacks[firstParentInd
+ 1].cost -

generation.descendingSortedBackpacks[firstParentInd].cost))):
            secondParentInd = firstParentInd - 1
        else:
            secondParentInd = firstParentInd + 1

selectedParents.append(generation.descendingSortedBackpacks[secondPare
ntInd])
```

```
        global generationNum
        if log and len(selectedParents) < 3 and generationNum == 1:
            print("\nИНБРИДИНГ")
            print(f"Случайно выбранная особь:")

print(f"\t{generation.descendingSortedBackpacks[firstParentInd]}")
            print(f"\tЕе порядковый номер в популяции по убыванию ф-ии
приспособленности: {firstParentInd}")
            print(f"Ближайшая особь:")

print(f"\t{generation.descendingSortedBackpacks[secondParentInd]}")
            print(
                f"\tЕе порядковый номер в популяции по убыванию ф-ии
приспособленности: {secondParentInd if secondParentInd >= 0 else
len(generation) - secondParentInd}")
        return selectedParents

    def selectParent(self, generation: Generation,
algorithmParameters: AlgorithmParameters) -> list[Backpack]:
        selectedParents = []
        while len(selectedParents) <
algorithmParameters.amountOfIndividsPerGeneration:
            selectedParents += self.selectTwoParents(generation)
        return selectedParents
```

Название файла: src/libs/algorithm_parameters.py

```
from src.libs.objects import *
from prettytable import PrettyTable


generationNum = 1
mutationNum = 1
discreteRecomb = 1
log = 0


class AlgorithmParameters:
    def __init__(self,
                 maxBackpackWeight: int,
                 crossingProbability: float,
                 mutationProbability: float,
                 amountOfIndividsPerGeneration: int,
                 maxAmountOfGenerations: int,
                 parentsSelectionStrategy: 'ParentSelectionStrategy',
                 crossingStrategy: 'CrossingStrategy',
                 mutationStrategy: 'MutationStrategy',
                 generationSelectionStrategy:
'GenerationSelectionStrategy'):
        self.maxBackpackWeight = maxBackpackWeight
        self.crossingProbability = crossingProbability
        self.mutationProbability = mutationProbability
        self.amountOfIndividsPerGeneration =
amountOfIndividsPerGeneration
        self.maxAmountOfGenerations = maxAmountOfGenerations
        self.parentsSelectionStrategy = parentsSelectionStrategy
        self.crossingStrategy = crossingStrategy
```

```
        self.mutationStrategy = mutationStrategy
        self.generationSelectionStrategy = generationSelectionStrategy
```

Название файла: src/libs/genetic_algorithm.py

```
import matplotlib.pyplot as plt
import numpy as np

from src.libs.GA_methods.crossing import *
from src.libs.GA_methods.generation_selection import *
from src.libs.GA_methods.mutation import *
from src.libs.GA_methods.parent_selection import *


class GeneticAlgorithm:
    def __init__(self, items: list[Item], algorithmParameters:
AlgorithmParameters):
        self.items = items
        self.algorithmParameters = algorithmParameters
        self.parentsSelectionStrategy =
algorithmParameters.parentsSelectionStrategy
        self.crossingStrategy = algorithmParameters.crossingStrategy
        self.mutationStrategy = algorithmParameters.mutationStrategy
        self.generationSelectionStrategy =
algorithmParameters.generationSelectionStrategy

    def generateRandomBackpack(self) -> Backpack:
        remainingWeight = self.algorithmParameters.maxBackpackWeight
        availableItems = [i for i in range(len(self.items)) if
                          self.items[i].weight <= remainingWeight]
        genome = [0] * len(self.items)
        while availableItems and remainingWeight:
            itemIndex = random.choice(availableItems)
            item = self.items[itemIndex]
            maxAmount = remainingWeight // item.weight
            amount = maxAmount if (len(availableItems) == 1 or
maxAmount == 1) \
                else random.randint(1, maxAmount)

            genome[itemIndex] += amount
            remainingWeight -= amount * item.weight

            availableItems = [i for i in availableItems if
self.items[i].weight <= remainingWeight]
        return Backpack(genome)

    def generateRandomGeneration(self) -> Generation:
        randomGeneration = Generation([])
        for _ in
range(self.algorithmParameters.amountOfIndividsPerGeneration):
            backpack = self.generateRandomBackpack()
            backpack.calculateWeight(self.items)

backpack.calculateFitness(self.algorithmParameters.maxBackpackWeight,
self.items)
            randomGeneration.append(backpack)
        return randomGeneration
```

```
    def getSolution(self) -> list[IterationInfo]:
        generation = self.generateRandomGeneration()
        if log:
            print(f"Начальное случайно сгенерированное поколение:")
            self.outputBackpacks(generation.backpacks)
            print()

        maxFitness = []
        averageFitness = []
        allIterations = []
        global generationNum
        for generationNumber in range(1,
self.algorithmParameters.maxAmountOfGenerations + 1):
            generationNum = generationNumber
            generation.calculateWeight(self.items)

generation.calculateFitness(self.algorithmParameters.maxBackpackWeight
, self.items)
            generation.sortBackpacksInDescendingOrder()
            maxFitness.append(generation.getMaxFitness())
            averageFitness.append(generation.getAverageFitness())

allIterations.append(IterationInfo(generation.getBestBackpacks(),
maxFitness[-1], averageFitness[-1]))

            if log:
                print(f"\n------------------")
                print(f"Лучшие решения поколения №{generationNumber}")
                self.outputBackpacks(generation.getBestBackpacks())
                print(f"Текущая максимальная приспособленность:
{generation.getMaxFitness()}")
                print(f"Текущая средняя приспособленность:
{generation.getAverageFitness()}")

            selectedParents =
self.parentsSelectionStrategy.selectParent(generation,
self.algorithmParameters)
            producedChildren =
self.crossingStrategy.crossing(selectedParents,
self.algorithmParameters)
            self.mutationStrategy.mutation(producedChildren,
self.algorithmParameters, self.items)
            generation =
self.generationSelectionStrategy.select(generation, producedChildren,

self.algorithmParameters)
        if log:
            self.drawPlot(maxFitness, averageFitness)
        return allIterations

    def drawPlot(self, maxFitness: list[int], averageFitness:
list[float]) -> None:
        x_len = self.algorithmParameters.maxAmountOfGenerations
        plt.plot(list(range(x_len)), averageFitness, 'r-')
        plt.plot(list(range(x_len)), maxFitness, 'b-')
        plt.grid()
```

```
        plt.xticks(np.arange(0, x_len + 1, 2))
        plt.xlabel('Поколение')
        plt.ylabel('Приспособленность')
        plt.show()

    def outputGenerationInfo(self, generation: Generation,
generationNumber: int) -> None:
        print(f"\nПоколение №{generationNumber}:")
        for i, solution in
enumerate(generation.descendingSortedBackpacks):
            print(f"{i + 1}) {solution.genome}")
            print(f"\tСуммарная стоимость вещей: {solution.cost}")
            print(
                f"\tСуммарный вес вещей: {solution.weight}, дельта =
{self.algorithmParameters.maxBackpackWeight - solution.weight}")
        print(f"Текущая максимальная приспособленность:
{generation.getMaxFitness()}")
        print(f"Текущая средняя приспособленность:
{generation.getAverageFitness()}")

    def outputBackpacks(self, backpacks: list[Backpack]) -> None:
        for i, backpack in enumerate(backpacks):
            print(f"{i + 1}) {backpack.genome}")
            print(f"\tСуммарная стоимость вещей: {backpack.cost}")
            print(
                f"\tСуммарный вес вещей: {backpack.weight}, дельта =
{self.algorithmParameters.maxBackpackWeight - backpack.weight}")


if __name__ == '__main__':
    items = [Item(5, 2), Item(7, 3), Item(6, 4), Item(3, 2)]
    maxBackpackWeight = 22
    crossingProbability = 0.9
    mutationProbability = 0.2
    amountOfIndividsPerGeneration = 20
    maxAmountOfGenerations = 20

    parentsSelectionStrategy = TournamentSelection()
    crossingStrategy = UniformCrossing()
    mutationStrategy = DensityMutation()
    generationSelectionStrategy = EliteSelection()

    algorithmParameters = AlgorithmParameters(
        maxBackpackWeight,
        crossingProbability,
        mutationProbability,
        amountOfIndividsPerGeneration,
        maxAmountOfGenerations,
        parentsSelectionStrategy,
        crossingStrategy,
        mutationStrategy,
        generationSelectionStrategy
    )

    GA = GeneticAlgorithm(items, algorithmParameters)
    GA.getSolution()
```

Название файла: src/libs/objects.py

```
from typing import Iterator


class Item:
    def __init__(self, cost: int, weight: int):
        self.cost = cost
        self.weight = weight

    def __str__(self) -> str:
        return f"Вещь стоит {self.cost} и весит {self.weight}"

    def __lt__(self, other: 'Item') -> bool:
        return self.weight < other.weight

    def __le__(self, other: 'Item') -> bool:
        return self.weight <= other.weight

    def __gt__(self, other: 'Item') -> bool:
        return self.weight > other.weight

    def __ge__(self, other: 'Item') -> bool:
        return self.weight >= other.weight


class Backpack:
    def __init__(self, amountOfEachItems: list[int]):
        self.genome = amountOfEachItems
        self.cost = 0
        self.weight = 0

    def __str__(self):
        return f"{self.genome}, стоимость = {self.cost}, вес =
{self.weight}"

    def __iter__(self) -> Iterator:
        return iter(self.genome)

    def __len__(self) -> int:
        return len(self.genome)

    def __le__(self, other: 'Backpack') -> bool:
        return self.cost <= other.cost

    def __lt__(self, other: 'Backpack') -> bool:
        return self.cost < other.cost

    def __ge__(self, other: 'Backpack') -> bool:
        return self.cost >= other.cost

    def __gt__(self, other: 'Backpack') -> bool:
        return self.cost > other.cost

    def calculateWeight(self, items: list[Item]) -> None:
        self.weight = sum(items[i].weight * self.genome[i] for i in
range(len(items)))
```

```
    def calculateFitness(self, limitWeight: int, items: list[Item]) ->
None:
        sumCost = sum(items[i].cost * self.genome[i] for i in
range(len(items)))
        overload = self.weight - limitWeight
        if overload <= 0:
            self.cost = sumCost
        else:
            penalty = (self.weight - limitWeight) / limitWeight
            self.cost = int(sumCost * (1 - penalty)) if overload <=
max(item.weight for item in items) else 0


class Generation:
    def __init__(self, backpacks: list[Backpack]):
        self.backpacks = backpacks
        self.descendingSortedBackpacks = sorted(backpacks, key=lambda
x: x.cost, reverse=True)

    def __str__(self):
        return "\n".join(map(str, self.backpacks))

    def __iter__(self) -> Iterator:
        return iter(self.backpacks)

    def __len__(self) -> int:
        return len(self.backpacks)

    def __getitem__(self, key: int) -> Backpack:
        return self.backpacks[key]

    def append(self, item: Backpack) -> None:
        self.backpacks.append(item)

    def expend(self, other: 'Generation') -> None:
        self.backpacks.extend(other)

    def remove(self, item: Backpack) -> None:
        self.backpacks.remove(item)

    def getBestBackpacks(self) -> list[Backpack]:
        sorted_backpacks = sorted(self.backpacks, key=lambda x:
x.cost, reverse=True)
        return sorted_backpacks[:3]

    def getAverageFitness(self) -> float:
        return sum(backpack.cost for backpack in self.backpacks) /
len(self.backpacks)

    def getMaxFitness(self) -> int:
        return self.getBestBackpacks()[0].cost

    def calculateWeight(self, items: list[Item]) -> None:
        for backpack in self.backpacks:
            backpack.calculateWeight(items)
```

```
    def calculateFitness(self, limitWeight: int, items: list[Item]) ->
None:
        for backpack in self.backpacks:
            backpack.calculateFitness(limitWeight, items)

    def sortBackpacksInDescendingOrder(self) -> None:
        self.descendingSortedBackpacks = sorted(self.backpacks,
key=lambda x: x.cost, reverse=True)


class IterationInfo:
    def __init__(self, bestBackpacks: list[Backpack],
currentMaxFitness: int, currentAverageFitness: float):
        self.bestBackpacks = bestBackpacks
        self.currentMaxFitness = currentMaxFitness
        self.currentAverageFitness = currentAverageFitness
```

Название файла: src/UIs/UIHandInputUI.py

```
# Form implementation generated from reading ui file
'handInputDialogUI.ui'
#
# Created by: PyQt6 UI code generator 6.7.0
#
# WARNING: Any manual changes made to this file will be lost when
pyuic6 is
# run again.  Do not edit this file unless you know what you are
doing.


from PyQt6 import QtCore, QtGui, QtWidgets


class Ui_HandInputDialog(object):
    def setupUi(self, HandInputDialog):
        HandInputDialog.setObjectName("HandInputDialog")
        HandInputDialog.resize(280, 391)
        HandInputDialog.setSizeGripEnabled(False)
        HandInputDialog.setModal(False)
        self.gridLayout = QtWidgets.QGridLayout(HandInputDialog)
        self.gridLayout.setObjectName("gridLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.cancelButton =
QtWidgets.QPushButton(parent=HandInputDialog)
        self.cancelButton.setObjectName("cancelButton")
        self.horizontalLayout.addWidget(self.cancelButton)
        self.doneButton =
QtWidgets.QPushButton(parent=HandInputDialog)
        self.doneButton.setObjectName("doneButton")
        self.horizontalLayout.addWidget(self.doneButton)
        self.gridLayout.addLayout(self.horizontalLayout, 2, 0, 1, 1)
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.amountLabel = QtWidgets.QLabel(parent=HandInputDialog)

self.amountLabel.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.amountLabel.setObjectName("amountLabel")
```

```
        self.verticalLayout.addWidget(self.amountLabel)
        self.AmountLineEdit =
QtWidgets.QLineEdit(parent=HandInputDialog)
        self.AmountLineEdit.setObjectName("AmountLineEdit")
        self.verticalLayout.addWidget(self.AmountLineEdit)
        self.gridLayout.addLayout(self.verticalLayout, 0, 0, 1, 1)
        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.label_2 = QtWidgets.QLabel(parent=HandInputDialog)
        self.label_2.setObjectName("label_2")
        self.verticalLayout_2.addWidget(self.label_2)
        self.tableWidget =
QtWidgets.QTableWidget(parent=HandInputDialog)
        self.tableWidget.setGridStyle(QtCore.Qt.PenStyle.SolidLine)
        self.tableWidget.setRowCount(0)
        self.tableWidget.setColumnCount(2)
        self.tableWidget.setObjectName("tableWidget")
        item = QtWidgets.QTableWidgetItem()
        self.tableWidget.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.tableWidget.setHorizontalHeaderItem(1, item)
        self.verticalLayout_2.addWidget(self.tableWidget)
        self.gridLayout.addLayout(self.verticalLayout_2, 1, 0, 1, 1)

        self.retranslateUi(HandInputDialog)
        QtCore.QMetaObject.connectSlotsByName(HandInputDialog)

    def retranslateUi(self, HandInputDialog):
        _translate = QtCore.QCoreApplication.translate
        HandInputDialog.setWindowTitle(_translate("HandInputDialog",
"Ручной ввод"))
        self.cancelButton.setText(_translate("HandInputDialog",
"Назад"))
        self.doneButton.setText(_translate("HandInputDialog",
"Готово"))
        self.amountLabel.setText(_translate("HandInputDialog",
"Введите количество предметов"))

self.AmountLineEdit.setPlaceholderText(_translate("HandInputDialog",
"Кол-во предметов"))
        self.label_2.setText(_translate("HandInputDialog", "Введите
соответсвующую информацию"))
        self.tableWidget.setSortingEnabled(True)
        item = self.tableWidget.horizontalHeaderItem(0)
        item.setText(_translate("HandInputDialog", "Вес"))
        item = self.tableWidget.horizontalHeaderItem(1)
        item.setText(_translate("HandInputDialog", "Цена"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    HandInputDialog = QtWidgets.QDialog()
    ui = Ui_HandInputDialog()
    ui.setupUi(HandInputDialog)
    HandInputDialog.show()
    sys.exit(app.exec())
```

Название файла: src/UIs/UIMainWindow.py

```
# Form implementation generated from reading ui file
'backpackProblemUI.ui'
#
# Created by: PyQt6 UI code generator 6.7.0
#
# WARNING: Any manual changes made to this file will be lost when
pyuic6 is
# run again.  Do not edit this file unless you know what you are
doing.


from PyQt6 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1506, 810)
        self.wind = QtWidgets.QWidget(parent=MainWindow)
        self.wind.setEnabled(True)
        self.wind.setObjectName("wind")
        self.gridLayout = QtWidgets.QGridLayout(self.wind)
        self.gridLayout.setObjectName("gridLayout")
        self.verticalLayout_4 = QtWidgets.QVBoxLayout()
        self.verticalLayout_4.setObjectName("verticalLayout_4")
        self.gridLayout.addLayout(self.verticalLayout_4, 2, 0, 1, 1)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.backButton = QtWidgets.QPushButton(parent=self.wind)
        self.backButton.setObjectName("backButton")
        self.horizontalLayout.addWidget(self.backButton)
        self.forwardButton = QtWidgets.QPushButton(parent=self.wind)
        self.forwardButton.setObjectName("forwardButton")
        self.horizontalLayout.addWidget(self.forwardButton)
        self.resultButton = QtWidgets.QPushButton(parent=self.wind)
        self.resultButton.setObjectName("resultButton")
        self.horizontalLayout.addWidget(self.resultButton)
        self.gridLayout.addLayout(self.horizontalLayout, 2, 2, 1, 1)
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.paramsDataFrame = QtWidgets.QFrame(parent=self.wind)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Policy.Minimum,
QtWidgets.QSizePolicy.Policy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.paramsDataFrame.sizePolicy().hasHeig
htForWidth())
        self.paramsDataFrame.setSizePolicy(sizePolicy)
        self.paramsDataFrame.setMinimumSize(QtCore.QSize(100, 0))
        self.paramsDataFrame.setFrameShape(QtWidgets.QFrame.Shape.Box)

self.paramsDataFrame.setFrameShadow(QtWidgets.QFrame.Shadow.Raised)
        self.paramsDataFrame.setObjectName("paramsDataFrame")
```

```
        self.verticalLayoutWidget_9 =
QtWidgets.QWidget(parent=self.paramsDataFrame)
        self.verticalLayoutWidget_9.setGeometry(QtCore.QRect(0, 0,
481, 361))

self.verticalLayoutWidget_9.setObjectName("verticalLayoutWidget_9")
        self.verticalLayout_9 =
QtWidgets.QVBoxLayout(self.verticalLayoutWidget_9)
        self.verticalLayout_9.setContentsMargins(10, 10, 10, 10)
        self.verticalLayout_9.setObjectName("verticalLayout_9")
        self.ParamsLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)
        self.ParamsLabel.setEnabled(True)
        font = QtGui.QFont()
        font.setBold(True)
        font.setUnderline(False)
        font.setWeight(75)
        self.ParamsLabel.setFont(font)

self.ParamsLabel.setLayoutDirection(QtCore.Qt.LayoutDirection.LeftToRi
ght)
        self.ParamsLabel.setTextFormat(QtCore.Qt.TextFormat.AutoText)
        self.ParamsLabel.setScaledContents(False)

self.ParamsLabel.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.ParamsLabel.setObjectName("ParamsLabel")
        self.verticalLayout_9.addWidget(self.ParamsLabel)
        self.horizontalLayout_17 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_17.setObjectName("horizontalLayout_17")
        self.algParams = QtWidgets.QVBoxLayout()

self.algParams.setSizeConstraint(QtWidgets.QLayout.SizeConstraint.SetF
ixedSize)
        self.algParams.setContentsMargins(0, 0, 0, 0)
        self.algParams.setSpacing(10)
        self.algParams.setObjectName("algParams")
        self.backpackValueLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)
        self.backpackValueLabel.setObjectName("backpackValueLabel")
        self.algParams.addWidget(self.backpackValueLabel)
        self.backpackValueLE =
QtWidgets.QLineEdit(parent=self.verticalLayoutWidget_9)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Policy.Minimum,
QtWidgets.QSizePolicy.Policy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.backpackValueLE.sizePolicy().hasHeig
htForWidth())
        self.backpackValueLE.setSizePolicy(sizePolicy)
        self.backpackValueLE.setMinimumSize(QtCore.QSize(50, 0))
        self.backpackValueLE.setInputMask("")
        self.backpackValueLE.setText("")
        self.backpackValueLE.setClearButtonEnabled(False)
        self.backpackValueLE.setObjectName("backpackValueLE")
        self.algParams.addWidget(self.backpackValueLE)
```

```
        self.genAmountLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)
        self.genAmountLabel.setObjectName("genAmountLabel")
        self.algParams.addWidget(self.genAmountLabel)
        self.generationAmountLE =
QtWidgets.QLineEdit(parent=self.verticalLayoutWidget_9)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.generationAmountLE.sizePolicy().hasH
eightForWidth())
        self.generationAmountLE.setSizePolicy(sizePolicy)
        self.generationAmountLE.setObjectName("generationAmountLE")
        self.algParams.addWidget(self.generationAmountLE)
        self.entityAmountLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)
        self.entityAmountLabel.setObjectName("entityAmountLabel")
        self.algParams.addWidget(self.entityAmountLabel)
        self.entityAmountLE =
QtWidgets.QLineEdit(parent=self.verticalLayoutWidget_9)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.entityAmountLE.sizePolicy().hasHeigh
tForWidth())
        self.entityAmountLE.setSizePolicy(sizePolicy)
        self.entityAmountLE.setObjectName("entityAmountLE")
        self.algParams.addWidget(self.entityAmountLE)
        self.probabilityCrossingLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)

self.probabilityCrossingLabel.setObjectName("probabilityCrossingLabel"
)
        self.algParams.addWidget(self.probabilityCrossingLabel)
        self.crossingProbabilitySpin =
QtWidgets.QDoubleSpinBox(parent=self.verticalLayoutWidget_9)
        self.crossingProbabilitySpin.setMaximum(1.0)
        self.crossingProbabilitySpin.setSingleStep(0.1)

self.crossingProbabilitySpin.setObjectName("crossingProbabilitySpin")
        self.algParams.addWidget(self.crossingProbabilitySpin)
        self.probabilityCrossingLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)

self.probabilityCrossingLabel_2.setObjectName("probabilityCrossingLabe
l_2")
        self.algParams.addWidget(self.probabilityCrossingLabel_2)
        self.mutationProbabilitySpin =
QtWidgets.QDoubleSpinBox(parent=self.verticalLayoutWidget_9)
        self.mutationProbabilitySpin.setMaximum(1.0)
        self.mutationProbabilitySpin.setSingleStep(0.1)
```

```
self.mutationProbabilitySpin.setObjectName("mutationProbabilitySpin")
        self.algParams.addWidget(self.mutationProbabilitySpin)
        self.algParams.setStretch(0, 2)
        self.algParams.setStretch(1, 2)
        self.algParams.setStretch(2, 2)
        self.horizontalLayout_17.addLayout(self.algParams)
        self.line_2 =
QtWidgets.QFrame(parent=self.verticalLayoutWidget_9)
        self.line_2.setFrameShape(QtWidgets.QFrame.Shape.VLine)
        self.line_2.setFrameShadow(QtWidgets.QFrame.Shadow.Sunken)
        self.line_2.setObjectName("line_2")
        self.horizontalLayout_17.addWidget(self.line_2)
        self.params_layout_2 = QtWidgets.QVBoxLayout()
        self.params_layout_2.setContentsMargins(0, 0, 0, 0)
        self.params_layout_2.setSpacing(10)
        self.params_layout_2.setObjectName("params_layout_2")
        self.parent_selection_method_label =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)

self.parent_selection_method_label.setObjectName("parent_selection_met
hod_label")

self.params_layout_2.addWidget(self.parent_selection_method_label)
        self.parent_selection_comboBox =
QtWidgets.QComboBox(parent=self.verticalLayoutWidget_9)

self.parent_selection_comboBox.setObjectName("parent_selection_comboBo
x")
        self.parent_selection_comboBox.addItem("")
        self.parent_selection_comboBox.addItem("")
        self.parent_selection_comboBox.addItem("")
        self.params_layout_2.addWidget(self.parent_selection_comboBox)
        self.crossing_method_label =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)

self.crossing_method_label.setObjectName("crossing_method_label")
        self.params_layout_2.addWidget(self.crossing_method_label)
        self.crossing_method_comboBox =
QtWidgets.QComboBox(parent=self.verticalLayoutWidget_9)

self.crossing_method_comboBox.setObjectName("crossing_method_comboBox"
)
        self.crossing_method_comboBox.addItem("")
        self.crossing_method_comboBox.addItem("")
        self.crossing_method_comboBox.addItem("")
        self.params_layout_2.addWidget(self.crossing_method_comboBox)
        self.mutation_method_label =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)

self.mutation_method_label.setObjectName("mutation_method_label")
        self.params_layout_2.addWidget(self.mutation_method_label)
        self.mutation_method_comboBox =
QtWidgets.QComboBox(parent=self.verticalLayoutWidget_9)

self.mutation_method_comboBox.setObjectName("mutation_method_comboBox"
)
```

```
        self.mutation_method_comboBox.addItem("")
        self.mutation_method_comboBox.addItem("")
        self.mutation_method_comboBox.addItem("")
        self.params_layout_2.addWidget(self.mutation_method_comboBox)
        self.methodOfSelectingIndividsLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_9)

self.methodOfSelectingIndividsLabel.setObjectName("methodOfSelectingIn
dividsLabel")

self.params_layout_2.addWidget(self.methodOfSelectingIndividsLabel)
        self.methodOfSelectingIndividsComboBox =
QtWidgets.QComboBox(parent=self.verticalLayoutWidget_9)

self.methodOfSelectingIndividsComboBox.setObjectName("methodOfSelectin
gIndividsComboBox")
        self.methodOfSelectingIndividsComboBox.addItem("")
        self.methodOfSelectingIndividsComboBox.addItem("")
        self.methodOfSelectingIndividsComboBox.addItem("")

self.params_layout_2.addWidget(self.methodOfSelectingIndividsComboBox)
        self.horizontalLayout_17.addLayout(self.params_layout_2)
        self.verticalLayout_9.addLayout(self.horizontalLayout_17)
        self.saveButton =
QtWidgets.QPushButton(parent=self.verticalLayoutWidget_9)
        self.saveButton.setObjectName("saveButton")
        self.verticalLayout_9.addWidget(self.saveButton)
        self.verticalLayout.addWidget(self.paramsDataFrame)
        self.dataFrame = QtWidgets.QFrame(parent=self.wind)
        self.dataFrame.setFrameShape(QtWidgets.QFrame.Shape.Box)
        self.dataFrame.setFrameShadow(QtWidgets.QFrame.Shadow.Raised)
        self.dataFrame.setMidLineWidth(0)
        self.dataFrame.setObjectName("dataFrame")
        self.verticalLayoutWidget_2 =
QtWidgets.QWidget(parent=self.dataFrame)
        self.verticalLayoutWidget_2.setGeometry(QtCore.QRect(0, 0,
481, 361))

self.verticalLayoutWidget_2.setObjectName("verticalLayoutWidget_2")
        self.DataLayout =
QtWidgets.QVBoxLayout(self.verticalLayoutWidget_2)
        self.DataLayout.setContentsMargins(20, 20, 20, 20)
        self.DataLayout.setSpacing(20)
        self.DataLayout.setObjectName("DataLayout")
        self.DataLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_2)
        self.DataLabel.setEnabled(True)
        font = QtGui.QFont()
        font.setBold(True)
        font.setWeight(75)
        self.DataLabel.setFont(font)

self.DataLabel.setLayoutDirection(QtCore.Qt.LayoutDirection.LeftToRigh
t)
        self.DataLabel.setTextFormat(QtCore.Qt.TextFormat.AutoText)
        self.DataLabel.setScaledContents(False)
```

```
self.DataLabel.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.DataLabel.setObjectName("DataLabel")
        self.DataLayout.addWidget(self.DataLabel)
        spacerItem = QtWidgets.QSpacerItem(20, 40,
QtWidgets.QSizePolicy.Policy.Minimum,
QtWidgets.QSizePolicy.Policy.Expanding)
        self.DataLayout.addItem(spacerItem)
        self.randomGenButton =
QtWidgets.QPushButton(parent=self.verticalLayoutWidget_2)
        self.randomGenButton.setObjectName("randomGenButton")
        self.DataLayout.addWidget(self.randomGenButton)
        self.browseButton =
QtWidgets.QPushButton(parent=self.verticalLayoutWidget_2)
        self.browseButton.setObjectName("browseButton")
        self.DataLayout.addWidget(self.browseButton)
        self.inputButton =
QtWidgets.QPushButton(parent=self.verticalLayoutWidget_2)
        self.inputButton.setObjectName("inputButton")
        self.DataLayout.addWidget(self.inputButton)
        spacerItem1 = QtWidgets.QSpacerItem(20, 40,
QtWidgets.QSizePolicy.Policy.Minimum,
QtWidgets.QSizePolicy.Policy.Expanding)
        self.DataLayout.addItem(spacerItem1)
        self.resetDataButton =
QtWidgets.QPushButton(parent=self.verticalLayoutWidget_2)
        self.resetDataButton.setObjectName("resetDataButton")
        self.DataLayout.addWidget(self.resetDataButton)
        spacerItem2 = QtWidgets.QSpacerItem(20, 40,
QtWidgets.QSizePolicy.Policy.Minimum,
QtWidgets.QSizePolicy.Policy.Expanding)
        self.DataLayout.addItem(spacerItem2)
        self.verticalLayout.addWidget(self.dataFrame)
        self.gridLayout.addLayout(self.verticalLayout, 0, 0, 1, 1)
        self.iterationTabWidget_2 =
QtWidgets.QTabWidget(parent=self.wind)

self.iterationTabWidget_2.setTabPosition(QtWidgets.QTabWidget.TabPosit
ion.North)
        self.iterationTabWidget_2.setUsesScrollButtons(False)
        self.iterationTabWidget_2.setMovable(True)

self.iterationTabWidget_2.setObjectName("iterationTabWidget_2")
        self.instructionTab_2 = QtWidgets.QWidget()
        self.instructionTab_2.setObjectName("instructionTab_2")
        self.plainTextEdit =
QtWidgets.QPlainTextEdit(parent=self.instructionTab_2)
        self.plainTextEdit.setGeometry(QtCore.QRect(0, 0, 981, 721))
        font = QtGui.QFont()
        font.setPointSize(17)
        self.plainTextEdit.setFont(font)
        self.plainTextEdit.setReadOnly(True)
        self.plainTextEdit.setObjectName("plainTextEdit")
        self.iterationTabWidget_2.addTab(self.instructionTab_2, "")
        self.iterationTab_2 = QtWidgets.QWidget()
        self.iterationTab_2.setObjectName("iterationTab_2")
```

```
        self.noDataLabel =
QtWidgets.QLabel(parent=self.iterationTab_2)
        self.noDataLabel.setGeometry(QtCore.QRect(290, 20, 301, 71))
        font = QtGui.QFont()
        font.setPointSize(25)
        self.noDataLabel.setFont(font)

self.noDataLabel.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.noDataLabel.setObjectName("noDataLabel")
        self.iterationDataFrame =
QtWidgets.QFrame(parent=self.iterationTab_2)
        self.iterationDataFrame.setGeometry(QtCore.QRect(0, 0, 851,
721))

self.iterationDataFrame.setFrameShape(QtWidgets.QFrame.Shape.StyledPan
el)

self.iterationDataFrame.setFrameShadow(QtWidgets.QFrame.Shadow.Raised)
        self.iterationDataFrame.setObjectName("iterationDataFrame")
        self.verticalLayoutWidget_3 =
QtWidgets.QWidget(parent=self.iterationDataFrame)
        self.verticalLayoutWidget_3.setGeometry(QtCore.QRect(10, 0,
859, 701))

self.verticalLayoutWidget_3.setObjectName("verticalLayoutWidget_3")
        self.iterationTabLayout =
QtWidgets.QVBoxLayout(self.verticalLayoutWidget_3)
        self.iterationTabLayout.setContentsMargins(0, 0, 0, 0)
        self.iterationTabLayout.setObjectName("iterationTabLayout")
        self.iterationLabelLayout = QtWidgets.QHBoxLayout()

self.iterationLabelLayout.setObjectName("iterationLabelLayout")
        spacerItem3 = QtWidgets.QSpacerItem(40, 20,
QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Minimum)
        self.iterationLabelLayout.addItem(spacerItem3)
        self.iterationLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.iterationLabel.setObjectName("iterationLabel")
        self.iterationLabelLayout.addWidget(self.iterationLabel)
        self.iterationNumLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.iterationNumLabel.setObjectName("iterationNumLabel")
        self.iterationLabelLayout.addWidget(self.iterationNumLabel)
        spacerItem4 = QtWidgets.QSpacerItem(40, 20,
QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Minimum)
        self.iterationLabelLayout.addItem(spacerItem4)
        self.iterationTabLayout.addLayout(self.iterationLabelLayout)
        self.backpackTableWidget =
QtWidgets.QTableWidget(parent=self.verticalLayoutWidget_3)
        sizePolicy =
QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
```

```
sizePolicy.setHeightForWidth(self.backpackTableWidget.sizePolicy().has
HeightForWidth())
        self.backpackTableWidget.setSizePolicy(sizePolicy)

self.backpackTableWidget.setEditTriggers(QtWidgets.QAbstractItemView.E
ditTrigger.NoEditTriggers)
        self.backpackTableWidget.setRowCount(20)
        self.backpackTableWidget.setColumnCount(5)
        self.backpackTableWidget.setObjectName("backpackTableWidget")
        item = QtWidgets.QTableWidgetItem()
        self.backpackTableWidget.setHorizontalHeaderItem(0, item)
        item = QtWidgets.QTableWidgetItem()
        self.backpackTableWidget.setHorizontalHeaderItem(1, item)
        item = QtWidgets.QTableWidgetItem()
        self.backpackTableWidget.setHorizontalHeaderItem(2, item)
        item = QtWidgets.QTableWidgetItem()
        self.backpackTableWidget.setHorizontalHeaderItem(3, item)
        item = QtWidgets.QTableWidgetItem()
        self.backpackTableWidget.setHorizontalHeaderItem(4, item)
        self.iterationTabLayout.addWidget(self.backpackTableWidget)
        self.horizontalLayout_9 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_9.setObjectName("horizontalLayout_9")
        self.verticalLayout_5 = QtWidgets.QVBoxLayout()
        self.verticalLayout_5.setObjectName("verticalLayout_5")
        self.label =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.label.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.label.setObjectName("label")
        self.verticalLayout_5.addWidget(self.label)
        self.horizontalLayout_11 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_11.setObjectName("horizontalLayout_11")
        self.textCurBPCostLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.textCurBPCostLabel.setObjectName("textCurBPCostLabel")
        self.horizontalLayout_11.addWidget(self.textCurBPCostLabel)
        self.curBPCostLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.curBPCostLabel.setObjectName("curBPCostLabel")
        self.horizontalLayout_11.addWidget(self.curBPCostLabel)
        self.verticalLayout_5.addLayout(self.horizontalLayout_11)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.textCurWeightLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.textCurWeightLabel.setObjectName("textCurWeightLabel")
        self.horizontalLayout_2.addWidget(self.textCurWeightLabel)
        self.curWeightLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.curWeightLabel.setObjectName("curWeightLabel")
        self.horizontalLayout_2.addWidget(self.curWeightLabel)
        self.verticalLayout_5.addLayout(self.horizontalLayout_2)
        self.backpackData_1 = QtWidgets.QHBoxLayout()
        self.backpackData_1.setObjectName("backpackData_1")
        self.textFreeSpacLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.textFreeSpacLabel.setObjectName("textFreeSpacLabel")
```

```
        self.backpackData_1.addWidget(self.textFreeSpacLabel)
        self.freeSpaveLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.freeSpaveLabel.setObjectName("freeSpaveLabel")
        self.backpackData_1.addWidget(self.freeSpaveLabel)
        self.verticalLayout_5.addLayout(self.backpackData_1)
        self.horizontalLayout_9.addLayout(self.verticalLayout_5)
        self.verticalLayout_6 = QtWidgets.QVBoxLayout()
        self.verticalLayout_6.setObjectName("verticalLayout_6")
        self.label_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.label_2.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.label_2.setObjectName("label_2")
        self.verticalLayout_6.addWidget(self.label_2)
        self.horizontalLayout_12 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_12.setObjectName("horizontalLayout_12")
        self.textCurBPCostLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)

self.textCurBPCostLabel_2.setObjectName("textCurBPCostLabel_2")
        self.horizontalLayout_12.addWidget(self.textCurBPCostLabel_2)
        self.curBPCostLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.curBPCostLabel_2.setObjectName("curBPCostLabel_2")
        self.horizontalLayout_12.addWidget(self.curBPCostLabel_2)
        self.verticalLayout_6.addLayout(self.horizontalLayout_12)
        self.horizontalLayout_8 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_8.setObjectName("horizontalLayout_8")
        self.textCurWeightLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)

self.textCurWeightLabel_2.setObjectName("textCurWeightLabel_2")
        self.horizontalLayout_8.addWidget(self.textCurWeightLabel_2)
        self.curWeightLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.curWeightLabel_2.setObjectName("curWeightLabel_2")
        self.horizontalLayout_8.addWidget(self.curWeightLabel_2)
        self.verticalLayout_6.addLayout(self.horizontalLayout_8)
        self.backpackData_2 = QtWidgets.QHBoxLayout()
        self.backpackData_2.setObjectName("backpackData_2")
        self.textFreeSpacLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.textFreeSpacLabel_2.setObjectName("textFreeSpacLabel_2")
        self.backpackData_2.addWidget(self.textFreeSpacLabel_2)
        self.freeSpaveLabel_2 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.freeSpaveLabel_2.setObjectName("freeSpaveLabel_2")
        self.backpackData_2.addWidget(self.freeSpaveLabel_2)
        self.verticalLayout_6.addLayout(self.backpackData_2)
        self.horizontalLayout_9.addLayout(self.verticalLayout_6)
        self.verticalLayout_7 = QtWidgets.QVBoxLayout()
        self.verticalLayout_7.setObjectName("verticalLayout_7")
        self.label_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.label_3.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.label_3.setObjectName("label_3")
        self.verticalLayout_7.addWidget(self.label_3)
```

```
        self.horizontalLayout_14 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_14.setObjectName("horizontalLayout_14")
        self.textCurBPCostLabel_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)

self.textCurBPCostLabel_3.setObjectName("textCurBPCostLabel_3")
        self.horizontalLayout_14.addWidget(self.textCurBPCostLabel_3)
        self.curBPCostLabel_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.curBPCostLabel_3.setObjectName("curBPCostLabel_3")
        self.horizontalLayout_14.addWidget(self.curBPCostLabel_3)
        self.verticalLayout_7.addLayout(self.horizontalLayout_14)
        self.horizontalLayout_10 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_10.setObjectName("horizontalLayout_10")
        self.textCurWeightLabel_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)

self.textCurWeightLabel_3.setObjectName("textCurWeightLabel_3")
        self.horizontalLayout_10.addWidget(self.textCurWeightLabel_3)
        self.curWeightLabel_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.curWeightLabel_3.setObjectName("curWeightLabel_3")
        self.horizontalLayout_10.addWidget(self.curWeightLabel_3)
        self.verticalLayout_7.addLayout(self.horizontalLayout_10)
        self.backpackData_3 = QtWidgets.QHBoxLayout()
        self.backpackData_3.setObjectName("backpackData_3")
        self.textFreeSpacLabel_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.textFreeSpacLabel_3.setObjectName("textFreeSpacLabel_3")
        self.backpackData_3.addWidget(self.textFreeSpacLabel_3)
        self.freeSpaveLabel_3 =
QtWidgets.QLabel(parent=self.verticalLayoutWidget_3)
        self.freeSpaveLabel_3.setObjectName("freeSpaveLabel_3")
        self.backpackData_3.addWidget(self.freeSpaveLabel_3)
        self.verticalLayout_7.addLayout(self.backpackData_3)
        self.horizontalLayout_9.addLayout(self.verticalLayout_7)
        self.iterationTabLayout.addLayout(self.horizontalLayout_9)
        self.line =
QtWidgets.QFrame(parent=self.verticalLayoutWidget_3)
        self.line.setLineWidth(2)
        self.line.setMidLineWidth(1)
        self.line.setFrameShape(QtWidgets.QFrame.Shape.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Shadow.Sunken)
        self.line.setObjectName("line")
        self.iterationTabLayout.addWidget(self.line)
        self.iterationTabWidget_2.addTab(self.iterationTab_2, "")
        self.gridLayout.addWidget(self.iterationTabWidget_2, 0, 1, 1,
2)
        self.startButton = QtWidgets.QPushButton(parent=self.wind)
        self.startButton.setEnabled(False)
        font = QtGui.QFont()
        font.setBold(True)
        font.setUnderline(False)
        font.setWeight(75)
        self.startButton.setFont(font)
        self.startButton.setObjectName("startButton")
        self.gridLayout.addWidget(self.startButton, 2, 1, 1, 1)
```

```
        MainWindow.setCentralWidget(self.wind)

        self.retranslateUi(MainWindow)
        self.crossing_method_comboBox.setCurrentIndex(0)
        self.mutation_method_comboBox.setCurrentIndex(0)
        self.iterationTabWidget_2.setCurrentIndex(0)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Задача о
рюкзаках"))
        self.backButton.setText(_translate("MainWindow", "Шаг назад"))
        self.forwardButton.setText(_translate("MainWindow", "Шаг
вперед"))
        self.resultButton.setText(_translate("MainWindow",
"Результат"))
        self.ParamsLabel.setText(_translate("MainWindow", "Параметры
работы алгоритма"))
        self.backpackValueLabel.setText(_translate("MainWindow",
"Допустимый вес рюкзака"))

self.backpackValueLE.setPlaceholderText(_translate("MainWindow", "Вес
рюкзака"))
        self.genAmountLabel.setText(_translate("MainWindow",
"Количество поколений"))

self.generationAmountLE.setPlaceholderText(_translate("MainWindow",
"Кол-во поколений"))
        self.entityAmountLabel.setText(_translate("MainWindow",
"Количество особей в поколении"))

self.entityAmountLE.setPlaceholderText(_translate("MainWindow", "Кол-
во особей в поколении"))
        self.probabilityCrossingLabel.setText(_translate("MainWindow",
"Вероятность скрещевания"))

self.probabilityCrossingLabel_2.setText(_translate("MainWindow",
"Вероятность мутации"))

self.parent_selection_method_label.setText(_translate("MainWindow",
"Способ отбора родителей"))
        self.parent_selection_comboBox.setItemText(0,
_translate("MainWindow", "Турнир"))
        self.parent_selection_comboBox.setItemText(1,
_translate("MainWindow", "Рулетка"))
        self.parent_selection_comboBox.setItemText(2,
_translate("MainWindow", "Инбридинг"))
        self.crossing_method_label.setText(_translate("MainWindow",
"Способ скрещевания"))

self.crossing_method_comboBox.setCurrentText(_translate("MainWindow",
"Равномерное скрещивание"))
        self.crossing_method_comboBox.setItemText(0,
_translate("MainWindow", "Равномерное скрещивание"))
        self.crossing_method_comboBox.setItemText(1,
_translate("MainWindow", "Дискретная рекомбинация"))
```

```
        self.crossing_method_comboBox.setItemText(2,
_translate("MainWindow", "Промежуточная рекомбинация"))
        self.mutation_method_label.setText(_translate("MainWindow",
"Способ мутации"))

self.mutation_method_comboBox.setCurrentText(_translate("MainWindow",
"Плотность мутации"))
        self.mutation_method_comboBox.setItemText(0,
_translate("MainWindow", "Плотность мутации"))
        self.mutation_method_comboBox.setItemText(1,
_translate("MainWindow", "Мутация перестановкой"))
        self.mutation_method_comboBox.setItemText(2,
_translate("MainWindow", "Мутация обменом"))

self.methodOfSelectingIndividsLabel.setText(_translate("MainWindow",
"Способ отбора особей в следующее поколение"))
        self.methodOfSelectingIndividsComboBox.setItemText(0,
_translate("MainWindow", "Элитарный отбор"))
        self.methodOfSelectingIndividsComboBox.setItemText(1,
_translate("MainWindow", "Отбор вытеснением"))
        self.methodOfSelectingIndividsComboBox.setItemText(2,
_translate("MainWindow", "Отбор усечением"))
        self.saveButton.setText(_translate("MainWindow", "Изменить"))
        self.DataLabel.setText(_translate("MainWindow", "Данные"))
        self.randomGenButton.setText(_translate("MainWindow",
"Сгенерировать случайно"))
        self.browseButton.setText(_translate("MainWindow", "Загрузить
из файла"))
        self.inputButton.setText(_translate("MainWindow", "Ввести
вручную"))
        self.resetDataButton.setText(_translate("MainWindow",
"Сбросить данные"))
        self.plainTextEdit.setPlainText(_translate("MainWindow",
"Данное приложение решает задачу о неограниченном рюкзаке с помощью
генетического алгоритма.\n"
"\n"
"Кнопка *изменить* в области \"Параметры работы алгоритма\" позволяет
применить внесённые изменения в параметры работы алгоритма.\n"
"Кнопка *запуск* позволяет запустить алгоритм с выбранными
характеристиками и выбранными данными. До выбора данных кнопка
неактивна. Повторное нажатие кнопки перезапускает алгоритм с теми же
параметрами и данными.\n"
"Кнопки *шаг вперёд* и *шаг назад* позволяют рассмотреть
соответственно следующую и предыдущую итерации алгоритма. Кнопка
*результат* позволяет рассмотреть результат работы алгоритма, т.е. его
последнюю итерацию.\n"
"\n"
"\n"
"*Параметры работы алгоритма*:\n"
"Допустимый вес рюкзака - наибольший вес совокупности предметов,
которые можно поместить в рюкзак. Варьируется от 1 до 999.\n"
"Количество поколений - количество итераций, которые проведёт
алгоритм. Чем число больше, тем больше шансов на нахождение истинного
решения. Варьируется от 1 до 999 (тут стоило бы поправить на 99).\n"
"Количество особей в поколении - количество рюкзаков с вещами, которые
рассматриваются на одной итерации алгоритма. Большое число
```

```
способствует большему разнообразию внутри поколения, а значит и
повышению вероятности нахождения верного решения.\n"
"Вероятность скрещивания - вероятность того, что между двумя
родителями, отобранными определённым \"способом отбора родителей\",
произойдёт скрещивание согласно \"способу скрещивания\". Высокая
вероятность скрещивания благоприятно влияет на разнообразие внутри
поколения. Рекомендованный диапозон значений от 0.5 до 0.9.\n"
"Вероятность мутации - вероятность того, что потомок, полученный в
результате срещивания мутирует согласно \"способу мутации\". Мутация
позволяет зародиться в поколении принципиально новому решению. Высокая
вероятность мутации понижает скорость сходимости алгоритма, но
позволяет с большей вероятностью найти истинное решение.\n"
"Способ отбора родителей в следующее поколение - отбор пригодных
решений для дальнейшего рассмотрения\n"
"\n"
"Данные:\n"
"Кнопка *Сгенерировать случайно* открывает окно с полем для ввода
количества предметов. Алгоритм создаёт случайную выборку предметов
введённого размера, каждый предмет из которой может иметь вес и цену
от 1 до 100.\n"
"Кнопка *Загрузить из файла* окрывает окно проводника для выбора файла
.txt для загрузки данных. Файл должен содержать 2 числа на каждой
строке - цену и вес."))

self.iterationTabWidget_2.setTabText(self.iterationTabWidget_2.indexOf
(self.instructionTab_2), _translate("MainWindow", "Инструкция"))
        self.noDataLabel.setText(_translate("MainWindow", "Нет
данных"))
        self.iterationLabel.setText(_translate("MainWindow", "Итерация
номер "))
        self.iterationNumLabel.setText(_translate("MainWindow", "0"))
        self.backpackTableWidget.setSortingEnabled(True)
        item = self.backpackTableWidget.horizontalHeaderItem(0)
        item.setText(_translate("MainWindow", "Вес"))
        item = self.backpackTableWidget.horizontalHeaderItem(1)
        item.setText(_translate("MainWindow", "Стоимость"))
        item = self.backpackTableWidget.horizontalHeaderItem(2)
        item.setText(_translate("MainWindow", "Количество в 1"))
        item = self.backpackTableWidget.horizontalHeaderItem(3)
        item.setText(_translate("MainWindow", "Количество в 2"))
        item = self.backpackTableWidget.horizontalHeaderItem(4)
        item.setText(_translate("MainWindow", "Количество в 3"))
        self.label.setText(_translate("MainWindow", "Рюкзак 1"))
        self.textCurBPCostLabel.setText(_translate("MainWindow",
"Текущая стоимость рюкзака:"))
        self.curBPCostLabel.setText(_translate("MainWindow", "-1"))
        self.textCurWeightLabel.setText(_translate("MainWindow",
"Текущий вес рюкзака:"))
        self.curWeightLabel.setText(_translate("MainWindow", "-1"))
        self.textFreeSpacLabel.setText(_translate("MainWindow",
"Оставшееся свободное место:"))
        self.freeSpaveLabel.setText(_translate("MainWindow", "-1"))
        self.label_2.setText(_translate("MainWindow", "Рюкзак 2"))
        self.textCurBPCostLabel_2.setText(_translate("MainWindow",
"Текущая стоимость рюкзака:"))
        self.curBPCostLabel_2.setText(_translate("MainWindow", "-1"))
```

```
        self.textCurWeightLabel_2.setText(_translate("MainWindow",
"Текущий вес рюкзака:"))
        self.curWeightLabel_2.setText(_translate("MainWindow", "-1"))
        self.textFreeSpacLabel_2.setText(_translate("MainWindow",
"Оставшееся свободное место:"))
        self.freeSpaveLabel_2.setText(_translate("MainWindow", "-1"))
        self.label_3.setText(_translate("MainWindow", "Рюкзак 3"))
        self.textCurBPCostLabel_3.setText(_translate("MainWindow",
"Текущая стоимость рюкзака:"))
        self.curBPCostLabel_3.setText(_translate("MainWindow", "-1"))
        self.textCurWeightLabel_3.setText(_translate("MainWindow",
"Текущий вес рюкзака:"))
        self.curWeightLabel_3.setText(_translate("MainWindow", "-1"))
        self.textFreeSpacLabel_3.setText(_translate("MainWindow",
"Оставшееся свободное место:"))
        self.freeSpaveLabel_3.setText(_translate("MainWindow", "-1"))

self.iterationTabWidget_2.setTabText(self.iterationTabWidget_2.indexOf
(self.iterationTab_2), _translate("MainWindow", "Итерация алгоритма"))
        self.startButton.setText(_translate("MainWindow", "Запуск"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec())
```

Название файла: src/UIs/UIRandGenDialog.py

```
# Form implementation generated from reading ui file 'randGenUI.ui'
#
# Created by: PyQt6 UI code generator 6.7.0
#
# WARNING: Any manual changes made to this file will be lost when
pyuic6 is
# run again.  Do not edit this file unless you know what you are
doing.


from PyQt6 import QtCore, QtGui, QtWidgets


class Ui_randGenDialog(object):
    def setupUi(self, randGenDialog):
        randGenDialog.setObjectName("randGenDialog")
        randGenDialog.resize(400, 152)
        randGenDialog.setMinimumSize(QtCore.QSize(400, 152))
        randGenDialog.setMaximumSize(QtCore.QSize(400, 152))
        self.verticalLayoutWidget =
QtWidgets.QWidget(parent=randGenDialog)
        self.verticalLayoutWidget.setGeometry(QtCore.QRect(80, 30,
229, 49))

self.verticalLayoutWidget.setObjectName("verticalLayoutWidget")
```

```
        self.verticalLayout =
QtWidgets.QVBoxLayout(self.verticalLayoutWidget)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout.setObjectName("verticalLayout")
        self.amountLabel =
QtWidgets.QLabel(parent=self.verticalLayoutWidget)

self.amountLabel.setAlignment(QtCore.Qt.AlignmentFlag.AlignCenter)
        self.amountLabel.setObjectName("amountLabel")
        self.verticalLayout.addWidget(self.amountLabel)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        spacerItem = QtWidgets.QSpacerItem(40, 20,
QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Minimum)
        self.horizontalLayout_2.addItem(spacerItem)
        self.AmountLineEdit =
QtWidgets.QLineEdit(parent=self.verticalLayoutWidget)
        self.AmountLineEdit.setObjectName("AmountLineEdit")
        self.horizontalLayout_2.addWidget(self.AmountLineEdit)
        spacerItem1 = QtWidgets.QSpacerItem(40, 20,
QtWidgets.QSizePolicy.Policy.Expanding,
QtWidgets.QSizePolicy.Policy.Minimum)
        self.horizontalLayout_2.addItem(spacerItem1)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.doneButton = QtWidgets.QPushButton(parent=randGenDialog)
        self.doneButton.setGeometry(QtCore.QRect(230, 110, 158, 32))
        self.doneButton.setObjectName("doneButton")
        self.CancelButton =
QtWidgets.QPushButton(parent=randGenDialog)
        self.CancelButton.setGeometry(QtCore.QRect(20, 110, 121, 32))
        self.CancelButton.setObjectName("CancelButton")

        self.retranslateUi(randGenDialog)
        QtCore.QMetaObject.connectSlotsByName(randGenDialog)

    def retranslateUi(self, randGenDialog):
        _translate = QtCore.QCoreApplication.translate
        randGenDialog.setWindowTitle(_translate("randGenDialog",
"Случайная генерация"))
        self.amountLabel.setText(_translate("randGenDialog", "Введите
количество предметов"))
        self.doneButton.setText(_translate("randGenDialog", "Готово"))
        self.CancelButton.setText(_translate("randGenDialog",
"Назад"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    randGenDialog = QtWidgets.QDialog()
    ui = Ui_randGenDialog()
    ui.setupUi(randGenDialog)
    randGenDialog.show()
    sys.exit(app.exec())
```

Название файла: src/gui.py

```
from PyQt6 import QtCore, QtGui, QtWidgets
from PyQt6.QtWidgets import QMainWindow

from UIs.UIMainWindow import Ui_MainWindow
from UIs.UIRandGenDialog import Ui_randGenDialog
from UIs.UIHandInputUI import Ui_HandInputDialog


class MainWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.mainWindowUI = Ui_MainWindow()
        self.mainWindowUI.setupUi(self)


class RandGenDialog(QtWidgets.QDialog):
    def __init__(self):
        QtWidgets.QDialog.__init__(self)
        self.randGenDialogUI = Ui_randGenDialog()
        self.randGenDialogUI.setupUi(self)


class HandInputDialog(QtWidgets.QDialog):
    def __init__(self):
        QtWidgets.QDialog.__init__(self)
        self.handInputDialogUI = Ui_HandInputDialog()
        self.handInputDialogUI.setupUi(self)
```

Название файла: src/UILogic.py

```
import os
import sys

from PyQt6 import QtWidgets
from PyQt6.QtGui import QIntValidator
from PyQt6.QtWidgets import QFileDialog, QTableWidgetItem, QMessageBox
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.figure import Figure

import gui
from src.libs.genetic_algorithm import *


class Data:
    def __init__(self):
        self.parent_selection_strategies = {
            "Турнир": TournamentSelection(),
            "Рулетка": RouletteSelection(),
            "Инбридинг": InbreedingSelection()
        }

        self.crossing_strategies = {
            "Равномерное скрещивание": UniformCrossing(),
            "Дискретная рекомбинация": DiscreteRecombination(),
            "Промежуточная рекомбинация": IntermediateRecombination()
```

```
        }

        self.mutation_strategies = {
            "Плотность мутации": DensityMutation(),
            "Мутация перестановкой": PermutationMutation(),
            "Мутация обменом": ExchangeMutation()
        }

        self.generation_selection_strategies = {
            "Элитарный отбор": EliteSelection(),
            "Отбор вытеснением": ExclusionSelection(),
            "Отбор усечением": TruncationSelection()
        }

        self.algParams = AlgorithmParameters(100,
                                             0.9,
                                             0.3,
                                             25,
                                             50,

self.parent_selection_strategies["Турнир"],

self.crossing_strategies["Равномерное скрещивание"],

self.mutation_strategies["Плотность мутации"],

self.generation_selection_strategies["Элитарный отбор"])
        self.geneticAlg = None
        self.iterationsInfo = list[IterationInfo]
        self.iteration = 0

        self.backpackAmount = -1
        self.inputFileName = ""
        self.items = []

        self.algNum = -1

    def generateRandomItems(self) -> None:
        self.items.clear()
        for i in range(self.backpackAmount):
            item = Item(random.randint(1, 100), random.randint(1,
100))
            self.items.append(item)

    def readItemsFromFile(self) -> bool:
        self.items.clear()
        self.backpackAmount = 0
        with open(self.inputFileName, 'r') as file:
            lines = file.readlines()
            if len(lines) < 1:
                return False
            for line in lines:
                try:
                    weight, cost = line.split()
                    self.items.append(Item(int(cost), int(weight)))
                    self.backpackAmount += 1
                except ValueError:
```

```
                    self.items.clear()
                    return False
        return True


class MplCanvas(FigureCanvas):
    # зачем аргумент parent?
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)
        self.axes = fig.add_subplot(111)
        super(MplCanvas, self).__init__(fig)


class UILogic:
    def __init__(self):
        self.mainWindow = gui.MainWindow()
        self.mainWindowUI = self.mainWindow.mainWindowUI

        self.randGenDialog = gui.RandGenDialog()
        self.randGenDialogUI = self.randGenDialog.randGenDialogUI

        self.handInputDialog = gui.HandInputDialog()
        self.handInputDialogUI =
self.handInputDialog.handInputDialogUI

        self.data = Data()
        self.canvas = MplCanvas(self, width=5, height=4, dpi=100)

        self.mainWindowUI.startButton.setEnabled(False)
        self.mainWindowUI.iterationDataFrame.setVisible(False)
        self.mainWindowUI.iterationTabLayout.addWidget(self.canvas)

        self.connectButtons()
        self.adjustLineEdits()

    def drawPlot(self, maxFitness: list[float], averageFitness:
list[float], iter: int) -> None:
        x_len = iter
        self.canvas.axes.clear()

        self.canvas.axes.plot(list(range(x_len)), averageFitness, 'r-
', label='Средняя приспособленность')
        self.canvas.axes.plot(list(range(x_len)), maxFitness, 'b-',
label='Максимальная приспособленность')

        self.canvas.axes.grid()

        self.canvas.axes.set_xticks(np.arange(0, x_len + 1, 2))

        self.canvas.axes.set_xlabel('Поколение')
        self.canvas.axes.set_ylabel('Приспособленность')

        # Добавляем легенду
        self.canvas.axes.legend()

        # Перерисовываем график
        self.canvas.draw()
```

```
    def showErrorMessage(self, title: str, message: str) -> None:
        msg = QMessageBox()
        msg.setIcon(QMessageBox.Icon.Critical)
        msg.setText(message)
        msg.setWindowTitle(title)
        msg.setStandardButtons(QMessageBox.StandardButton.Ok)
        msg.exec()

    def adjustLineEdits(self):
        validator = QIntValidator(1, 999)
        self.mainWindowUI.backpackValueLE.setValidator(validator)
        self.mainWindowUI.generationAmountLE.setValidator(validator)
        self.mainWindowUI.entityAmountLE.setValidator(validator)
        self.randGenDialogUI.AmountLineEdit.setValidator(validator)


self.mainWindowUI.backpackValueLE.setText(str(self.data.algParams.maxB
ackpackWeight))

self.mainWindowUI.generationAmountLE.setText(str(self.data.algParams.m
axAmountOfGenerations))

self.mainWindowUI.entityAmountLE.setText(str(self.data.algParams.amoun
tOfIndividsPerGeneration))

self.mainWindowUI.crossingProbabilitySpin.setValue(self.data.algParams
.crossingProbability)

self.mainWindowUI.mutationProbabilitySpin.setValue(self.data.algParams
.mutationProbability)

    def connectButtons(self):

self.mainWindowUI.randomGenButton.clicked.connect(self.openRandomGenDi
alog)

self.mainWindowUI.browseButton.clicked.connect(self.browseEvent)

self.mainWindowUI.inputButton.clicked.connect(self.openHandInputDialog
)

self.mainWindowUI.startButton.clicked.connect(self.startButtonEvent)

self.mainWindowUI.saveButton.clicked.connect(self.updateParams)

self.mainWindowUI.forwardButton.clicked.connect(self.forwardButtonEven
t)

self.mainWindowUI.resultButton.clicked.connect(self.resultButtonEvent)

self.mainWindowUI.backButton.clicked.connect(self.backButtonEvent)

self.mainWindowUI.resetDataButton.clicked.connect(self.resetButtonEven
t)
```

```
self.handInputDialogUI.cancelButton.clicked.connect(self.closeHandInput
tDialogEvent)

self.handInputDialogUI.doneButton.clicked.connect(self.handInputDoneBu
ttonEvent)


self.randGenDialogUI.CancelButton.clicked.connect(self.closeGenDialogE
vent)

self.randGenDialogUI.doneButton.clicked.connect(self.randGenDoneButton
Event)

    def handInputDoneButtonEvent(self):
        if self.isTableEmpty():
            for i in
range(self.handInputDialogUI.tableWidget.rowCount()):
                weight = 0
                cost = 0

                if (self.handInputDialogUI.tableWidget.item(i, 0) is
not None) and \
                        self.handInputDialogUI.tableWidget.item(i,
0).text().isdigit():
                    weight =
int(self.handInputDialogUI.tableWidget.item(i, 0).text())
                if (self.handInputDialogUI.tableWidget.item(i, 1) is
not None) and \
                        self.handInputDialogUI.tableWidget.item(i,
1).text().isdigit():
                    cost =
int(self.handInputDialogUI.tableWidget.item(i, 1).text())
                new_item = Item(cost=cost, weight=weight)
                self.data.items.append(new_item)
            self.data.backpackAmount =
int(self.handInputDialogUI.AmountLineEdit.text())
            self.data.algNum = 3
            self.switchAlgorithms(self.data.algNum)

            self.mainWindowUI.startButton.setEnabled(True)
            self.handInputDialog.close()

    def isTableEmpty(self) -> bool:
        for i in range(self.handInputDialogUI.tableWidget.rowCount()):
            for j in
range(self.handInputDialogUI.tableWidget.columnCount()):
                if self.handInputDialogUI.tableWidget.item(i, j) is
None or \
                        not self.handInputDialogUI.tableWidget.item(i,
j).text().isdigit():
                    # print("poop " + str(i) + str(j))
                    return False
        return True

    def backButtonEvent(self):
        if self.data.iteration <= 1:
```

```
            return
        self.data.iteration -= 1
        self.iterateAlgorithm(self.data.iteration)

    def resultButtonEvent(self):
        self.data.iteration =
self.data.algParams.maxAmountOfGenerations
        self.iterateAlgorithm(self.data.iteration)

    def forwardButtonEvent(self):
        self.data.iteration += 1 if self.data.iteration <
self.data.algParams.maxAmountOfGenerations else 0
        self.iterateAlgorithm(self.data.iteration)

    def updateParams(self):
        if int(self.mainWindowUI.backpackValueLE.text()) <= 0:
            self.data.algParams.maxBackpackWeight = 1
        else:
            self.data.algParams.maxBackpackWeight =
int(self.mainWindowUI.backpackValueLE.text())

        self.data.algParams.crossingProbability =
float(self.mainWindowUI.crossingProbabilitySpin.value())
        self.data.algParams.mutationProbability =
float(self.mainWindowUI.mutationProbabilitySpin.value())

        if int(self.mainWindowUI.entityAmountLE.text()) < 3:
            self.data.algParams.amountOfIndividsPerGeneration = 3
        else:
            self.data.algParams.amountOfIndividsPerGeneration =
int(self.mainWindowUI.entityAmountLE.text())

        if int(self.mainWindowUI.generationAmountLE.text()) <= 0:
            self.data.algParams.maxAmountOfGenerations = 1
        else:
            self.data.algParams.maxAmountOfGenerations =
int(self.mainWindowUI.generationAmountLE.text())

        self.data.algParams.crossingStrategy = \

self.data.crossing_strategies[self.mainWindowUI.crossing_method_comboB
ox.currentData(0)]

        self.data.algParams.generationSelectionStrategy = \
            self.data.generation_selection_strategies[

self.mainWindowUI.methodOfSelectingIndividsComboBox.currentData(0)]

        self.data.algParams.parentsSelectionStrategy = \

self.data.parent_selection_strategies[self.mainWindowUI.parent_selecti
on_comboBox.currentData(0)]

        self.data.algParams.mutationStrategy = \

self.data.mutation_strategies[self.mainWindowUI.mutation_method_comboB
ox.currentData(0)]
```

```
    def openRandomGenDialog(self):
        self.randGenDialog.finished.connect(self.closeGenDialogEvent)
        self.mainWindow.setEnabled(False)

        self.randGenDialog.show()

    def switchAlgorithms(self, n: int):
        if n == 1:
            self.mainWindowUI.browseButton.setEnabled(False)
            self.mainWindowUI.inputButton.setEnabled(False)
            self.mainWindowUI.startButton.setEnabled(True)
        elif n == 2:
            self.mainWindowUI.randomGenButton.setEnabled(False)
            self.mainWindowUI.inputButton.setEnabled(False)
            self.mainWindowUI.startButton.setEnabled(True)
        elif n == 3:
            self.mainWindowUI.randomGenButton.setEnabled(False)
            self.mainWindowUI.browseButton.setEnabled(False)
            self.mainWindowUI.startButton.setEnabled(True)
        elif n == -1:
            self.mainWindowUI.randomGenButton.setEnabled(True)
            self.mainWindowUI.browseButton.setEnabled(True)
            self.mainWindowUI.inputButton.setEnabled(True)
            self.mainWindowUI.startButton.setEnabled(False)

    def startButtonEvent(self):
        self.mainWindowUI.iterationTabWidget_2.setCurrentIndex(1)
        self.startAlgorithm()

    def resetButtonEvent(self):
        self.data.algNum = -1
        self.data.items.clear()

        self.data.inputFileName = ""
        self.data.backpackAmount = -1

        self.switchAlgorithms(self.data.algNum)

    def openHandInputDialog(self):
        self.mainWindow.setEnabled(False)

        self.handInputDialog.show()

self.handInputDialog.finished.connect(self.closeHandInputDialogEvent)

self.handInputDialogUI.AmountLineEdit.textEdited.connect(self.handInpu
tTextChanged)

    def handInputTextChanged(self):
        text = self.handInputDialogUI.AmountLineEdit.text()

        if text != "":
            self.handInputDialogUI.tableWidget.setRowCount(int(text))

    def switchAllButtons(self, state: bool):
        self.mainWindowUI.inputButton.setEnabled(state)
```

```
        self.mainWindowUI.browseButton.setEnabled(state)
        self.mainWindowUI.backButton.setEnabled(state)
        self.mainWindowUI.forwardButton.setEnabled(state)
        self.mainWindowUI.resultButton.setEnabled(state)
        self.mainWindowUI.randomGenButton.setEnabled(state)

    def closeHandInputDialogEvent(self):
        self.mainWindow.setEnabled(True)
        self.handInputDialog.close()

    def closeGenDialogEvent(self):
        if self.randGenDialogUI.AmountLineEdit.text() != "" and
self.data.backpackAmount != -1:

self.randGenDialogUI.AmountLineEdit.setText(str(self.data.backpackAmou
nt))

        self.randGenDialog.close()
        self.mainWindow.setEnabled(True)

    def randGenDoneButtonEvent(self):
        if self.randGenDialogUI.AmountLineEdit.text() != "":
            self.data.algNum = 1
            self.switchAlgorithms(self.data.algNum)

            self.data.backpackAmount =
int(self.randGenDialogUI.AmountLineEdit.text())

            self.data.generateRandomItems()

            self.randGenDialog.close()
            self.mainWindowUI.startButton.setEnabled(True)

    # Открываем диалог (выбор файла)
    def browseEvent(self):
        self.mainWindow.setEnabled(False)

        file_filter = 'Text File (*.txt)'
        file_dialog = QFileDialog
        file_name = file_dialog.getOpenFileName(
            parent=self.mainWindow,
            caption='Открыть файл',
            directory=os.getcwd(),
            filter=file_filter,
            initialFilter='Text File (*.txt)'
        )
        if file_name[0] != "":
            self.data.inputFileName = file_name[0]
            if self.data.readItemsFromFile():
                self.data.algNum = 2
            else:
                self.data.inputFileName = ""
                self.showErrorMessage("Ошибка",
                                      "Ошибка при чтении файла:
несоответствующий формат данных")

        self.mainWindow.setEnabled(True)
```

```
        self.switchAlgorithms(self.data.algNum)

    def startAlgorithm(self):
        self.data.iteration = 0
        self.mainWindowUI.iterationDataFrame.setVisible(True)
        self.mainWindowUI.noDataLabel.setVisible(False)
        self.mainWindowUI.backpackTableWidget.clearContents()
        self.canvas.axes.clear()
        self.canvas.draw()

self.mainWindowUI.backpackTableWidget.setRowCount(self.data.backpackAm
ount)
        self.mainWindowUI.iterationNumLabel.setText("0")

        self.data.geneticAlg = GeneticAlgorithm(self.data.items,
self.data.algParams)

        self.data.iterationsInfo = self.data.geneticAlg.getSolution()

        for i in
range(self.mainWindowUI.backpackTableWidget.rowCount()):
            self.mainWindowUI.backpackTableWidget.setItem(i, 0,
QTableWidgetItem(str(self.data.items[i].weight)))
            self.mainWindowUI.backpackTableWidget.setItem(i, 1,
QTableWidgetItem(str(self.data.items[i].cost)))

    def iterateAlgorithm(self, iter: int):
        iteration = iter
        # print(iteration)
        if iteration <= len(self.data.iterationsInfo):

self.mainWindowUI.iterationNumLabel.setText(str(iteration))
            for i in
range(self.mainWindowUI.backpackTableWidget.rowCount()):
                for j in range(3):
                    self.mainWindowUI.backpackTableWidget.setItem(i, 2
+ j, QTableWidgetItem(
                        str(self.data.iterationsInfo[iteration -
1].bestBackpacks[j].genome[i])))


self.mainWindowUI.curBPCostLabel.setText(str(self.data.iterationsInfo[
iteration - 1].bestBackpacks[0].cost))
            self.mainWindowUI.curWeightLabel.setText(
                str(self.data.iterationsInfo[iteration -
1].bestBackpacks[0].weight))

self.mainWindowUI.freeSpaveLabel.setText(str(self.data.algParams.maxBa
ckpackWeight -

self.data.iterationsInfo[iteration - 1].bestBackpacks[

0].weight))

            self.mainWindowUI.curBPCostLabel_2.setText(
                str(self.data.iterationsInfo[iteration -
1].bestBackpacks[1].cost))
```

```
            self.mainWindowUI.curWeightLabel_2.setText(
                str(self.data.iterationsInfo[iteration -
1].bestBackpacks[1].weight))

self.mainWindowUI.freeSpaveLabel_2.setText(str(self.data.algParams.max
BackpackWeight -

self.data.iterationsInfo[iteration - 1].bestBackpacks[

1].weight))

            self.mainWindowUI.curBPCostLabel_3.setText(
                str(self.data.iterationsInfo[iteration -
1].bestBackpacks[2].cost))
            self.mainWindowUI.curWeightLabel_3.setText(
                str(self.data.iterationsInfo[iteration -
1].bestBackpacks[2].weight))

self.mainWindowUI.freeSpaveLabel_3.setText(str(self.data.algParams.max
BackpackWeight -

self.data.iterationsInfo[iteration - 1].bestBackpacks[

2].weight))

            self.drawPlot([x.currentMaxFitness for x in
self.data.iterationsInfo[:iteration]],
                          [x.currentAverageFitness for x in
self.data.iterationsInfo[:iteration]], iteration)


if __name__ == "__main__":
    app = QtWidgets.QApplication([])

    logic = UILogic()
    logic.mainWindow.show()

    sys.exit(app.exec())
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Рассмотрим решение следующей задачи о неограниченном рюкзаке.

Набор из следующих 10 предметов:

№	Вес	Цена
1	8	14
2	50	50
3	4	6
4	38	10
5	3	7
6	22	25
7	2	3
8	16	9
9	12	5
10	10	12

Максимальный допустимый вес рюкзака = 100

Количество поколений = 50

Количество особей в поколении = 25

Вероятность скрещивания, мутации, а также способы отбора родителей, скрещивания, мутации и отбора в следующее поколение будем менять.

Результаты тестирования:

№	Р скрещ-ия	Р мутации	Отбор род-лей	Скрещивание	Мутация	Отбор в следующее поколение	Макс. стоим-ть	Вес
1	0,9	0,3	Турнир	Равномерное	Плотность	Вытесн-е	**1975**	100
2	0,9	0,3	Рулетка	Дискретная	Перест-кой	Элитарный	**1792**	97
3	0,9	0,3	Инбр-инг	Промеж-ая	Обменом	Усечением	**1975**	100
4	0,9	0,45	Турнир	Равномерное	Плотность	Элитарный	**1792**	97
5	0,5	0,45	Турнир	Равномерное	Плотность	Элитарный	**1975**	100
6	0,9	0,15	Турнир	Равномерное	Плотность	Элитарный	**1975**	100

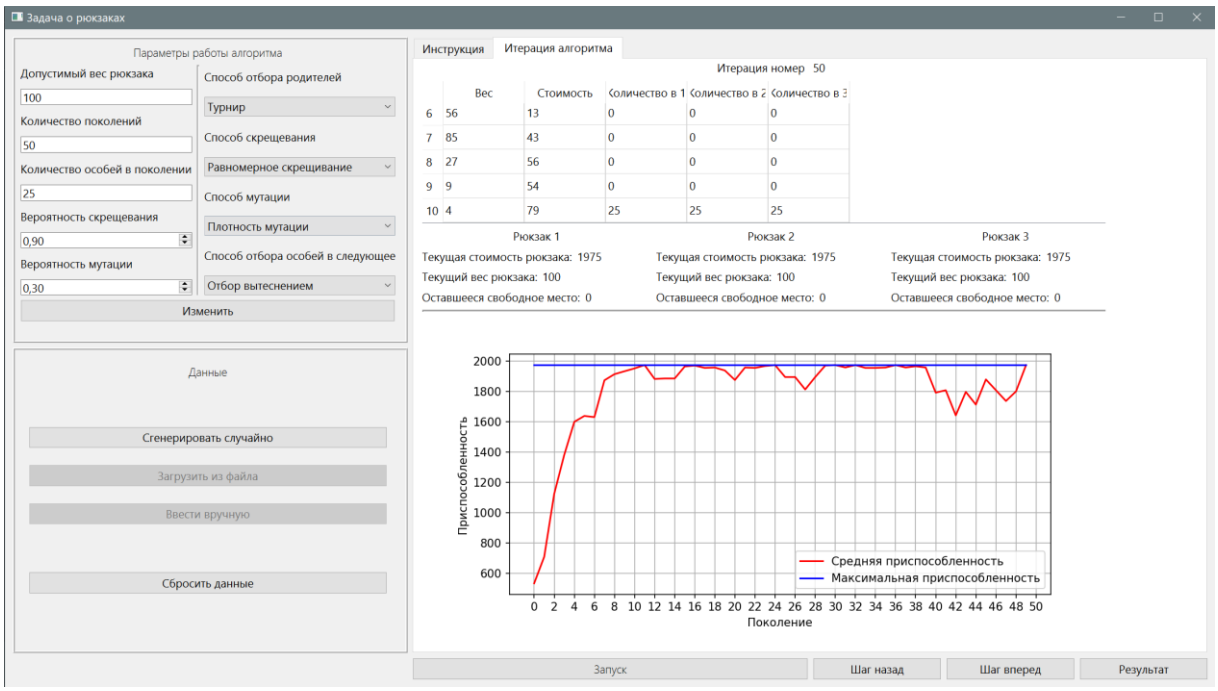

Рисунок 1 – Результат работы для тестового случая 1

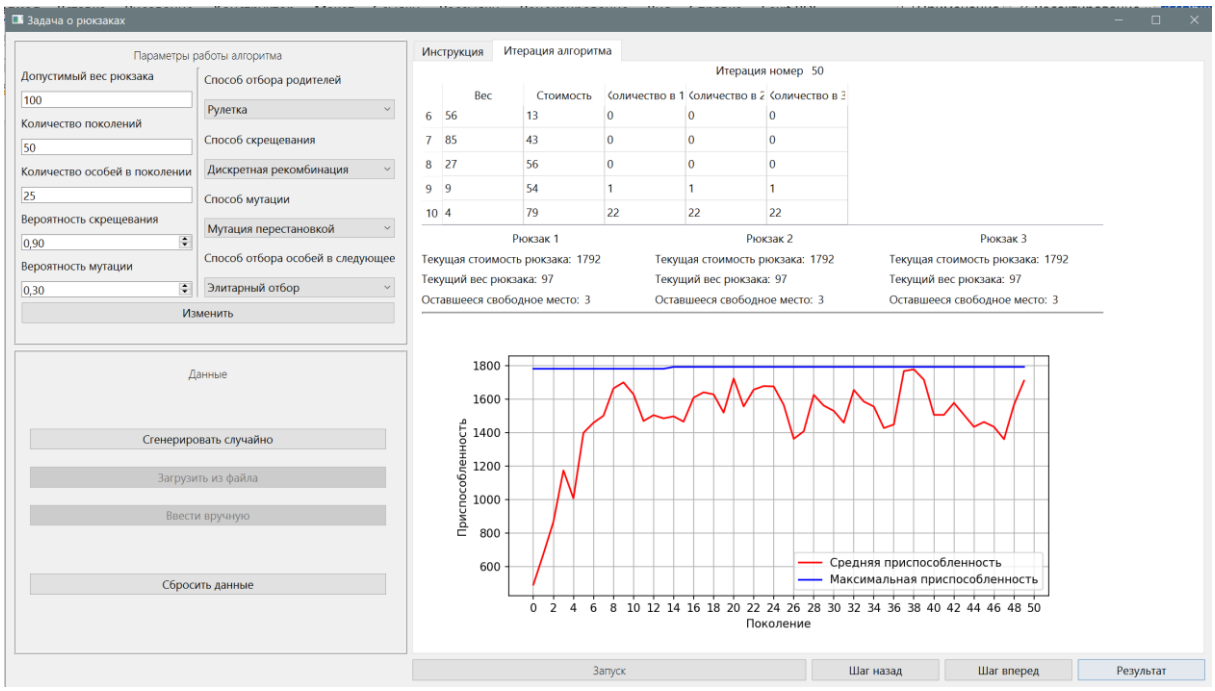

Рисунок 2 – Результат работы для тестового случая 2

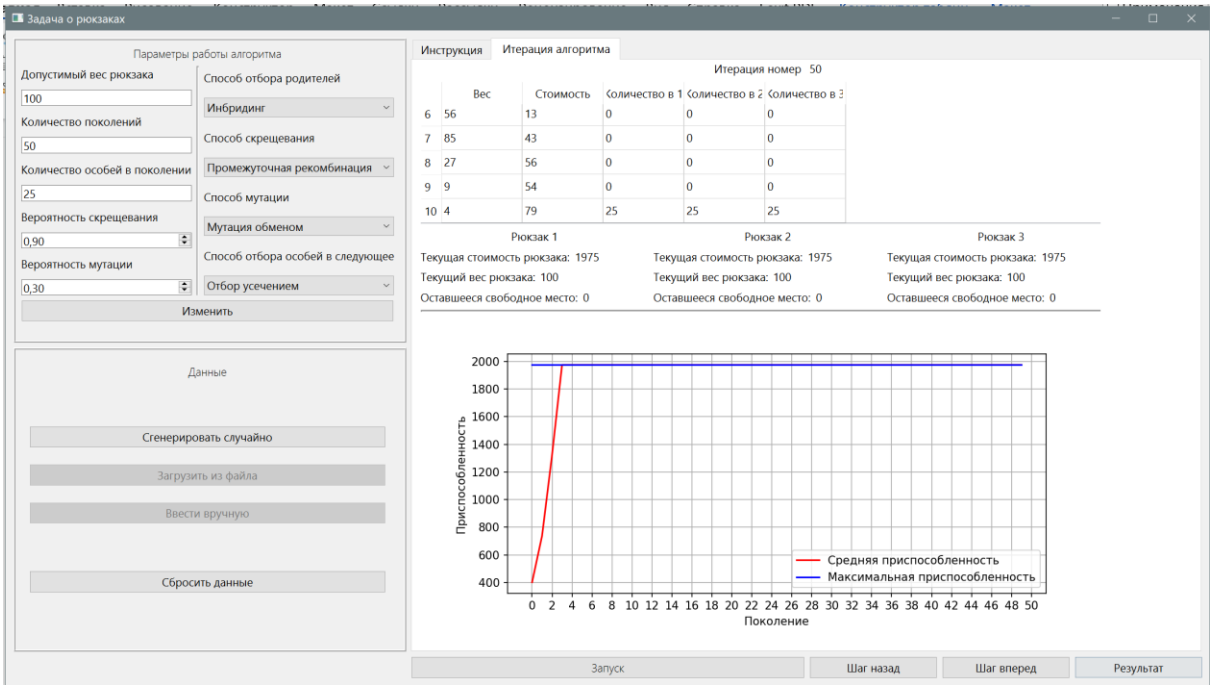

Рисунок 3 – Результат работы для тестового случая 3

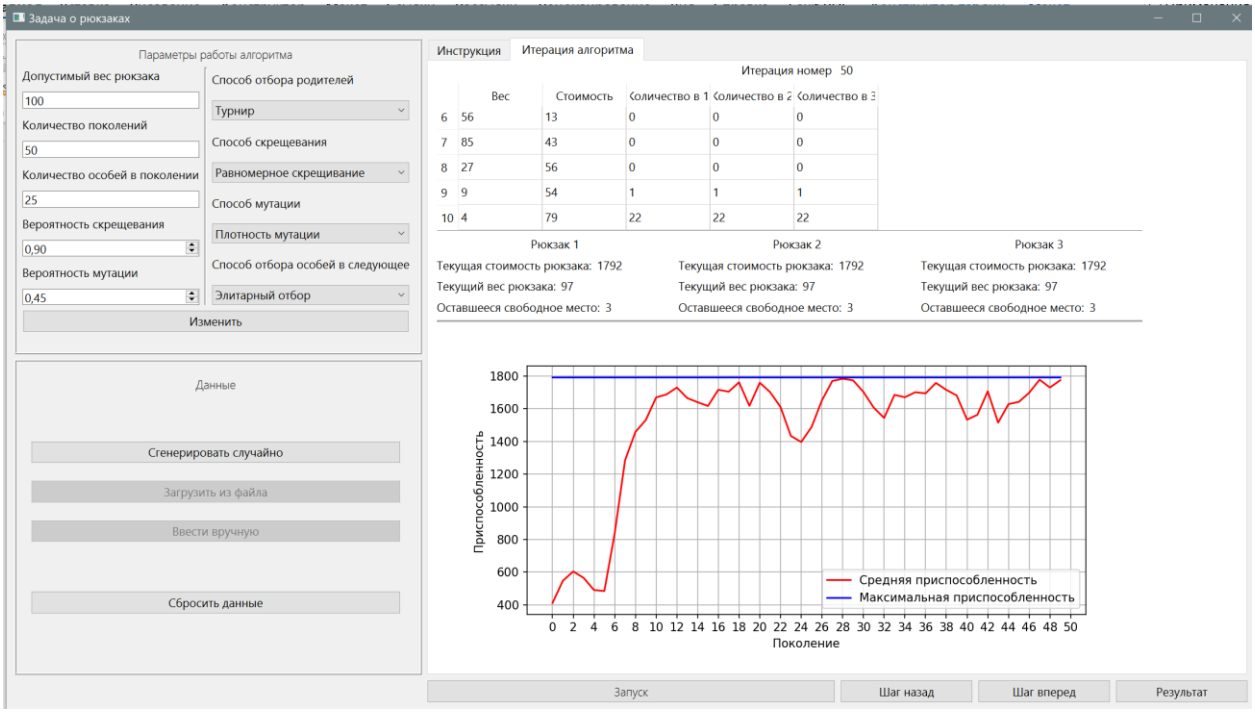

Рисунок 4 – Результат работы для тестового случая 4

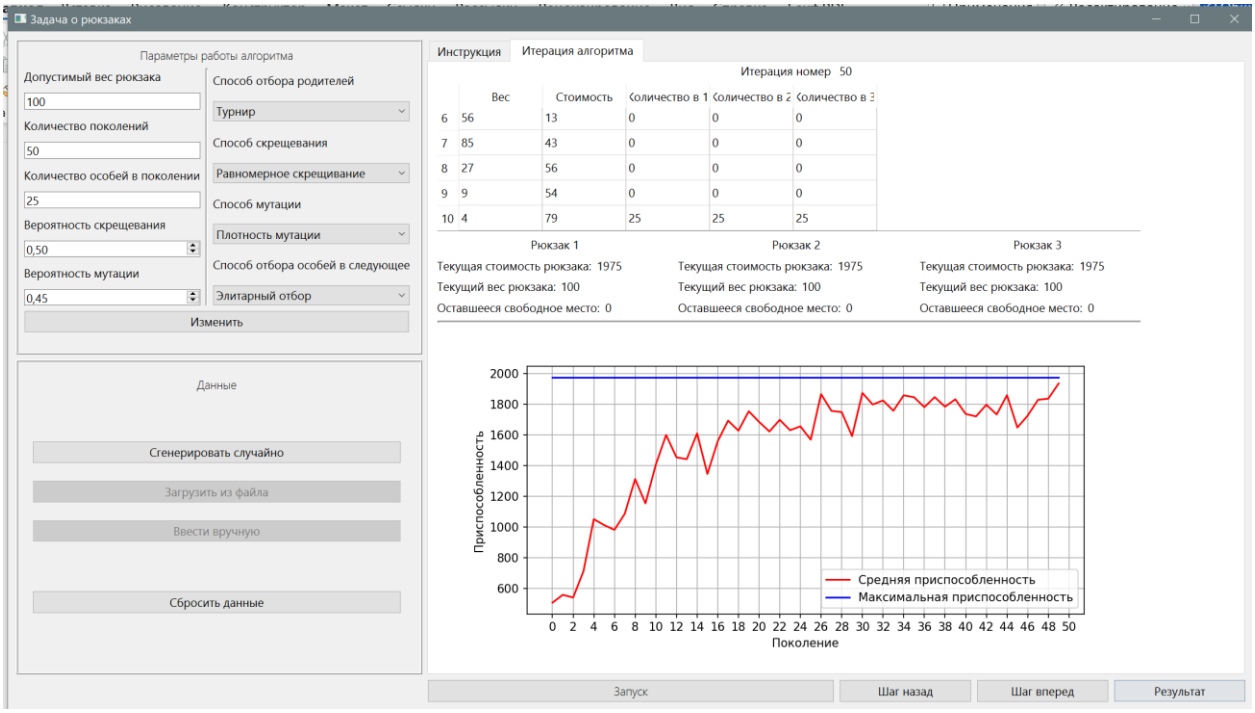

Рисунок 5 – Результат работы для тестового случая 5

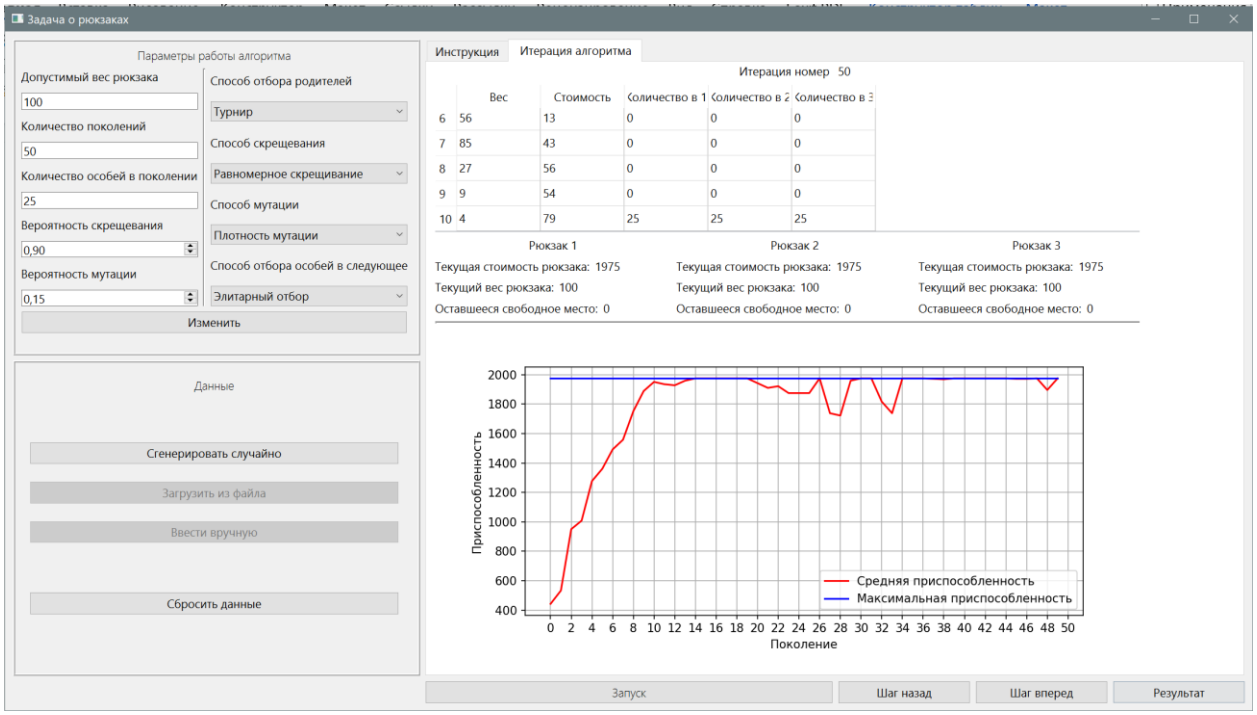

Рисунок 6 – Результат работы для тестового случая 6