Documentation for Assignment 1 – Automatic Score Calculator for the Qwirkle Game

The file **solution.py** implements a complete solution for image processing and score calculation for the game **Qwirkle Connect**.

Steps performed by the code for each image (round) in a game:

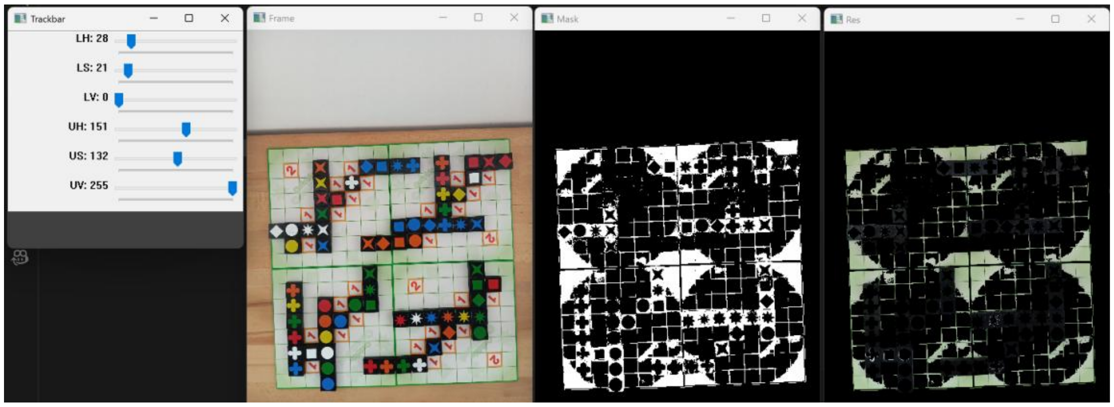
1. Extract the board from the image
2. Detect the positions with bonus points (only in the first image)
3. Detect the pieces on the board
   - 3.1. Detect the shapes of the pieces
   - 3.2. Detect the colors of the pieces
4. Calculate the score
5. Write the result to a .txt file

1. Extracting the Board from the Image

Before obtaining information about the pieces on the board and processing them, a clear view of the board is needed. This is achieved by calling the function **extract_square**.

This function resizes the original image and applies an HSV mask with lower (28, 21, 0) and upper (151, 132, 255) values, which highlights the contour of the board and removes the background, as in Figure 1. Then, the function applies morphological operations on the resulting image to better define the contours and remove remaining noise. Finally, it determines the corners of the board and extracts the board from the original image.

The resulting image is 1280×1280 pixels and will be later divided into 32 rows and 32 columns, each "square" having 80×80 pixels.



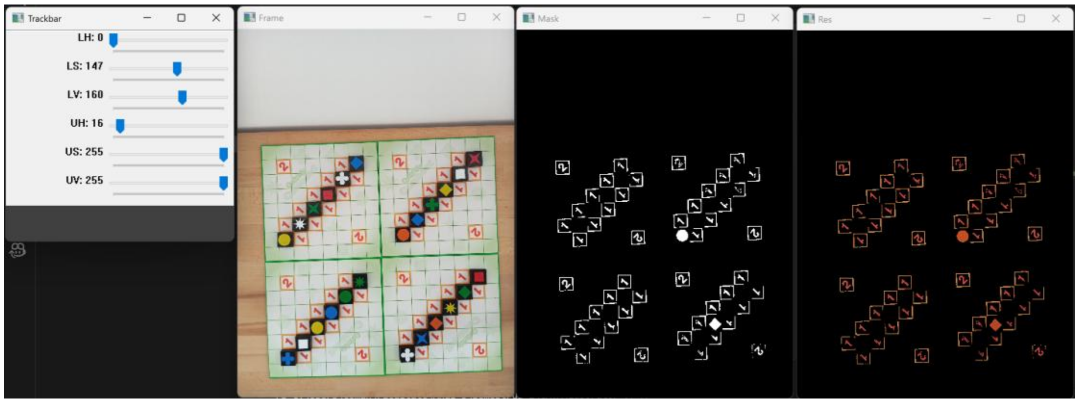
LH: 28
LS: 21
LV: 0
UH: 151
US: 132
UV: 255

Figure 1

2. Detecting Bonus Point Positions

Now that the image has been cropped to include only the board, each small square can be scanned.

Bonus points are detected only in images ending in "_00.jpg", because bonus positions are the same in every image of a game, so they do not need to be searched repeatedly. To check whether a square contains a bonus point, an HSV mask with values lower (0, 147, 160) and upper (16, 255, 255) is applied, which detects bonus point regions (Figure 2). With the mask applied, the function **crop_to_shape** is used to crop the square further so that the content of the possible digit occupies the entire surface. Finally, the function **find_digit** is called to identify whether the square contains a digit and whether it is 1 or 2.



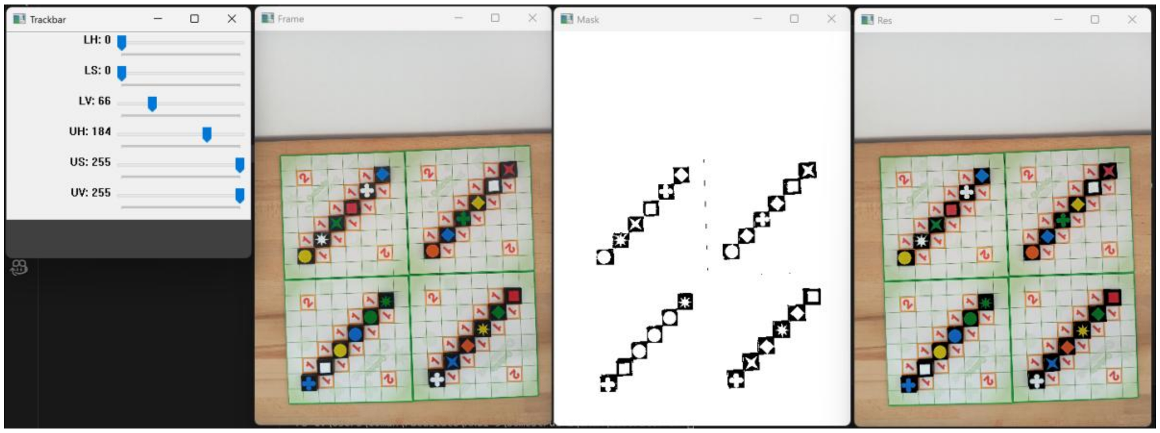
LH: 0
LS: 147
LV: 160
UH: 16
US: 255
UV: 255

Figure 2

The **find_digit** function uses template matching with images from the templates folder, for digits 1 and 2. Each template is placed over the square and rotated by 90 degrees until the best match is found. If no match is found, the function returns 0, and we then check whether the square contains a piece.

3. Detecting Pieces on the Board

First, an HSV mask with values lower (0, 0, 66) and upper (184, 255, 255) is applied to detect only pieces on the board (Figure 3). If a piece is found in the square, the coordinates are added to the shapes list only if the piece is detected for the first time. Using the masked square, **crop_to_shape** is called to remove background and keep only the piece shape. Then the function **find_shape** is used to identify the shape.



LH: 0
LS: 0
LV: 66
UH: 184
US: 255
UV: 255

Figure 3

3.1 Detecting Piece Shapes

Just like **find_digit**, the **find_shape** function uses template matching to detect the shape of the piece. Each shape template is applied to the square, and the number corresponding to the matching shape is returned. If no shape matches, the function returns 0. If a shape **is** detected, **find_color** is called to determine the color.

3.2 Detecting Piece Colors

The function **find_color** uses HSV masks for the following colors:

- **Red:** lower (177, 146, 0), upper (255, 255, 255)

- **Orange:** lower (0, 144, 156), upper (24, 255, 255)
- **Yellow:** lower (24, 132, 58), upper (59, 255, 255)
- **Green:** lower (64, 206, 31), upper (101, 255, 193)
- **Blue:** lower (101, 160, 56), upper (113, 255, 255)

Each mask is applied to the detected piece and the corresponding color is returned. If no mask matches, the piece is assumed to be white. After identifying the shape and color of the piece in the square, the coordinates, shape, and color are saved in the list pieces.

If both **find_digit** and **find_shape** return 0, the square is empty and nothing is done.

4. Calculating the Score

Three cases are considered when calculating the score:

- All pieces added in the round are on the same row → checked using **have_the_same_line**
- All pieces added in the round are on the same column → checked using **have_the_same_column**
- Only one piece was added in the round

In the first case, the leftmost and rightmost ends of the line are found using **go_left** and **go_right**, and the score increases by the number of pieces on that line.

In the second case, the top and bottom ends of the column are found using **go_up** and **go_down**, and the score increases by the number of pieces in that column.

In the third case, we check if the added piece has neighbors. If yes, the ends of the line/column it forms with the first neighbor are found, and the number of pieces in that line/column is added to the score.

For the pieces added in the current round, if they are placed on bonus squares, the corresponding bonus points are also added to the score.

5. Writing the Result to a .txt File

Finally, the function **write_solution** is called to create a text file with the same name as the current image. Information about the added pieces and the score is written into this file.