# Project Report

# Goal

The goal of this project was to enable a computer program to play the Game of Sets. Given to it an image of the card layout, the program would identify the shape, number, color and shading on each card and show the possible sets in the image.

Sets is a card game where 12 cards from the deck are randomly chosen and set on the table. The players must find a group of 3 cards where all the cards for each of the 4 features have either the same feature or all different features. Ex: Card 1 (Purple, squiggly, solid, 1), Card 2 (Red, squiggly, solid, 2) and Card 3 (Green, squiggly, solid, 3) is a set since all cards have the same shape (squiggly), have the same shaping (solid) but have unique colors (Purple, Red and Green) and unique shape count (1,2 and 3).

# Sample Image

A picture containing text

Description automatically generated

# Issues with Input Images

As we can see in the sample image above, the cards are put on a black cloth, but the image has a lot of other information that we should be ignoring. The images have, part of the table, the card deck and some images also have the background beyond the table. Since we are only interested in the 12 cards on the black cloth, we had to do some pre-processing to ignore those details.

There is also some noise in the image such as lint on the cloth. This was removed using an averaging filter, particularly a Gaussian filter.

# Overview of Steps

1. Remove CRUD from the image
2. Ignore the table, the background beyond the table and the deck of cards from the images.
3. Identify and separate the individual cards on the table
4. Count the shape on each card
5. Classify the color of each card
6. Identify the shape
7. Identify the shading inside the shape for each card
8. Using the count, shape, shading and color, identify the possible sets in the image.

# Detailed Steps

## Remove CRUD from the images:

Since most of the noise that we are really concerned about getting rid of is the lint on the black cloth and the texture on the table, we use a Gaussian blur filter that blurs these lint and texture giving us a more uniform surface. Also, since we don’t want to lose the lines on the card (the stripe texture), we had to come up with a Gaussian filter using the method of adjustments to blur the noise but keep the stripes in the card intact.

## Ignore the table, background and deck of cards:

We observed all the images and notices that all these objects we want to ignore are touching the edge of the image. We tried taking advantage of that fact. To start with, we first binarized the blurred image and using morphology tried getting rid of all the “holes” in the image so that the only regions visible would be the cards, the table, the deck and the background. Then using bwlabel, we divided the image into regions. We then looked at all the regions that bwlabel found, iterated on them and kept a tab of all the regions that have the row or column as 0 or the height/width of the image indicating that their regions touch the edge of the image. This cell array of the region ID’s we are ignoring will then help us when separating the individual cards from the image.

## Identify and Separate Individual Cards:

From the previous step, we already have the cards divided into regions. We iterate on all the regions and ignore those that we have in the cell array which denote the table, the deck and the background regions. On all the other regions (i.e., the individual cards) we draw a convex hull around it to show on the subplot figure. We also use the row and the columns returned by the convhul function and find the minimum and the maximum row and column value and using these row and column values, crop the original image to get an RGB version of the original image. These cropped versions are then sent to the classification functions to identify the shape, shading, color and the count.

## Count the shape on each card:

The function that counts the number of shapes on a card takes in the RGB image on an individual card. It then starts by using the red channel and binarizes it to get a binary image and then using morphological operation, we get rid of the stripped lines in the image if there are any and then using bwlabel again divide the card into regions. This time, the regions would be the shapes and part of the black background on which the cards are kept. Counting the number of regions and subtracting 1 (to ignore the black background region) then gives us the count of the shape in that card. Since we are using the red channel initially, the red colored cards will give a zero count. Before ending the function, we check if the count is zero and if so, we then perform the exact same operations but this time using the green channel.

## Classify color of each card:

TBD

## Identify the shape:

TBD

## Identify the shading inside the shape for each card:

We are again using Morphology for this. This time, instead of getting rid of the stripes in the shapes as we did for when counting the number of shapes, we keep those stripes, but we get rid of the “edges” between the card and the background, essentially combining those regions into one. We then count the number of regions in the inverted binary image. If the number of regions is 1, it means that the card has solid shapes since the only detected region would be the white part of the card. If the region count is below a threshold (7 in our case) but not 1, that indicates that the shapes are outlines and if it is more than that threshold of seven, this indicates that the shapes have stripes in them which divided each shape into numerous smaller regions.

## Using the count, shape, shading and color, identify the possible sets in an image:

Once we are done iterating on all the individual cards and collecting their properties in a cell array, we send this array to a function that loops over it and compares 3 cards at a time. We then keep track of how many features of the three cards we are looking at satisfy the rules of the game (each feature is the same across all three cards or are all unique). If the count is 4 i.e., the shape, color, count and shading all satisfy the rules of making a set, we keep track of those 3 cards. In the end once all the cards are looked at, we print out all the possible sets in the image to the console and end out program.

# Results

TBD