Project Documentation: Qwirkle Game Monitor

Gheorghe Bogdan-Alexandru (Group 343)

1 Project Overview

The goal of this project is to build a software system that can automatically calculate the score for a Qwirkle game.

The main challenge was to make the computer recognize the game pieces (shapes and colors) even if the camera angle was not perfect or if the pieces were slightly not in place.

2 Step 1: Preparing the Image

Before I can find any pieces, I need to isolate the game board.

2.1 Finding the Board

The photos taken with a phone show the table with some background noise. To fix this, I converted the image to Grayscale and applied Gaussian blur. This removes small details such as dust. Then, I use *Adaptive Thresholding* to highlight the grid lines.

After finding the edges, the program looks for the biggest square shape in the picture, and this is how I isolate the game board.

2.2 Straightening the Perspective

Since the photo was taken from an angle, the board looks like a trapezoid. I use a function called *warpPerspective* to stretch the image so that it looks like a perfect top-down view (a 16x16 grid).

The Padding Solution: I faced a problem where the pieces that were placed on the edge of the board were cut off. To fix this, I created two versions of the board image:

1. **Standard View:** A 1600x1600 pixel image used to check the grid alignment.
2. **Padded View:** A larger image with an extra 25-pixel border. This allows the computer to see pieces that stick out of the grid boundaries.

3 Step 2: Detecting Moves

The system needs to know *when* and *where* a player placed a piece.

I do this by comparing the current photo with the photo from the previous turn. The two images are subtracted. If a pixel was black and is now colorful, the result is a white spot. If there are enough white spots in a grid square, I know a piece was placed there.

4 Step 3: Recognizing the Pieces

Once I find a new piece, I need to know two things: its Shape and its Color.

4.1 Shape Recognition (Template Matching)

This was the tricky part. I manually created the 6 shapes (Circle, Square, Diamond, Clover, 4-Point Star and 8-Point Star).

The problem is that the pieces are not perfectly centered, so a standard computer search would fail. Every square on the table was taken with a small padding in every direction, so the template matching will have the highest value in the point where the piece is centered. Then I keep the best match score between each comparison.

4.2 Color Recognition

I use the HSV color mode because it separates "Color" (Hue) from "Brightness" (Value). This helps when shadows fall on the board. I defined ranges for Red (both high and low values), Orange, Yellow, Green, Blue, and White.

5 Step 4: Calculating the Score

After identifying the new pieces, the program calculates the points based on Qwirkle rules.

How it works:

- The algorithm scans the board horizontally and vertically starting from the new pieces.
- It counts how many pieces are connected in a line.
- If a line is longer than 1 piece, points are added.
- If a line has 6 pieces, the "Qwirkle Bonus" (x2) is applied.
- The code also checks for board bonuses (like double point spots) if the game configuration has them.

To make the code simpler, I used a trick for vertical scanning: I transpose the matrix (swap rows with columns) and reuse the horizontal scanning function.