Государственное бюджетное общеобразовательное учреждение города Москвы «Школа № 1532»

**Решение задач раскроя программным методом**

Скородина Майя Алексеевна, 10 класс,

ГБОУ школа 1532  
Руководитель: Сергиенко Антон Борисович, учитель информатики,

ГБОУ Школа №1532

Москва, 2023

**Оглавление**

Введение………………………………………….………………….…..………3

Цели и задачи работы…………………………...……….…………….….……3

Разработка вариантов решений.........………….……….…………..….………4

Реализация...........................................………….……….…………..….….....8

Исследовательская часть……………………………………………………...16

Выводы…………………..…………………………………………...…..…....17

Список используемой литературы………………...……………………...….18



Введение

Сегодня раскрой используется повсеместно: от расположения деталей на листах металла, до размещения наклеек на бумаге. Представим ситуацию: вам необходимо разместить несколько деталей на листе того же металла. При том, хотелось бы расположить детали максимально компактно, чтобы оставшуюся часть листа можно было использовать. Итак, если речь идет о нескольких деталях и относительно небольшом полотне материала, то человек может довольно хорошо справиться с такой задачей. Однако с десятками, сотнями разнообразных деталей и с квадратными метрами поверхности, на которой необходимо все разместить, вручную человек не сможет справиться быстро и также качественно.

Здесь формируется суть задачи раскроя: расположить фигуры максимально плотно друг к другу.

**Цель работы:** создать программу, решающую задачу раскроя с произвольно задаваемыми фигурами

**Задачи:**

1. Разработка и выбор алгоритма решения задачи
2. Создание плана реализации
3. Создание интерфейса с использованием Qt
4. Написание программ с разными алгоритмами согласно плану
5. Сравнение результатов, определение наилучшего варианта программы

**Актуальность:** Программа, решающая задачу раскроя, способна минимизировать количество производственного мусора в самых разных промышленных сферах. Что в свою очередь положительно отразится на экологии и экономике.

Примечание: в 2022 году автором решалась похожая задача раскроя. Нынешняя работа автора содержит другие алгоритмы и планы реализации, все программные коды были переписаны (по ООП). Результаты также стали лучше.

№ 1 Разработка вариантов решений

Было придумано несколько вариантов решений:

**1. «Маленький конструктор»:**

Фигура разбивается на составляющие (например, на прямоугольники) (рис 1). Программа ищет варианты расположения уже составляющих, а не данных фигур.

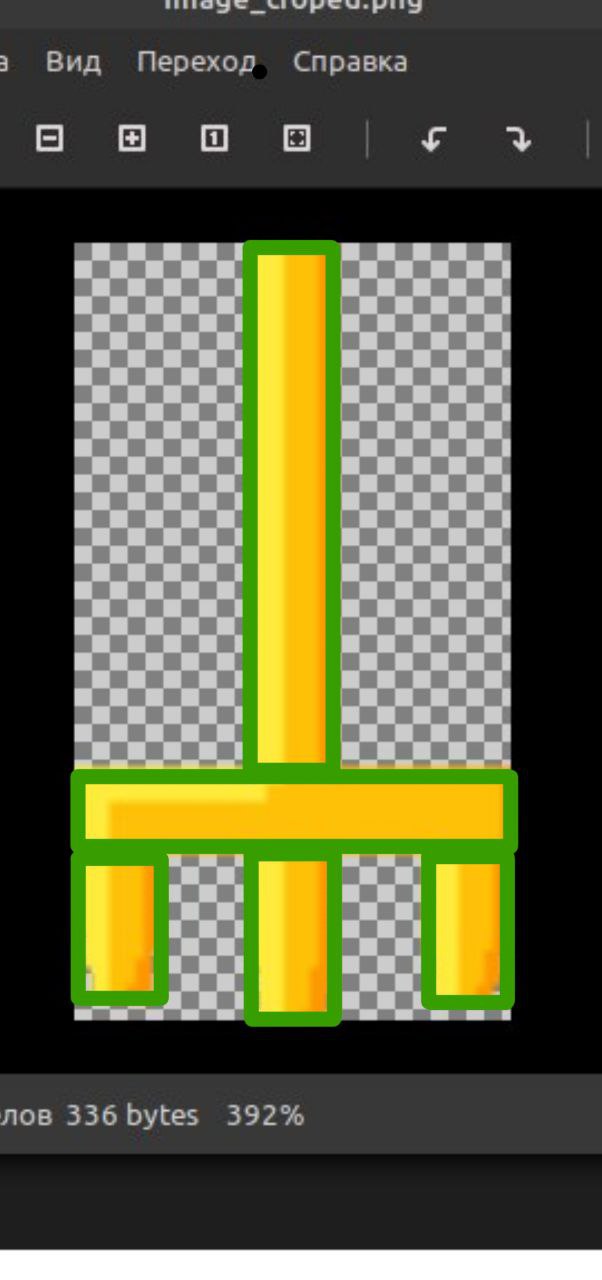


Рисунок 1 – фигура, разбитая на прямоугольники

Плюсы:

1) Эффективность при работе с фигурами, которые легко разбить на прямоугольники

2) Эффективность при работе с одинаковыми фигурами

Минусы:

1) Невозможность решения с фигурами, которые трудно разбить на составляющие. (Такие фигуры разбиваются на очень маленькие. Работа программы с такими фигурами может длиться тысячелетиями) (рис 2)

2) Низкая скорость работы, когда подаются разные фигуры

3) Время работы может сильно колебаться из-за одной фигуры



Рисунок 2 – фигура сложная для разбиения

**2. «Большой конструктор»:**

В этом варианте фигура не разбивается, а наоборот, вписывается в прямоугольник. И программа уже работает с прямоугольниками.

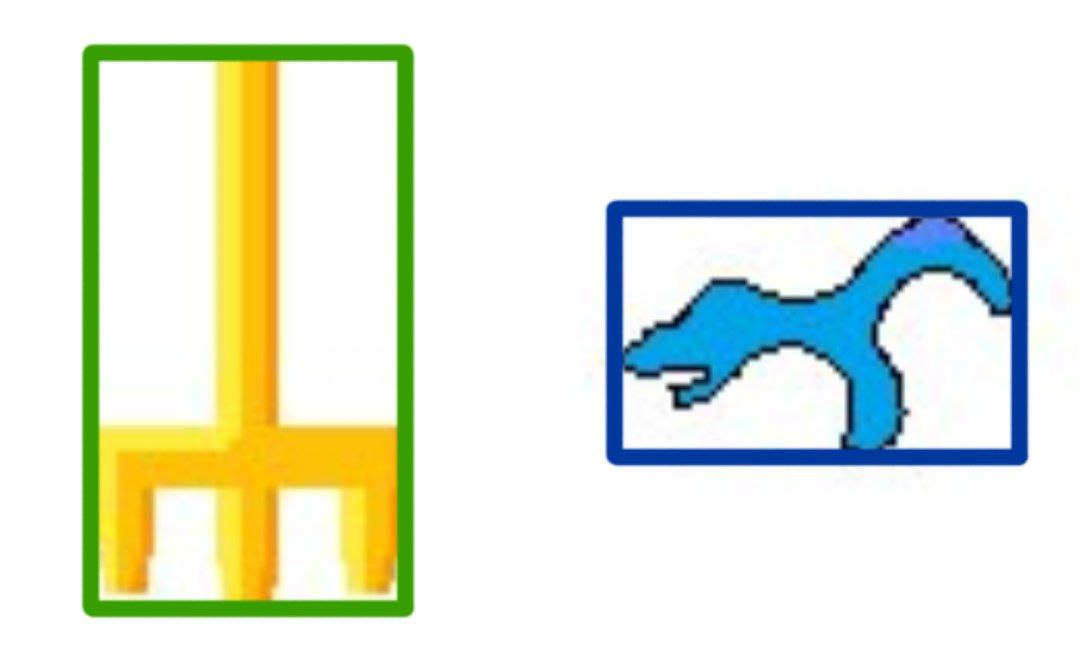


Рисунок 3 – вписанные в прямоугольник фигуры

Плюсы:

1) В отличие от «маленького конструктора» время работы не зависит от мудрености фигуры

2) Хорошее качество при работе с фигурами, похожими на прямоугольник

Минусы:

1. Низкое качество с невыпуклыми фигурами (рис 4)
2. Низкая скорость при работе с большим количеством фигур



Рисунок 4 –невыпуклая фигура, вписанная в прямоугольник

**3. “Грязный стол”**

Здесь нет какого-либо адекватного алгоритма. Просто случайное наложение необходимого количества фигур на поверхность. Как если бы мы бездумно бросали фигуры на стол.

Плюсы:

1) Простота создания

Минусы:

1) Возможность наложения фигур друг на друга

2) Неплотное расположение = плохое качество

**4. “Тетрис”**

Этот вариант наследуется от «Грязного стола», но тут уже есть алгоритм улучшения качества. Как он работает:

Фигура накладывается на поверхность в самом низу в случайном месте (y = min, x = random)

Если есть наложение, фигура смещается вверх на единицу (пример: рис 5)

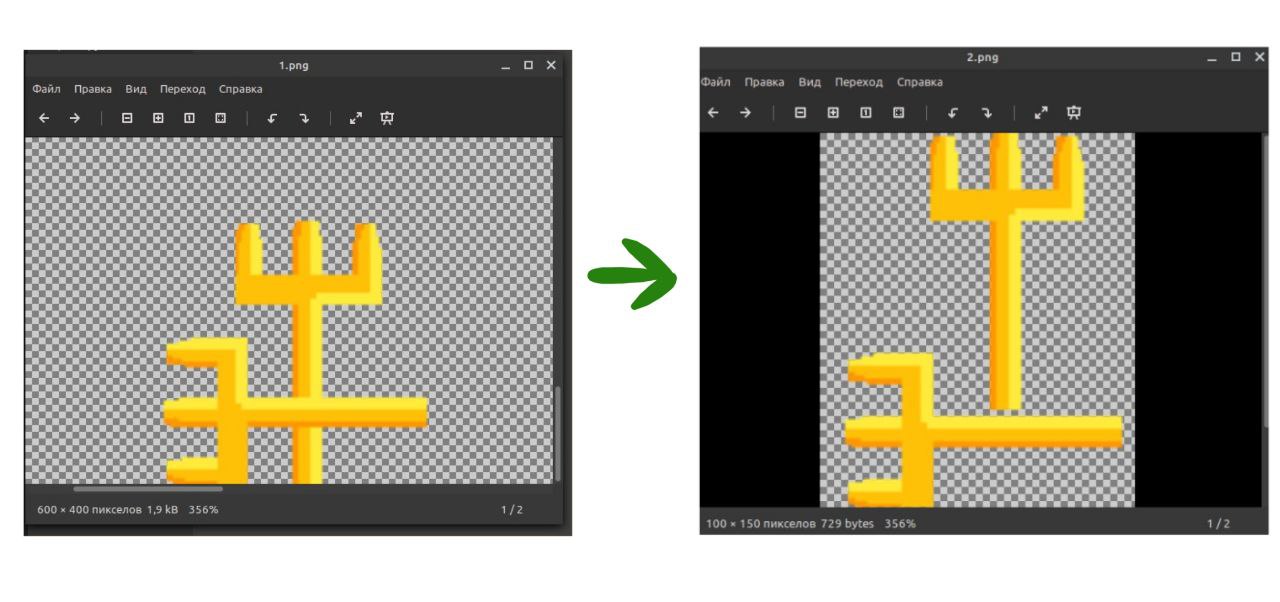


Рисунок 5 – смещение фигуры при наложении

Плюсы:

1) Хорошее качество при работе с небольшим количеством фигур (до 10) (пример - рис 6)

2) Мудреность фигуры не влияет на качество, как в «Конструкторах»

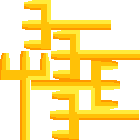


Рисунок 6 – пример результата работы

Минусы:

1. Качество падает при увеличении количества фигур

**5. “+ эволюция”**

Продолжаем прокачку существующих вариантов. Здесь добавляется генетический алгоритм. Как он работает?

1) «Поколение 0»

Создается несколько маленьких решений с помощью алгоритма «тетрис»

2) «Бои»

Попарно сравнивается качество результатов. Критерий сравнения – количество пустых пикселей. Чем их меньше, тем качество выше. Индивид с лучшим качеством становится родителем.

3) «Потомство»

Создаются новые решения на основе двух индивидов. Потомок состоит из частей родителей. + Он с вероятностью 0.1 получает мутацию в виде смещения или поворота одной из частей.

4) Цикл повторяется: потомки сравниваются, лучшие становятся родителями.

Плюсы:

1) Генетический алгоритм очень гибкий, можно создавать разные варианты эволюции для разных типов фигур и их комбинаций

2) Скорость работы (в сравнении с «Конструкторами»)

3) Качество

Минусы:

1) Сложность работы с разными фигурами

На основе плюсов и минусов был выбран вариант «+ эволюция» на основе «Тетриса»

№ 2 Реализация

Условные обозначения:

* Малое изображение – изображение, вводимое пользователем
* Основное изображение – картинка, на которую располагаются малые, иначе говоря, холст, поверхность
* Малая матрица – массив из 0 и 1, где каждое число соответствует пикселю малого изображения (0 обозначает, что этот пиксель прозрачный, 1 – что непрозрачный)
* Основная матрица – массив из 0 и натуральных чисел, где каждое число соответствует пикселю основного изображения. 0 обозначает, что этот пиксель прозрачный. Натуральное число – что пиксель непрозрачный. Также натуральное число означает номер накладываемого изображения, начиная с 1.
* Ген. алг. – Генетический алгоритм

**План реализации:**

1. Создание версии со случайным расположением («Грязного стола»):

1) Первичная обработка изображений (Обрезка пустых краев PNG)

2) Случайные повороты и задание случайных координат по X

2. Реализация версии «Тетрис» на основе «Грязного стола»:

1) Создание малых и основной матриц. Благодаря им можно отслеживать и предотвращать наложения.

2) Реализация алгоритма смещения малого изображения при наложении на другое малое изображение.

3. Добавление в версию генетического алгоритма: «+ эволюция»:

1) Написание ген. алг. для решения задачи OneMax\*

2) Переделка под задачу раскроя

**Часть 1: «Грязный стол»**

1. В первичной обработке у картинки формата PNG обрезаются пустые края (рис 7):

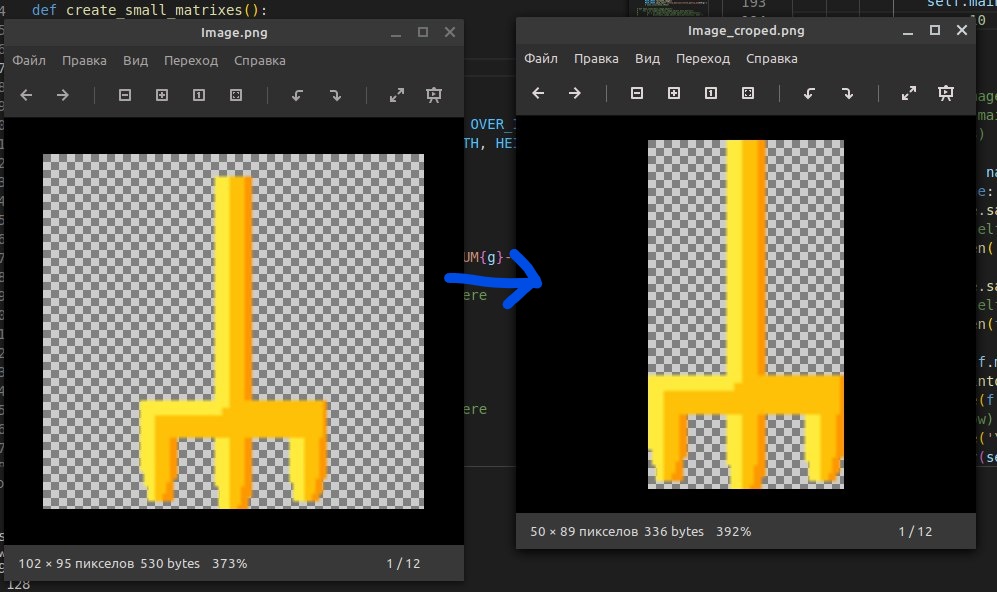


Рисунок 7

Это делается для избегания ошибок и уменьшения затрат памяти при создании матриц.

1. Далее задается случайное значение x в диапазоне от 0 до значения ширины основного изображения и случайный градус поворота от 0 до 270 с шагом 90 (выборка из значений: 0, 90, 180, 270) (рис 8)

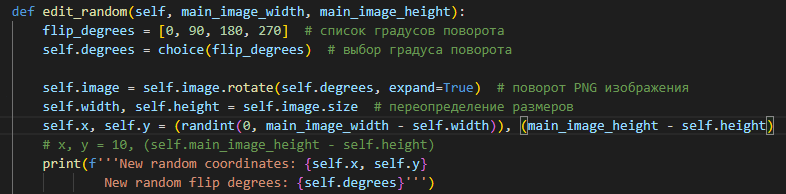


Рисунок 8 - задание случайных значений

**Часть 2: “Тетрис”**

1. Построение матриц (рис 9). Оно происходит в два этапа:
2. Создание массива, заполненного нулями с размерами, равными длине малого изображения в пикселях \* ширину малого изображения в пикселях.
3. Замена нулей на единицы в местах, где пиксель не прозрачный

Примечание: используются не стандартные массивы python, а массивы numpy. Так работа с матрицами происходит быстрее.

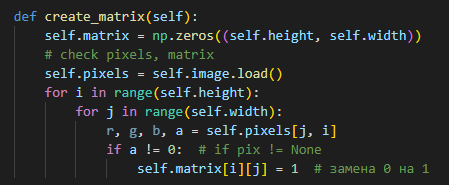
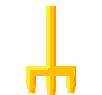


Рисунок 9 - фрагмент кода, отвечающий за создание матрицы малого изображения

В результате для изображения трезубца (рис 10) получается матрица (рис 11).



Рисунки 10, 11: картинка и ее матрица

1. При бездумном размещении изображений может случиться наложение малых изображений (рис 11)

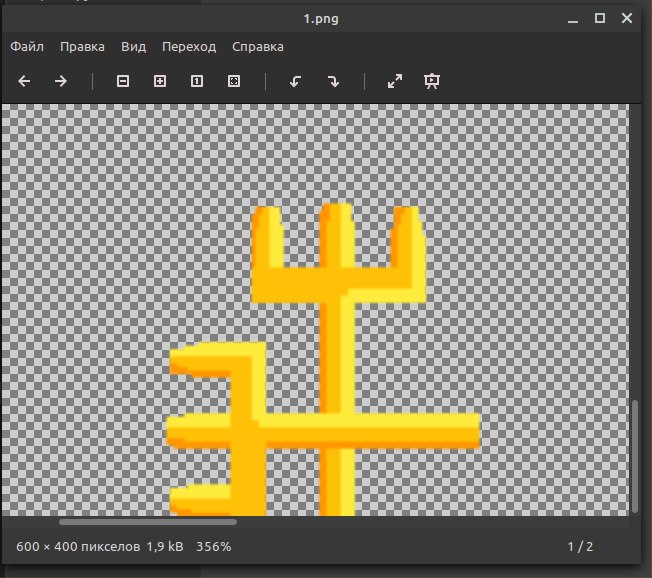


Рисунок 11 - наложение двух изображений

Чтобы этого не допустить, был написан алгоритм смещения (рис 12):

1. Пробег по матрице:
   1. Сравнение значений основной и малой матриц
   2. Если и там, и там значения отличные от 0, значит произошло наложение. Малая матрица смещается на 10 пикселей вверх.
   3. Если наложений не было, малое изображение накладывается на большое



Рисунок 12 - алгоритм отслеживания и предотвращения наложений

**Часть 3: “+ эволюция”**

1. Для понимания сути генетического алгоритма была взята задача OneMax. Суть ее следующая: случайно создаются последовательности нулей и единиц одного размера. Чем больше в последовательности единиц, тем она лучше. Задача: Написать алгоритм, создающий наилучшую последовательность.

Нам, людям, понятно, что лучшим решением будет последовательность, состоящая полностью из единиц. Однако наша задача - написать генетический алгоритм, который придет к этому решению через сравнение и создание новых последовательностей на основе имеющихся.

Для решения этой задачи была написана программа структуры ООП следующего содержания:

1. Создание класса Individ:
   1. создание случайной последовательности из N значений (индивида) (рис 13)

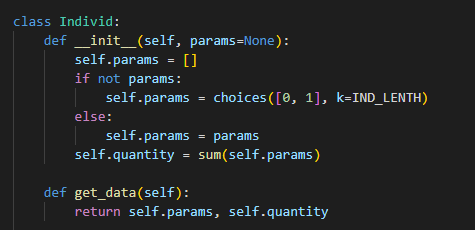


Рисунок 13 – код класса Individ

1. Создание класса Population:
   1. Создание популяции из M индивидов (популяции) (рис 14)

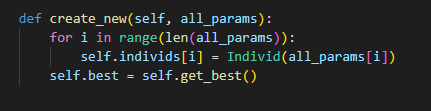


Рисунок 14 – код метода создания популяции класса Population

* 1. “Бой” между индивидами популяции (попарное сравнение, выявление лучшего из пары) (рис 15)

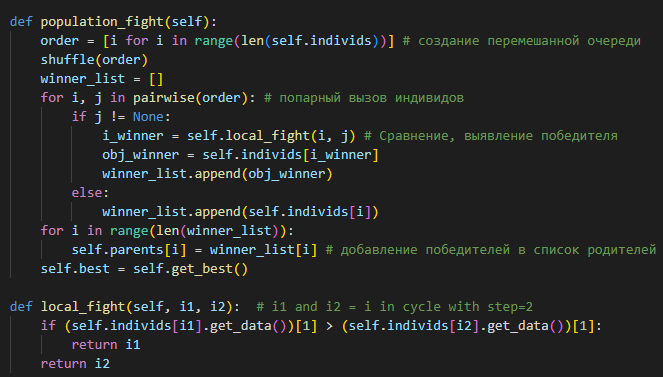


Рисунок 15 – код методов population\_fight (вся популяции) и local fight (пара)

* 1. Создание новых индивидов (потомков) из половинок двух победивших (рис 16)
  2. Цикл замыкается: “бой” начинается между потомками

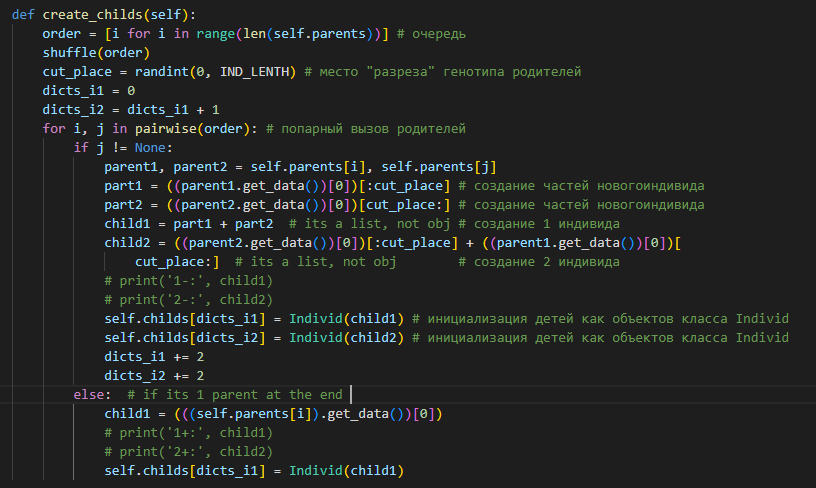


Рисунок 16 – код метода создания потомства класса Population

1. Теперь, когда принцип и тонкости работы с ген. алг. понятны, можно внедрять его в нашу задачу раскроя, в реализацию “Тетрис” со следующими изменениями:
2. Индивиды - небольшие основные матрицы, содержащие несколько малых (рис 17)

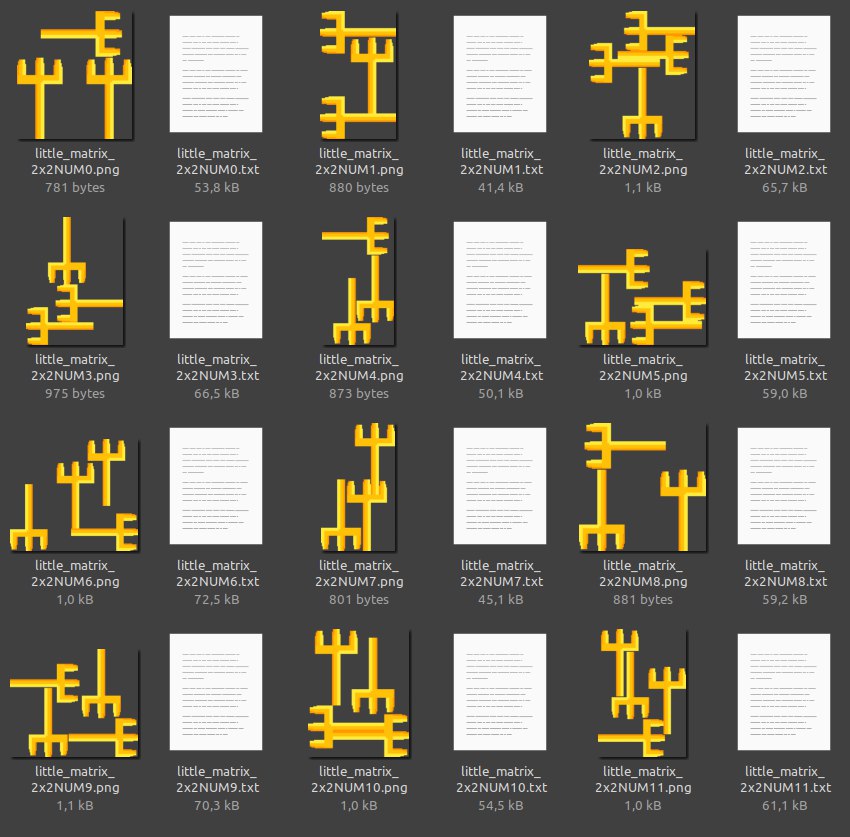


Рисунок 17 – 12 индивидов (изображение и его матрица)

1. Критерий качества - количество пустых пикселей на изображении (или количество нулей в матрице): чем их меньше, тем качество индивида выше
2. Скрещивание имеет два вида:
   1. Новый индивид состоит из частей двух других и количество его малых изображений такое же, как у родителей .
   2. Новый индивид представляет собой объединение родителей, а значит, количество его малых изображений равно сумме количества изображений родителей.

Таким образом, сначала создаются небольшие индивиды высокого качества, затем они объединяются в крупных индивидов. И уже среди них идет поиск лучшего решения (рис 18).

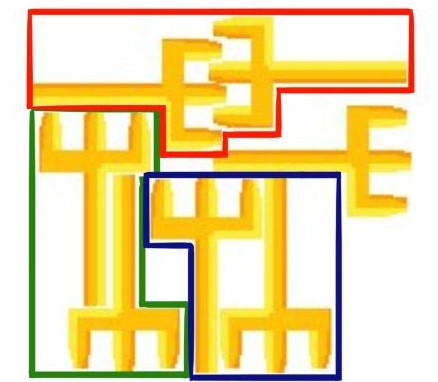


Рисунок 18 – Результат работы алгоритма: создан крупный индивид из маленьких (маленькие индивиды обведены цветными линиями)

№ 3 Исследовательская часть

**1. Направление исследования** - сравнение результатов работы программы версий “+ эволюция”, “Тетрис” и “Грязный стол”.

**2. Выбор критерия сравнения для анализа:**

Критерием качества, как и в работе генетического алгоритма, будет соотношение количество заполненных пикселей к количеству пустых / прозрачных пикселей. Чем этот коэффициент больше, тем качество выше.

Каждой версией было создано 40 решений. Решения создавались в диапазоне от 9 до 15 фигур. Для каждого решения рассчитывался коэффициент качества по формуле:

Таким образом было с помощью numpy построено 3 графика, представляющих зависимость коэффициента качества от количества фигур (N фигур) (рис 19)

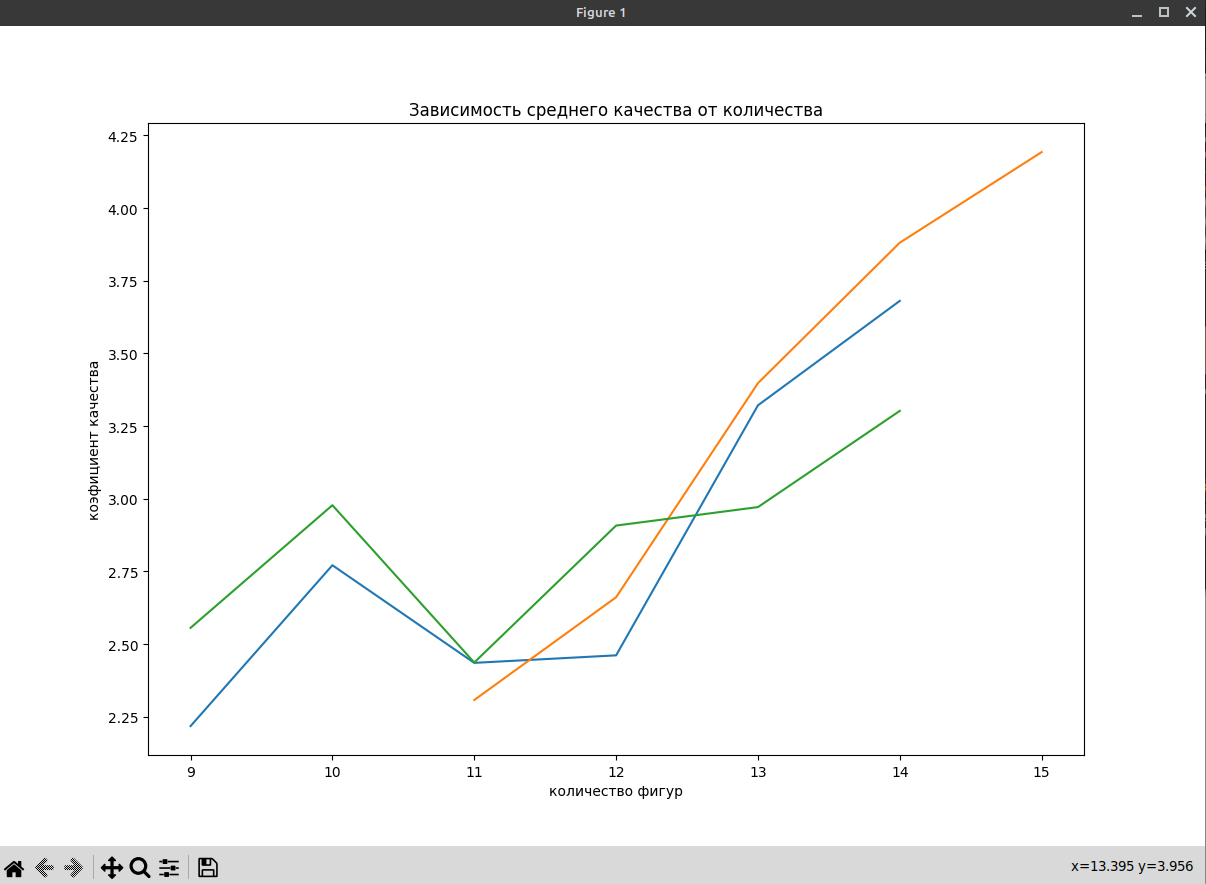


Рисунок 19 – график построенный на значениях коэффициентов качества, взятых от трех версий решения.

Примечание:

* оранжевая функция - на основе данных от версии “+ эволюция”
* зеленая функция - на основе данных от версии “Грязный стол”
* синяя функция - на основе данных от версии “Тетрис”

**Вывод по исследовательской части:**

Таким образом на графике видно, что коэффициент качества наивысший при наибольшем количестве фигур у версии “+ эволюция”. Следовательно, реализация алгоритма перемещения при наложении совместно с генетическим алгоритмом является наиболее успешной.

Вывод работы

Автором было реализовано несколько вариантов решения задачи раскроя, проанализированы результаты работ программ и выявлен наиболее успешный способ решения задачи. В итоге создано приложение для пк с понятным интерфейсом, генерирующее решение для вводимого изображения.

Список используемой литературы

1. Python Pillow

URL: <https://pythonexamples.org/python-pillow/>

1. Numpy документация

URL: <https://numpy.org/doc/>

1. ООП на Python: концепции, принципы и примеры реализации

URL: <https://proglib.io/p/python-oop>

1. Linux. От новичка к профессионалу. 7-е издание. Денис Колисниченко