

Московский Государственный Университет имени М.В.Ломоносова  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ  
КАФЕДРА МАТЕМАТИЧЕСКИХ МЕТОДОВ ПРОГНОЗИРОВАНИЯ

**Обработка изображений  
Лабораторная работа №1**

**Изучение и освоение методов обработки и сегментации изображений  
Отчет**

Авдеев Роман Артемович  
Группа 317

Март, 2023

# Оглавление

<b>1 Введение</b>	<b>2</b>
<b>2 Постановка задачи</b>	<b>2</b>
<b>3 Класс Intermediate. Задача 3</b>	<b>2</b>
3.1 Алгоритм . . . . .	3
3.1.1 Эксперименты с разными подходами . . . . .	3
3.1.2 Финальный алгоритм . . . . .	5
3.2 Программная реализация . . . . .	6
3.3 Результаты . . . . .	7
<b>4 Класс Intermediate. Задача 4</b>	<b>8</b>
4.1 Алгоритм . . . . .	8
4.2 Программная реализация . . . . .	8
4.3 Результаты . . . . .	10
<b>5 Класс Beginner</b>	<b>11</b>
5.1 Задача 1 . . . . .	11
5.2 Задача 2 . . . . .	11
<b>6 Вывод</b>	<b>13</b>

# 1 Введение

В данном задании была поставлена задача разработать и реализовать программу для работы с изображениями фишек игрового набора Тантрикс, обеспечивающую:

1. ввод и отображение на экране изображений в формате BMP
2. сегментацию изображений на основе точечных и пространственных преобразований
3. генерацию признаковых описаний фишек
4. классификацию отдельных фишек и групп фишек

Каждая фишечка представляет собой правильный шестиугольник черного цвета, на котором изображены сегменты трех линий синего, красного и желтого цвета. Для отладки и тестирования разработанного алгоритма были предоставлены различные наборы изображений, содержащие описанные фишечки: изображения с одной фишечкой ('Single\_\*.bmp'), изображения, содержащие группы фишечек ('Group\_\*.bmp').

Было решено выполнять класс задач Intermediate, но в процессе решения были также реализованы задачи из класса Beginner.

## 2 Постановка задачи

**Intermediate.** Разработать и реализовать алгоритм, способный:

1. определить номер фишечки (вход - файл типа 'Single\_\*.bmp')
2. определить расположение и номера всех фишечек в кадре (вход - файл типа 'Group\_\*.bmp')

**Beginner (в процессе выполнения Intermediate).** Разработать и реализовать алгоритм, способный:

1. определить количество фишечек на изображении (вход - файл типа 'Group\_\*.bmp')
2. определить тип и цвет линий на фишечке (вход - файл типа 'Single\_\*.bmp')

## 3 Класс Intermediate. Задача 3

Рассмотрим набор фишечек ('Dozen\_0.bmp') подробнее. Заметим, что фишечки отличаются набором линий: разные цвета и формы.

Ниже представлены 10 типов выданных фишечек:

1. фишечка - длинная синяя, длинная красная, короткая желтая
2. фишечка - прямая синяя, короткая красная, короткая желтая
3. фишечка - короткая синяя, короткая красная, короткая желтая
4. фишечка - прямая синяя, длинная красная, длинная желтая
5. фишечка - короткая синяя, прямая красная, короткая желтая
6. фишечка - длинная синяя, длинная красная, прямая желтая
7. фишечка - короткая синяя, длинная красная, длинная желтая
8. фишечка - короткая синяя, длинная красная, длинная желтая
9. фишечка - длинная синяя, прямая красная, длинная желтая
10. фишечка - длинная синяя, длинная красная, короткая желтая

## 3.1 Алгоритм

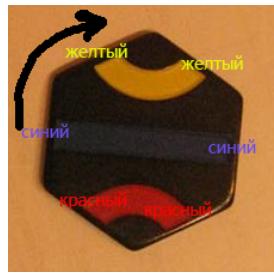
### 3.1.1 Эксперименты с разными подходами

Первой идеей реализации алгоритма детекции было подсчитать площадь каждого из цветов, сравнить их и на основании этих признаков определить тип фишки. Но тут возникло сразу несколько сложностей:

1. для точного подсчета площади нужна идеальная (или близкая к идеальной) детекция цветов; но выделение цвета не всегда получается хорошо, этому могут помешать засветы на фотографии, острый угол, с которого сделан снимок и т.д.
2. некоторые линии пересекаются, возникают пропуски; это можно учесть, заметив, что всего на данных десяти фишках наблюдается 4 принципиально разные ситуации: 2 длинные линии + 1 короткая (одно пересечение), 2 короткие + 1 прямая (ноль пересечений), 3 короткие (ноль пересечений), 2 длинные + 1 прямая (два пересечения); но тут все равно возникает неопределенность во время классификации

При первом же тестировании указанные выше недостатки показали ненадежность такого алгоритма, поэтому было решено начать разработку нового.

Следующей идеей стало выделение некоторой области в середине каждого ребра фишки и фиксация цвета линии, примыкающей к данному ребру. Таким способом можно получить схему вида: (синий, желтый, желтый, синий, красный, красный) - соответствует фишке №2. Соседние лейблы цвета соответствуют короткой линии,

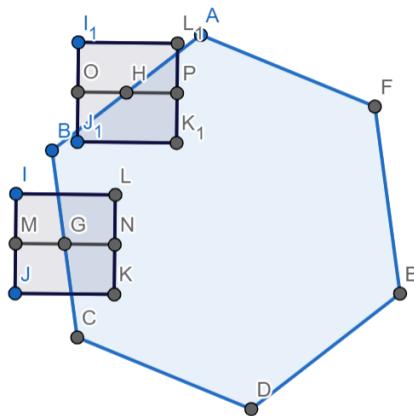


одинарный промежуток - длинной линии, двойной промежуток - прямой линии. Зная координаты вершин (которые можно найти, пользуясь методами findContours и approxPolyDP из библиотеки OpenCV), мы можем легко вычислить координату середины, а также длину ребра как евклидово расстояние:

$$\text{edge} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Главная мысль заключалась в том, чтобы легко и быстро выделить область вокруг середины ребра, тут подошло выделение квадрата с координатами  $[(middle_x - 0.5 * edge, middle_y - 0.5 * edge), (middle_x - 0.5 * edge, middle_y + 0.5 * edge), (middle_x + 0.5 * edge, middle_y + 0.5 * edge), (middle_x + 0.5 * edge, middle_y - 0.5 * edge)]$ .

Рисунок представлен ниже:



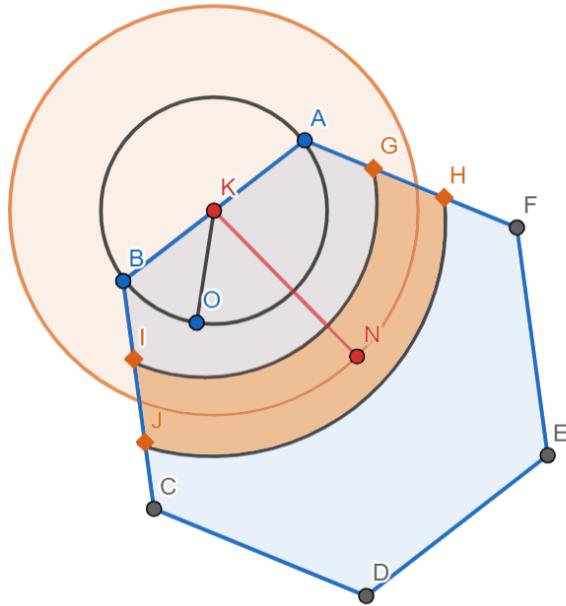
Где G и H - середины ребер BC и BA соответственно, а стороны квадратов  $ILKJ$  и  $I_1L_1K_1J_1$  равны половине ребра **правильного** шестиугольника.

Проблемой данного подхода стало то, что при разном наклоне граней шестиугольника выделенный квадрат захватывает разную площадь внутри. И тогда при неточной аппроксимации и выделении сторон фишк можно не зафиксировать нужный цвет или зафиксировать, но лишний. Поэтому, было решено продолжить эксперименты с методами по выделению типа и цвета линий.

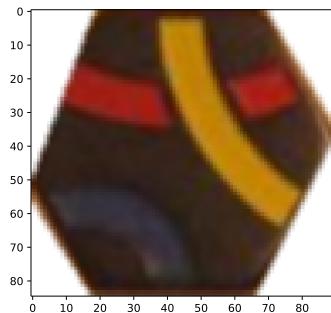
Логичным усовершенствованием предыдущей идеи стали окружности (полуокружности).

Этот подход решал многие проблемы прошлых стратегий, но здесь возникала новая сложность: при неправильном выборе радиуса полуокружности можно было считать дугу другого цвета, проходящую от одного соседнего ребра к другому.

Данный случай представлен ниже на рисунке:



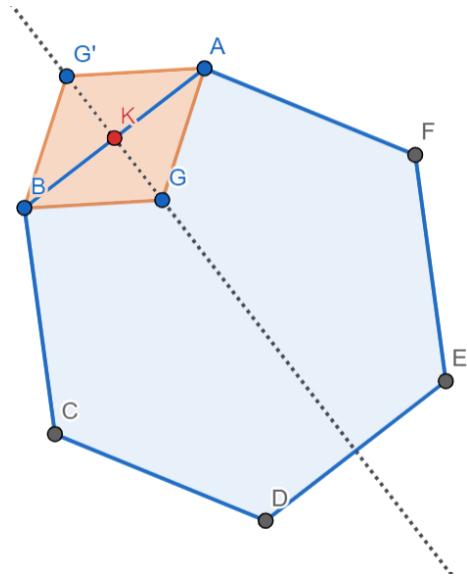
Здесь GHJI - линия некоторого цвета, выделяем область вокруг середины ребра BA - строим окружность с центром в точке K(середина BA). Рассматривая ребро BA, мы должны фиксировать только линии, подходящие к ребру BA, а выбрав окружность с большим радиусом (например, KN), мы "поймаем" ненужный фрагмент оранжевой линии. Идеальным радиусом в данном случае будет радиус = половине ребра BA. Но при выделении фишк на изображении (а впоследствии и групп фишек) не всегда получается правильный шестиугольник, что может существенно затруднить выбор оптимального радиуса окружности. Пример ниже:



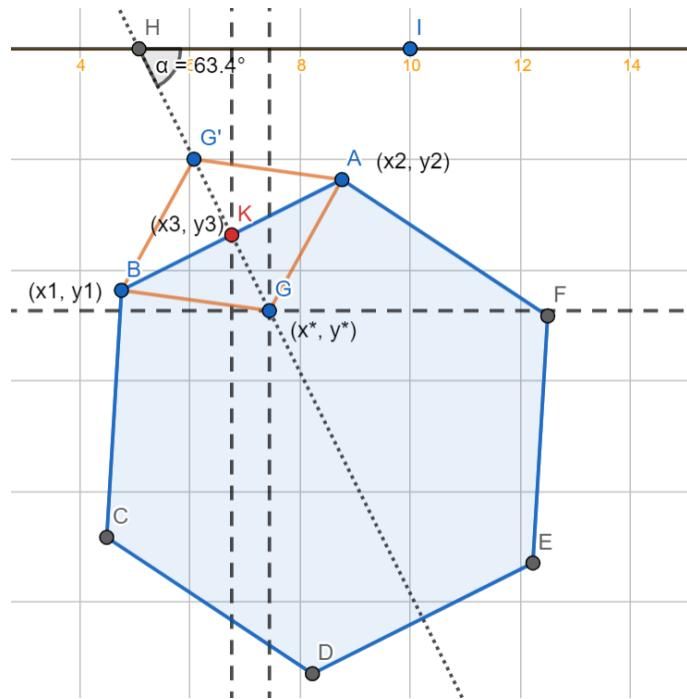
(Фотография размытая, так как к ней был применен GaussianBlur)

### 3.1.2 Финальный алгоритм

Финальным алгоритмом детекции цвета стало выделение треугольников на каждом ребре (пар треугольников/ромбов). Линия касается ребра в его центре, поэтому весь "цвет" линии сосредоточен в середине, незачем выделять область, близкую к краю ребра. Даже если аппроксимация будет неточная и центр сместится, треугольник все равно "захватит" нужный цвет, и при этом не будет случайно выделять другие цвета (в отличие от полуокружности). Рассматриваем ребро BA, построим к нему серединный перпендикуляр, содержащий точки G и G'. Для простоты



будем строить два симметричных треугольника для каждого ребра - например, нижний треугольник BAG нужен для выделения цвета на BA, а треугольник BAG' - для ребра DE. Координаты точек G и G' можно найти, пользуясь простыми вычислительными формулами. По двум вершинам ребра можно построить уравнение



прямой, содержащей это ребро. Затем найти уравнение перпендикулярной прямой → знаем угол наклона, а

тогда из прямоугольных треугольников найдем координаты  $(x^*, y^*)$  как  $x^* = x_3 + KG * \cos(\alpha)$  и  $y^* = y_3 + tg(\alpha) * (x^* - x_3)$ . Расстояние  $KG$  было решено выбрать основываясь на длине ребер, но так как при аппроксимации они могут получиться разными, вычисляется средняя длина ребра, и  $KG = \frac{edge_{average}}{3}$ .

Строя такие пары треугольников на каждом ребре, мы получаем величины площадей, которые занимает конкретный цвет у конкретного ребра. Затем выбираем две наибольшие площади, и это и будут именно те ребра, которых касается линия. Такие признаки могут дать описание изображенной фишке (**Beginner. Задание 2**), но однозначно классифицировать ее не получится. Только 6 из 10 фишек можно опознать по типу и цвету линий; фишке 1 и 10, 7 и 8 очень похожи, в них надо дополнительно определять площадь занимаемого цвета. Например, на фишке 1 больше красного цвета, чем синего, на фишке 10 наоборот. На 7 фишке больше красного, а на 8 больше желтого.

Итоговый алгоритм выглядит следующим образом:

1. вырезаем фишку из изображения, находим координаты вершин и середин ребер
2. для каждого ребра строим треугольники, внутри которых проверяем наличие определенного цвета
3. получаем описание фишки (длина линий и их цвет); в случае однозначной классификации - выводим лейбл, в случае фишек 1 и 10, 7 и 8 - считаем общую площадь каждого цвета и получаем дополнительный признак, который позволяет однозначно классифицировать оставшиеся фишки

## 3.2 Программная реализация

Алгоритм реализован на языке Python 3.9 (Jupyter notebook) с использованием библиотек numpy, OpenCV, matplotlib. Наиболее полезными оказались функции из OpenCV, отвечающие за морфологические преобразования (dilate, erode, morphologyEx(MORPH-OPEN | MORPH-CLOSE), medianBlur, GaussianBlur, threshold, bitwise-not), а также функции, позволяющие работать с аппроксимированными фишками и их площадями (findContours, drawContours, contourArea, Canny, approxPolyDP, arcLength, countNonZero).

Перед применением алгоритма изображения проходят предобработку - перевод в RGB/оттенки серого (cvtColor), размытие для частичного устранения бликов, засветов и другого "шума" (GaussianBlur), бинаризация и инверсия (threshold, bitwise-not). Для выделения каждого цвета по отдельности использовался метод inRange. Здесь сложность заключалась в корректном подборе нижнего и верхнего порогов для детекции цвета.

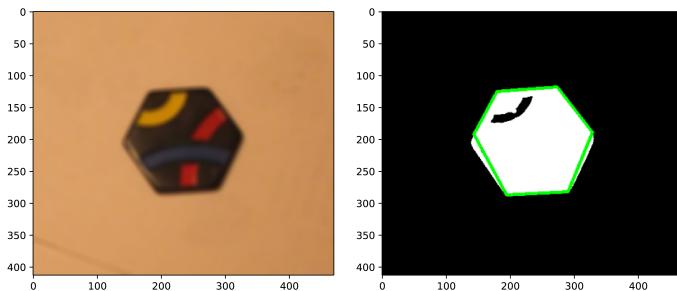
Весь код находится в едином Jupyter notebook, где поделен на классы **Intermediate/Beginner** и **Задачи (1, 2, 3, 4)**. Для каждой задачи есть раздел с реализованными методами предобработки, сегментации и детекции, раздел демонстрации работы на всех предоставленных изображениях (с пояснениями), а также раздел для удобной проверки работоспособности программы.

Скриншоты промежуточных шагов работы алгоритма:

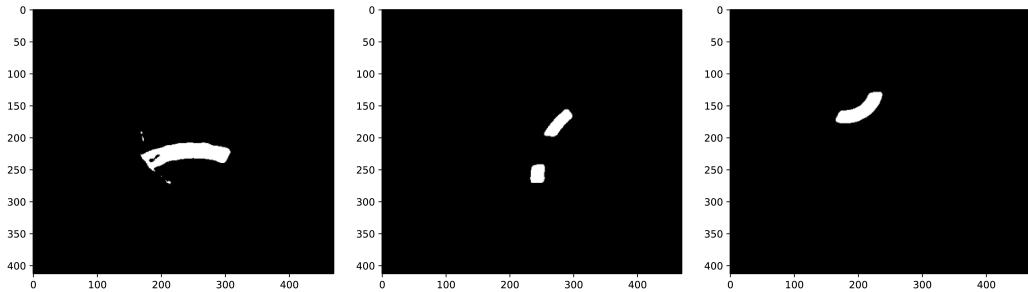
Дано изображение `Single_1.bmp`:



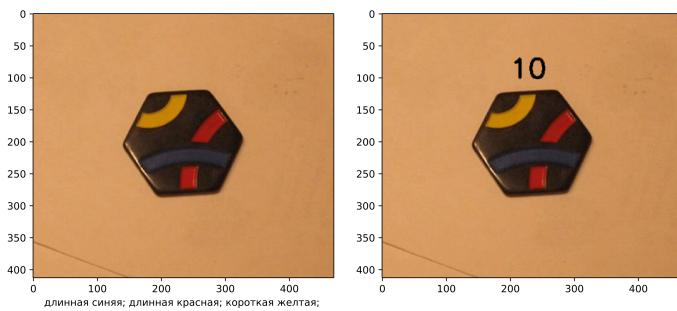
Размытие и аппроксимация:



Детекция синего, красного и желтого цветов:



Генерация описания и ответ:

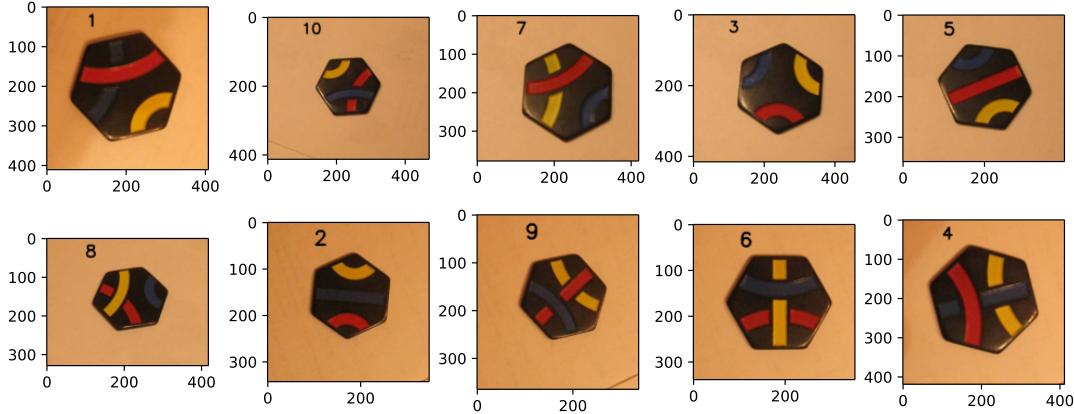


### 3.3 Результаты

Результат работы программы по классификации фишек на изображениях типа Single\_\*.bmp:  
Образец:



Ответ алгоритма:



Видим, что все фишкы распознаны верно.

## 4 Класс Intermediate. Задача 4

### 4.1 Алгоритм

Алгоритм работы с отдельными фишками в Задаче 4 точно такой же, как и в предыдущем задании. Но для того, чтобы работать с отдельными фишками, их сначала нужно распознать среди других фишек и шума, верно аппроксимировать и только потом подать на вход алгоритма, описанного в Задаче 3.

Первым делом были удалены сильные тени (особенно на `Group_1.bmp` и `Group_4.bmp`), которые могли помешать корректной бинаризации. Затем проходила процедура аппроксимации фишек и отсев шума (например, удаление на `Group_6.bmp` рисунков на поверхности стола). Также на данном этапе важно было выставить нижнюю границу площади объекта, который мы можем считать за фишку. На упомянутой `Group_6.bmp` в углу были небольшие изображения ягод и другие побочные рисунки. При аппроксимации они могли быть приняты за фишку, но, выставив нижний порог площади объекта и проверив, что объект является шестиугольником, мы исключили такую вероятность. Далее надо было вырезать обнаруженные фишкы, для этого использовались маски, созданные под каждую конкретную фишку. Таким образом, мы получили набор из изображений, на каждом из которых была только одна фишка, то есть свели задачу к предыдущей.

### 4.2 Программная реализация

Для удаления теней использовалась дилатация с  $\text{kernel}=(11, 11)$  и размытие. Экспериментировал с разными значениями точности аппроксимации. Это было необходимо, чтобы исключить проблему сильной несимметричности ребер, описанную в разделе **Эксперименты с разными подходами**. Для выделения фишек из изображения, а затем для приближения одиночной фишке использовались  $\text{epsilon} = 0.035$  и  $\text{epsilon} = 0.042$  соответственно.

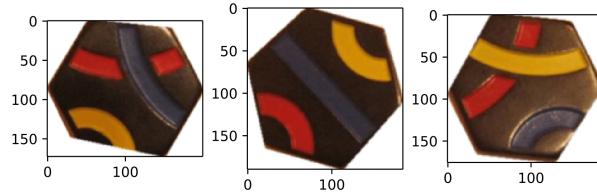
Важный момент: при аппроксимации фишек на исходном изображении мы получали одни координаты вершин и середин ребер, но на вырезанной одиночной фишке эти координаты надо было пересчитывать для корректного выделения областей при поиске цвета. Но исходные координаты все равно оказались нужны для печати отметки класса при выводе ответа.

Пример работы для `Group_1.bmp`:

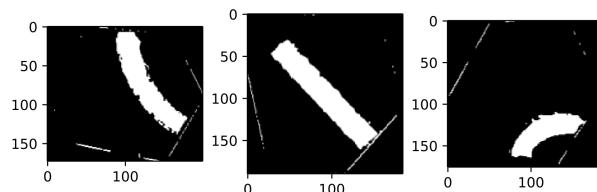
Входное изображение:



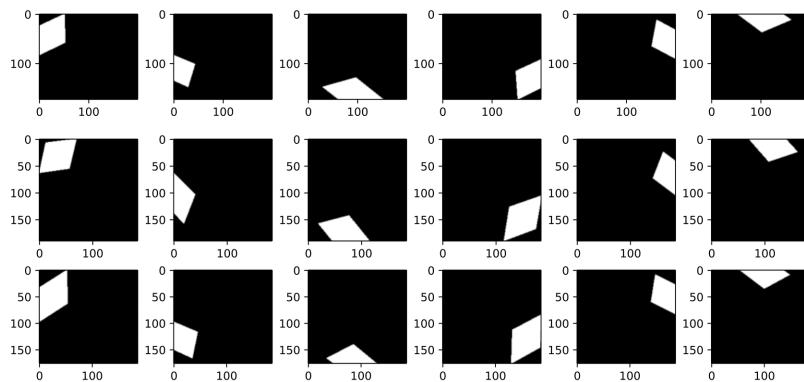
Вырезанные фишки:



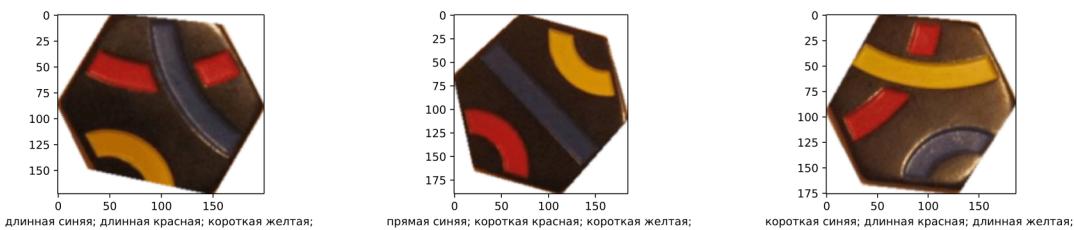
Выделение синего цвета (для красного и желтого аналогично):



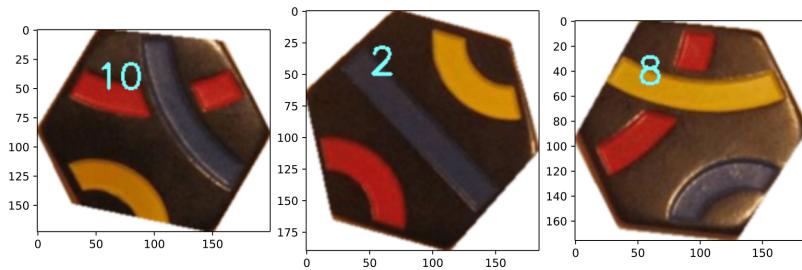
Построение треугольных масок для каждой фишкой:



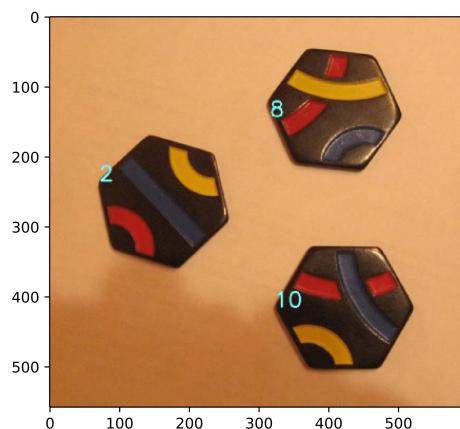
Генерация описаний:



Определение типов фишек:



Отображение типов на исходном изображении:



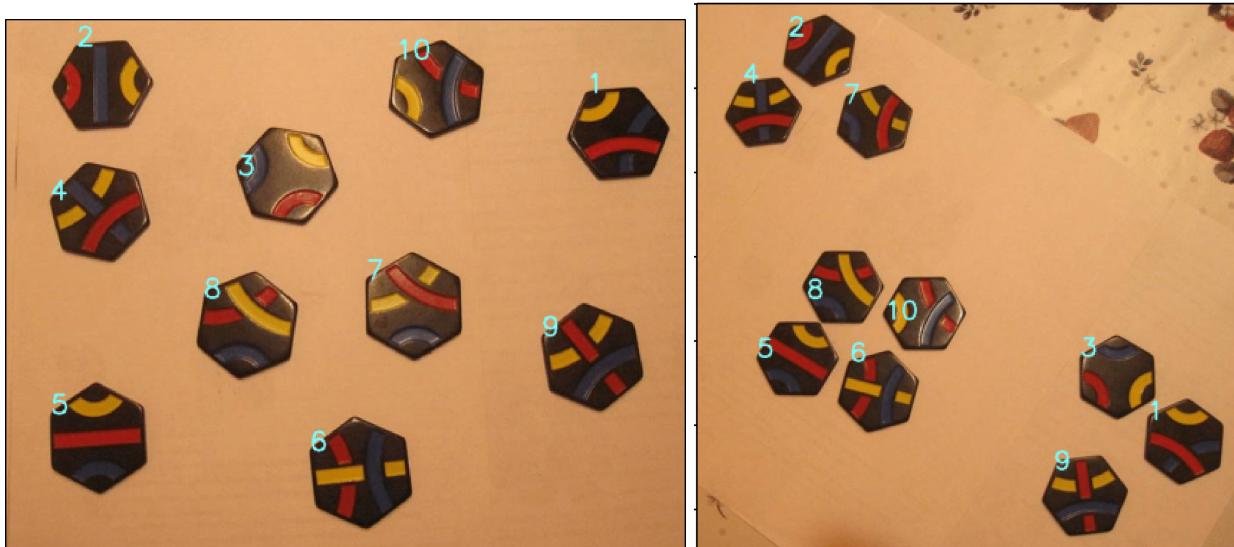
### 4.3 Результаты

С тестированием и подробным описанием работы алгоритма Вы можете ознакомиться в Jupyter notebook 'Lab1\_Avdeev\_317.ipynb'. Там же представлены ответы для всех тестовых изображений.

Ниже я прикреплю, на мой взгляд, наиболее сложные случаи:  
Образец:



Group\_5.bmp и Group\_6.bmp:



Классификация для Dozen\_0.bmp



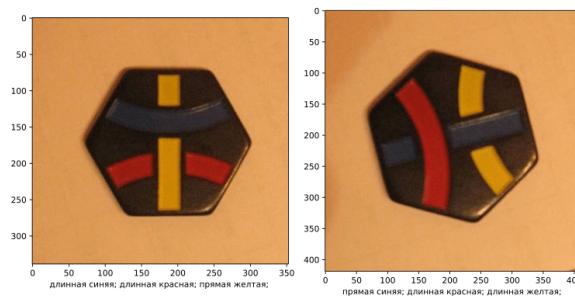
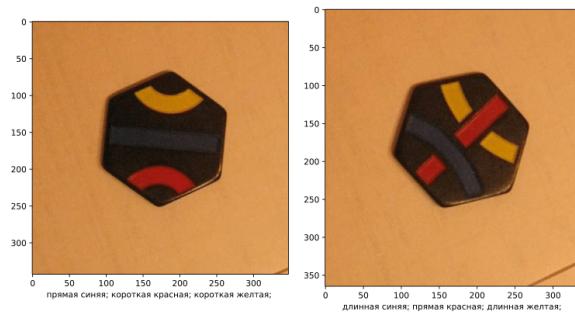
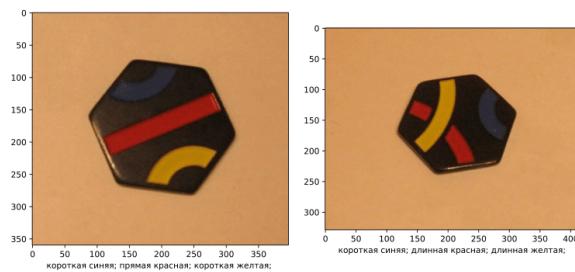
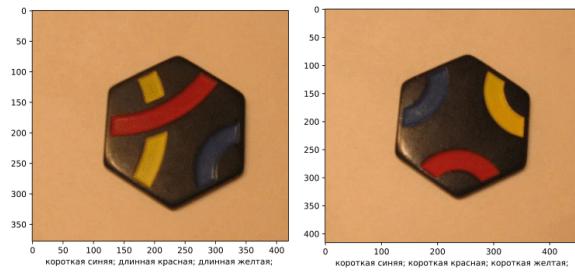
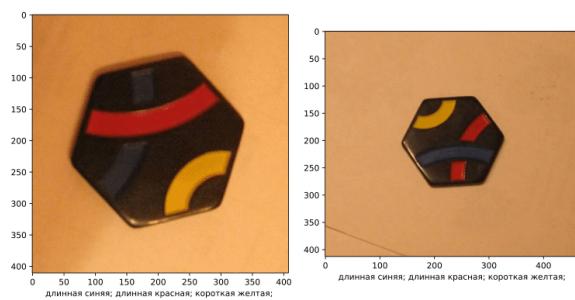
## 5 Класс Beginner

В ходе работы над задачами уровня Intermediate выполнил задачи Beginner. Алгоритм работы точно такой же, поэтому в данном разделе прикреплю ответы для Задачи 1 и Задачи 2.

### 5.1 Задача 1

```
for i in range(len(test_images)):
    coordinates = approx_figures_groups(test_images[i])
    print('На изображении ', image_names[i], ' находится ', len(coordinates), ' фишк(ек)')
✓ 0.2s
На изображении Group_1.bmp находится 3 фишк(ек)
На изображении Group_2.bmp находится 3 фишк(ек)
На изображении Group_3.bmp находится 3 фишк(ек)
На изображении Group_4.bmp находится 10 фишк(ек)
На изображении Group_5.bmp находится 10 фишк(ек)
На изображении Group_6.bmp находится 10 фишк(ек)
```

### 5.2 Задача 2



## **6 Вывод**

В ходе работы над заданием были реализованы методы выделения фигур, изменения изображений с помощью морфологических преобразований, детекции цветов, подсчета площадей, генерации признаковых описаний и классификации предложенных фишек. Удалось успешно выполнить задачи категории Intermediate и Beginner: определить номер фишк, расположение и номера всех фишек в кадре, количество фишек на изображении, тип и цвет линий на фишке. Кроме того, был изучен функционал библиотеки OpenCV. Получен новый алгоритм детекции типов и цветов линий (с помощью пар треугольников/ромбов), проанализированы недостатки других методов.