Qwirkle Score Calculator

Olaeriu Vlad Mihai

May 2025

1 Introduction

The goal of this project is to develop an automatic system to calculate the score for the game of Qwirkle Connect.

Qwirkle Connect is an extension that builds upon the classic Qwirkle rules. Qwirkle Connect employs a dedicated board to guide gameplay. The board is typically subdivided into squares (often configured in four interchangeable quadrants), with some special squares offering bonus points or anchors to place initial tiles.

As in the original Qwirkle, players use 108 tiles combining six shapes and six colors, forming lines that match either shape or color, without repeating tiles.

2 Approach

Before diving into the details, below is the list taken to be able to interpret a game of Qwirkle Connect:

2.1 Starting board

1. Extract the board using contour
2. Extract the structure of the board (bonus points, where the first 24 tiles where placed)

2.2 Following boards

1. Extract the board using contour
2. Resize the board to 1600 x 1600
3. Extract the tile patches after separating the board using a grid with cells of size 100 x 100 pixels
4. Compare the current board with the previous board to identify where new tiles where added

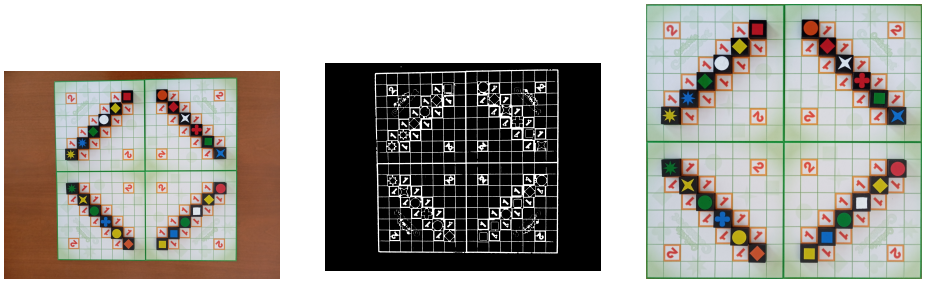
5. Identify the basic shapes using template matching (square, cirlce, diamond)
6. Identify more complex shapes using SIFT features (4-star, 8-star, clove)
7. Compute the score and repeat

3 Board

3.1 Extraction

To be able to extract the board, the image was transformed from RGB to Grayscale, blurred using *median blurring*, then it was transformed to a binary image using *adaptive thresholding*. Afterwards, to close small holes and gaps in the background, respectively to smooth boundaries: dilation followed by erosion, using morphological operations from OpenCV (Figure 1b).

It is used to close small holes or gaps in the foreground (white regions) and to smooth boundaries of the object.



(a) Initial (b) Preprocessed (c) Extracted

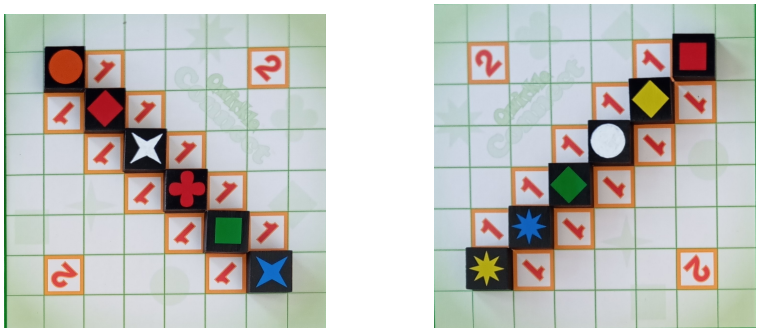
Figure 1: Steps taken to extract the Qwirkle board

After preprocessing the image, the next step was to find the largest contour that could fit into a rectangle with four corners. To do this, all contours were iterated and the one with the largest area was kept and considered to be the board. Then, using a perspective transform operation (*warp perspective*), the viewpoint of the board was changed to correctly align it.

3.2 Scoring board

Next, the configuration of the starting board was extracted in order to access the bonus points tiles at any time. The method chosen was based on the observation that a quadrant can only have two configurations:

Thus, the tiles (1, 1), (1, 9), (9, 1), (9, 9) where checked against a template tiles containing a two (oriented along the first diagonal), and if a match was found, it meant that the quadrant was of type Figure 3b. Otherwise, it was of type Figure 3a.



(a) First diagonal (b) Second diagonal

Figure 2: Quadrant types

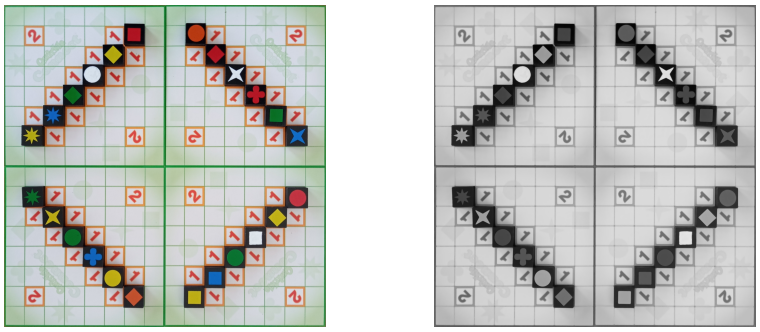
4 Tiles

4.1 Patches

After the perspective transform, the board was resized to 1600 x 1600 pixels. Each tile had a size of 100 x 100 pixels, and they were extracted based on the grid of 16 x 16 tiles.

4.2 Positions

There are 6 possible shapes for a tile: square, circle, diamond, clove, 4-star and 8-star. A template was created for each shape and used to identify where the tiles are placed in the image. (the templates where grayscaled, blurred).



(a) First diagonal (b) Second diagonal

Figure 3: Quadrant types

The template matching search was run for each of the shapes, using the *matchTemplate* function from OpenCV. The function used was the **Normalized Cross-Correlation Coefficient (NCC)**.

When a match was found, the center of the template was computed and its position was identified in the grid, then marked. All the marked positions represent positions where tiles are placed, and the others are empty spaces.

4.3 Identification

To identify the tiles types, two processes were run: the identification of shape and the identification of color.

To find the shape of the tiles, they were separated into two categories:

1. simple: square, circle, diamond
2. complicated: clove, 4-star, 8-diamond

and they were matched against multiple tiles templates.

(a) circle (b) square (c) diamond

(d) 8-star (e) clove (f) 4-star

Figure 4: Tiles templates

The complicated shapes were identified using a similarity score based on SIFT features: key points and descriptions were extracted from the images, and Brute Force matching with KNN (K-Nearest Neighbors) was used to find the closest matches between the descriptions. Lowe's ratio test was then applied to remove weak matches, ensuring only the best matches were used to calculate the final similarity score.

The simple shapes were identified using the template matching that was previously mentioned. From the test run with the SIFT similarity, the SIFT features were unable to make a clear distinction between squares, circles and diamonds. Luckily, the template matching using NCC worked well for these shapes.

To identify the color, the dominant color of a given tile image was determined using the HSV color space and predefined color ranges.This was done by creating binary masks for each color range, counting the number of matching pixels, and selecting the color with the highest pixel count as the dominant one.

5 Scorer

Finally, the scorer was mainly based on the initial scoring board, which was extracted from the starting board configuration, respectively on the difference between the two boards.

After extracting the tiles and identifying their shape and color, the next step was to compare the current board with the previous board. The assumption that each move made was correct.

The column and row of each new tile were analyzed to see how many points it generated. When a column or row shows that it generated points, it was marked and not considered for the remaining new tiles.