Double Dice Dominoes CV Analysis

Tools:

- Python 3.11
- OpenCV
- Scikit-Image

Main Stages

1. Obtaining the Full Game Board:

The initial step involves obtaining the full game board, including the sides with the scoring tab. We do this step because it is very hard to get the game board without the scoring tab directly, as it contains a lot of information and edges.

We use grayscale, gaussian and dilation to remove noise, then we find edges using the Canny detector. The largest contour in the image is detected to isolate the full game board. We use the rectangle that contains this contour, as we do not need the perfect contour (which is also susceptible to mistakes)

```
kernel = np.ones((5, 5), np.uint8)
gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
blurred = cv.GaussianBlur(gray, (5, 5), 0)
edges = cv.Canny(blurred, 50, 150)
dilated = cv.dilate(edges, kernel, iterations=3)
eroded = cv.erode(dilated, kernel, iterations=1)

contours, _ = cv.findContours(eroded.copy(), cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key=cv.contourArea,
reverse=True)[:1]
x, y, w, h = cv.boundingRect(contours[0])
cropped = image[y:y + h, x:x + w]
```

2. Crop Scoring Tab:

The scoring tab is cropped from the full board using predefined margins. The resulting image represents the game board without the scoring tab, but with some leftovers on the sides of the space between the scoring tab and the game board. I identified these values (as percentages) that the side board occupies

```
margin_of_error = -0.015
left = 0.159 + margin_of_error
top = 0.1548 + margin_of_error
right = 1 - (0.1527 + margin_of_error)
bottom = 1 - (0.1548 + margin_of_error)
```

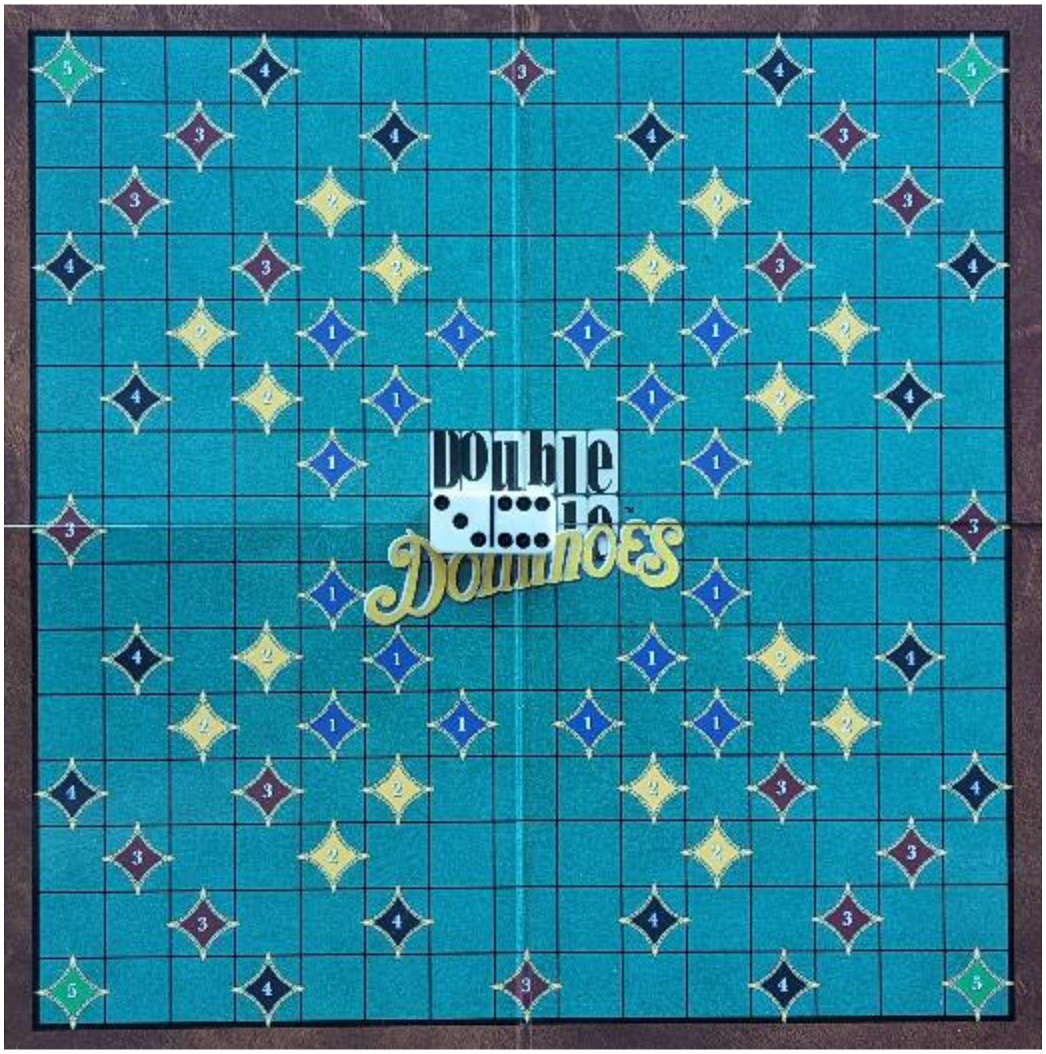


Figure 1, board after cropping

This is a safe method as all normal DDD boards have the same ratios between the scoring tab and game board.

3. Obtaining the Actual Game Board:

Similar to step one, we preprocess the image by removing noise. We use a 2 binary threshold, of 30 and 220, to highlight the black outline of the game board AND the possible white, played pieces that may eventually cover this black outline. We find contours on this result and identify the board as the largest contour. We pick the actual four corner points by calculating which points create the largest area. Note that no edge detection is used, as it detected a lot of noise or unwanted edges on the game board.

```
for i in range(len(contours)):
    if len(contours[i]) > 3:
        possible_top_left = None
        possible_bottom_right = None
        for point in contours[i].squeeze():
            if possible_top_left is None or point[0] + point[1]
< possible_top_left[0] + possible_top_left[1]:
                possible_top_left = point

            if possible_bottom_right is None or point[0] +
point[1] > possible_bottom_right[0] + possible_bottom_right[1]:
                possible_bottom_right = point

        diff = np.diff(contours[i].squeeze(), axis=1)
        possible_top_right =
contours[i].squeeze()[np.argmin(diff)]
        possible_bottom_left =
contours[i].squeeze()[np.argmax(diff)]
        if cv.contourArea(np.array([[possible_top_left],
[possible_top_right], [possible_bottom_right],
                                    [possible_bottom_left]])) >
max_area:
            max_area = cv.contourArea(np.array(
                [[possible_top_left], [possible_top_right],
[possible_bottom_right], [possible_bottom_left]]))
            top_left = possible_top_left
            bottom_right = possible_bottom_right
            top_right = possible_top_right
            bottom_left = possible_bottom_left
```

Finally, we transpose the four corners to a perfectly rectangular shape of size 1200x1200 (nicely divisible by 15x15) using perspective transform.

4. Splitting the Board into Patches:

game board is divided into a grid of 15x15 patches, representing the standard configuration of the dominoes game. Vertical and horizontal lines are defined to create the grid. We use them to parse the image, patch by patch.

5. Piece Placement Detection:

Changes in patches between consecutive images are identified using the Structural Similarity Index (SSIM). The patch with the most significant change is considered the location of the placed piece. SSIM is applied on images after we use a binary threshold of 150, as we are only interested in placed pieces, which are white.

Additional, logic based checks may be implemented, such as keeping track of what spots are already occupied, where it would be impossible to place a piece as there are no pieces nearby etc. I also tested other similary measures, but SSIM was the best for this scenario (~97%). RMSE -> terrible results, PSNR -> decent results (~50%), FSIM -> great, but really slow (~90%), SRE -> great and fast (~93%).

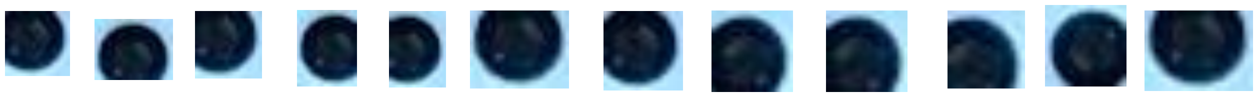
```
gray1 = cv.cvtColor(image1, cv.COLOR_BGR2GRAY)
gray2 = cv.cvtColor(image2, cv.COLOR_BGR2GRAY)
scores = []
for h_line in horizontal_lines:
    for v_line in vertical_lines:
        patch1 = gray1[h_line:h_line + square_size,
v_line:v_line + square_size]
        patch2 = gray2[h_line:h_line + square_size,
v_line:v_line + square_size]

        patch1 = cv.threshold(patch1, 150, 255,
cv.THRESH_BINARY)[1]
        patch2 = cv.threshold(patch2, 150, 255,
cv.THRESH_BINARY)[1]
        score = ssim(patch1, patch2)
        scores.append(score)
```

6. Piece Identification:

After determining the placement, the type of the placed piece is identified using template matching. Binary thresholding and Gaussian blur are applied to preprocess both the template and patches in order to remove noise, as the pieces are black and white.

We use around 50 templates of single dots in various positions to attempt to count the number of dots found in a patch. This number represents the piece we found. The threshold used is 0.8 and the results are very sensible to changes to it. If we add more templates, we also need to increase the threshold, while the reverse is also true. We use non maximum supression to remove overlapping identified patterns to get our final result.

Some images of dot templates:



7. Scoring Calculation:

The score for each move is calculated based on the game rules. The score is influenced by the type of pieces placed, their positions on the board, and specific game rules.

Parameters and Configurations

Parameters such as fixed_width, fixed_height, and square_count are defined to ensure consistency in image processing. templates_path specifies the directory containing templates for piece identification. ladder_tiles and score_board store information about piece types and scoring rules.

Workflow

The project processes multiple game instances in parallel, each represented by a separate process. For each game instance, images and corresponding move files are loaded. The game board is initialized using an empty board image. Image processing steps, including contour detection, patch extraction, and template matching, are applied to each image in sequence. Scores are calculated for each move based on the identified pieces and their positions on the board.