Piece detection and score calculation in Scrabble boardgame

In this paper, I will describe an approach to detect and calculate the score of a configuration modification in a Scrabble game from an image set provided.

Step 1: Board detection

In this step, we want to isolate the relevant information of the input image and that is, in our case, the central square of the board.

Fig 1. The desired central square described above

We will sharpen the image, so we can highlight the edges for a better detection later on. I'm doing that by creating a gray copy of the initial image. From that copy we are making two more copies: one which is the gray copy after applying a median blur with a size of 3 and another one which is the gray copy after applying a gaussian blur of deviation 5. We will add those two copies in a final sharpened image, the median blurred one having a weight of -0.9 while the gaussian blurred one having a weight of 1.1.

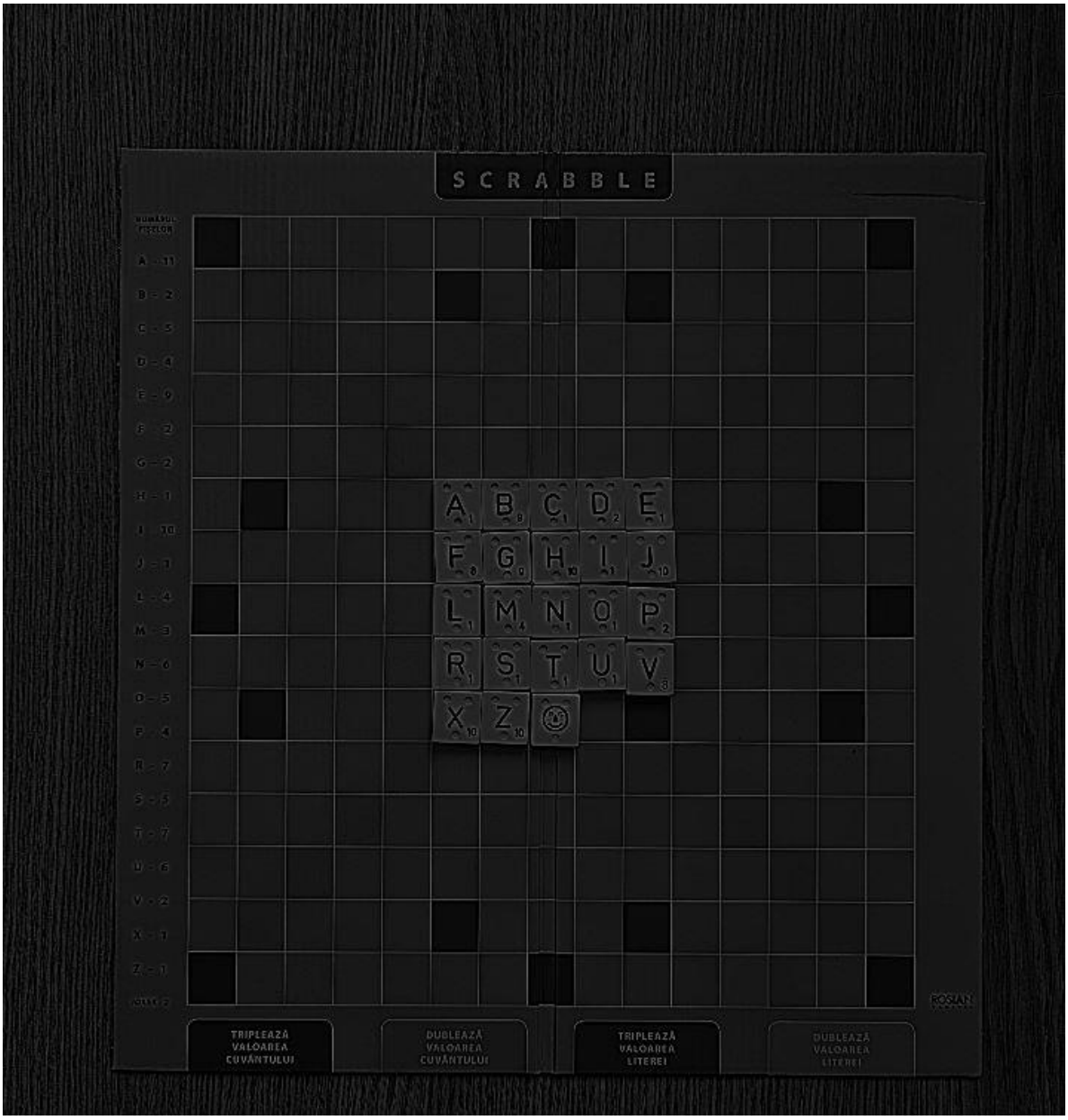

Fig 2. The sharpened image, we can observe how the white interior lines have sharpened

After sharpening the image, we will threshold the new image in the following way: pixels over the value of 47 to white, while the others pixels will be converted to black.

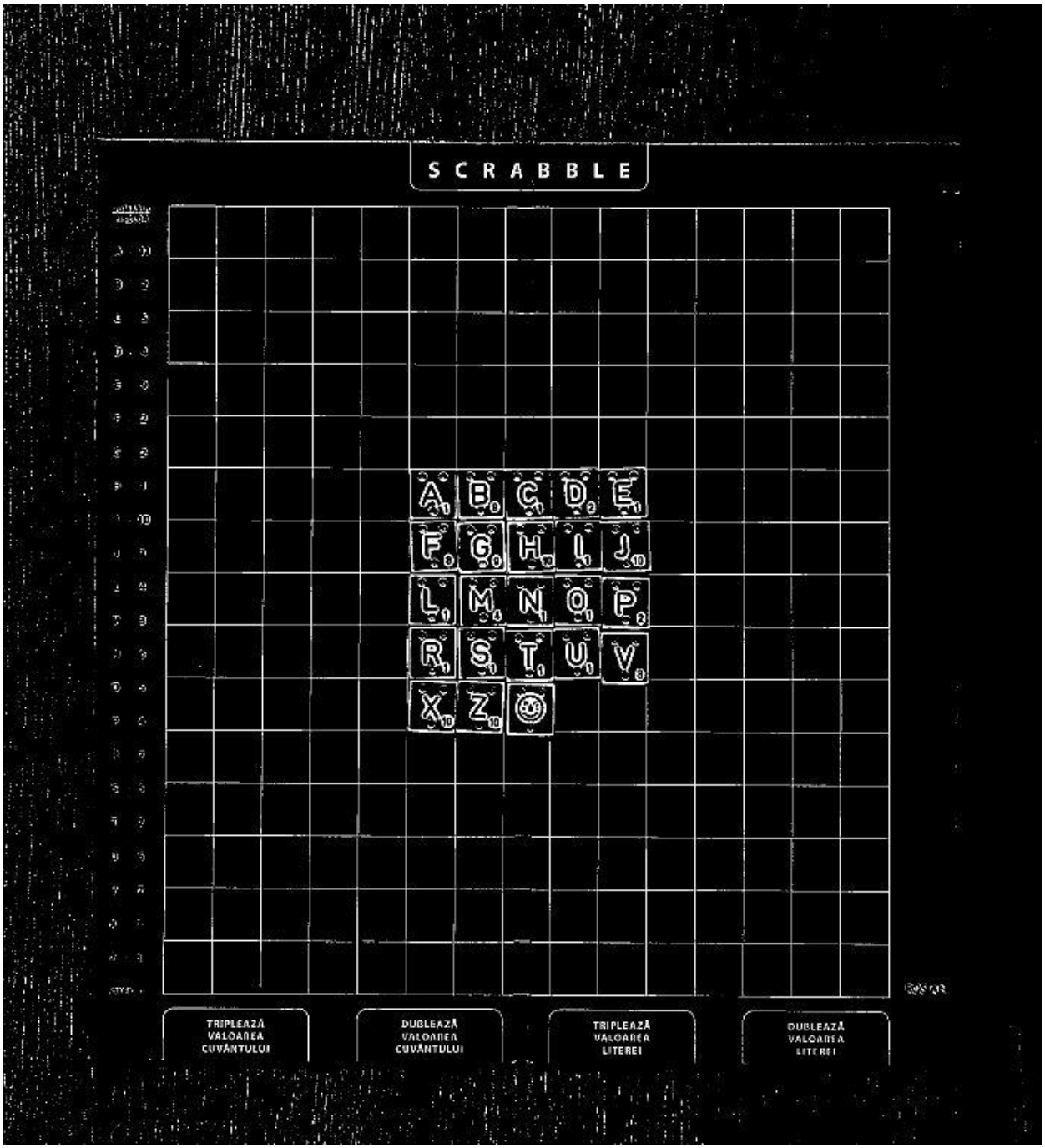

Fig 3. The image thresholded

We can observe that after the thresholding, the center square desired to extract is much clearer, but we also remained with a lot of noise outside of it. The approach we will take now will seem a little odd. Trying to remove the noise through an erosion followed by a dilatation will likely make irreparable damage to our center square edges. What we will actually do is embrace the noise, and make it even more powerful. We can observe that our noise is mostly vertical lines, so a horizontal dilatation will be a step in creating a more uniform noise. We will apply two dilatation filters on the above picture, one of size (1, 10) to create the masses described above, and another one of size (5, 10) to better connect the vertical noise.

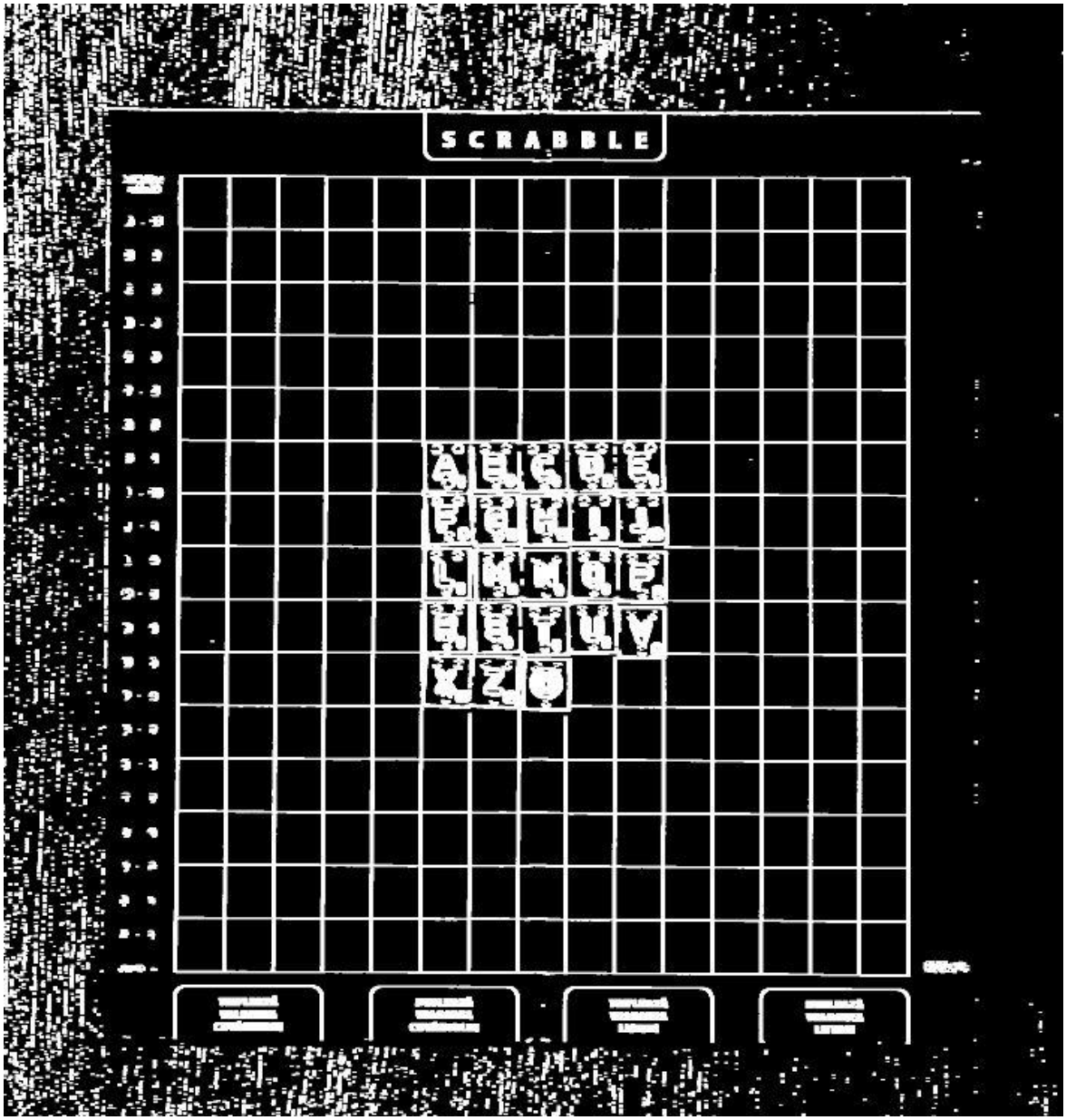

Fig 4. The pictured dilated as described above

After all the filtering done above, we will finally try to draw the edges using the Canny function from OpenCV.

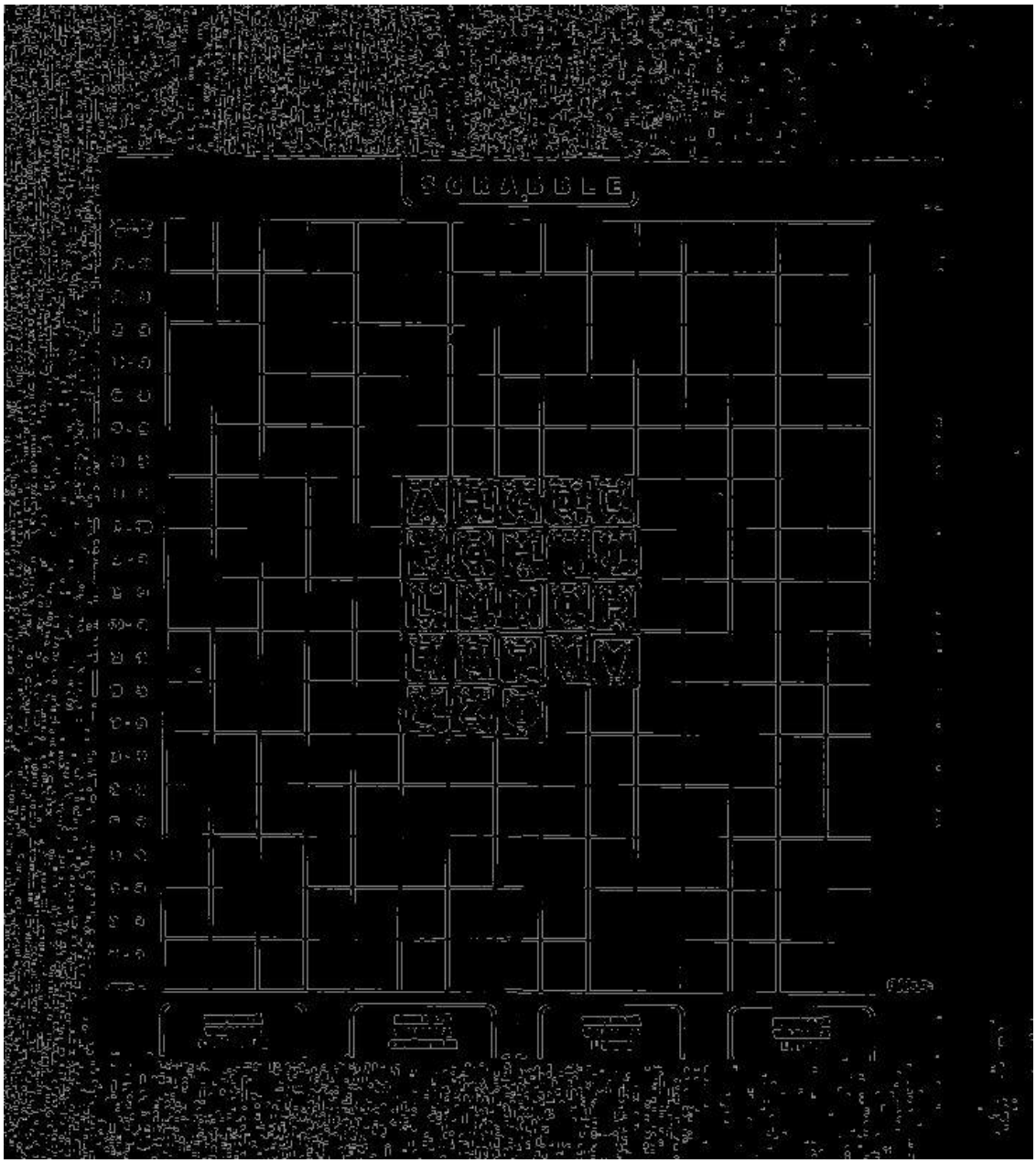

Fig 5. The edges picture generated by cv2.Canny

Now, having the edges extracted, we will use the findContours function from OpenCV to determine all the polygons from the edges image, from which we will assume that the one with the biggest area is the one we are looking for. As you can observe, the noise was transformed in

a lot of smaller edges that will not interfere with out maximum contour. Also, the top row of the board will get confounded with the noise, so it won't interfere with our maximum contour. After finding the biggest area polygon, we will crop from the image our square and resize it to (2100, 2100) for a better handling latter on.

Fig 6. The detected central square (with the 4 red circles)

Step 2: Information extraction

After knowing that we are in a (2100, 2100) photo of our central board, we will now determine for each cell weather there is or there isn't a letter placed and, if there is one, determine which one it is.

The board contains 225 images (the boardgame has 15 rows and 15 columns) we can determine the location of each cell as follows: the cell from row **i** and column **j** has the top left corner at **((i - 1) * (2100 / 15), (j - 1) * (2100 / 15))** and top right corner at **(i * (2100 / 15) , j * 2100 / 15))**.

Now, after we know the position of each cell, we must find what is located there and if there is a anything located at all. Before this, we will have to know for each letter how it looks, so we will make images for each letter before continuing.

Fig 7. The letters we can find in our cells, each of these pictures with the size (60, 70)

What will now do is to try, for each cell of the board, to match an image from the ones above to the current cell. For that we will use the cv2.matchTemplate which returns an accuracy between 0 to 100%. If we have no matching with the accuracy better than 75%, then we will assume that there is no letter placed, so the cell will be empty.

In the Figure 8 we can observe the code used to determine for one image it's corresponding matrix. We have the function call "extrage_careu(image)" that receives an image of the board as parameter and returns the cropped (2100, 2100) image of the central square as described in Step 1, followed by the "get_matrix(image, options)" function call that receives the central square of the board and the path to the option pictures (images from Figure 7) and returns a matrix of the information of each cell: character "_" if the cell is empty or the letter from that cell otherwise (for the joker token it outputs "?")

```
img = cv.imread("./CAVA-2022-TEMA1/imagini_auxiliare/litere_1.jpg")
show_image("imaginelala", img)
rez = extrage_careu(img)
#show_image("imagine", rez)
matrix = get_matrix(rez, "./CAVA-2022-TEMA1/optiuni/")

for i in range(len(matrix[0])):
    for j in range(len(matrix[i])):
        if matrix[i][j] == "K":
            print("?", end = "")
        else:
            print(matrix[i][j], end = " ")
    print("\n", end = "")
print("\n", end = "\n")
```

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ A B C D E _ _ _ _ _
_ _ _ _ _ F G H I J _ _ _ _ _
_ _ _ _ _ L M N O P _ _ _ _ _
_ _ _ _ _ R S T U V _ _ _ _ _
_ _ _ _ _ X Z ?_ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Fig 8. The code used and the matrix detected using our algorithm.

Step 3: Score calculus

To calculate the score, we first hardcode the positions of the special characters and the values of each letter (because they are always the same).

```
In [28]: score_per_letter = {
             "A" : 1,
             "B" : 9,
             "C" : 1,
             "D" : 2,
             "E" : 1,
             "F" : 8,
             "G" : 9,
             "H" : 10,
             "I" : 1,
             "J" : 10,
             "K" : 0,
             "L" : 1,
             "M" : 4,
             "N" : 1,
             "O" : 1,
             "P" : 2,
             "R" : 1,
             "S" : 1,
             "T" : 1,
             "U" : 1,
             "V" : 8,
             "X" : 10,
             "Z" : 10,
         }

         red_cells = [(0, 0), (0, 7), (0, 14), (7, 0), (7, 14), (14, 0), (14, 7), (14, 14)]

         blue_cells = [(1, 5), (1, 9), (5, 1), (5, 5), (5, 9), (5, 13), (9, 1), (9, 5), (9, 9), (9, 13), (13, 5), (13, 9)]

         purple_cells = [(1, 1), (2, 2), (3, 3), (4, 4), (1, 13), (2, 12), (3, 11), (4, 10),
                         (13, 1), (12, 2), (11, 3), (10, 4), (10, 10), (11, 11), (12, 12), (13, 13), (7, 7)]

         cyan_cells = [(0, 3), (0, 11), (2, 6), (2, 8), (3, 0), (3, 7), (3, 14), (6, 2), (6, 6), (6, 8), (6, 12), (7, 3), (7, 11)
                       , (8, 2), (8, 6), (8, 8), (8, 12), (11, 0), (11, 7), (11, 14), (12, 6), (12, 8), (14, 3) , (14, 11)]
```

Fig 9. The scores for each letter (K = joker) and the special positions

To determine the newly added word, we will check the positions where the value changed between the previous state matrix and our new matrix calculated after Step 2, resulted in a list the positions of the newly added letters. To know in each of those positions what letters is there, we will just check the respective position in the matrix. For each position added in the last turn, we will add them in a dictionary **added.** Now, we will look in the newly added letters and check the top-left most and the bottom-right most. Now the problem splits in 2 cases depending on those 2 positions (whether they are on the same row or column):

- If the 2 corners are on the same column, then the word is placed on the same column. We will check if above or lower of the word if there are already placed letters, in which case we will concatenate them to the word before calculating the score. Now, from each newly added letter, we will check if it has neighbors added in his neighboring columns, if there are, then on that row we have a new word and we will add to the score that word's score, applying the bonuses only for the cells added in this stage.

- If the 2 corners are on the same row, then the word is placed on the same row. We will check if to the left or to the right of the word there are already placed letters, in which case we will concatenate them to the word before calculating the score. Now, from each newly added letter, we will check if it has neighbors added in his neighboring rows, if there are, then on that column we have a new word and we will add to the score that word's score, applying the bonuses only for the cells added in this stage.