Documentatie - Tema 1

Concepte Si Aplicatii In Vederea Artificiala

Proiectul meu contine un folder “templates” si un fisier python “main.py”.

Folder-ul “templates” contine 6 imagini cu ajutorul carora realizam template matching pentru fiecare piesa de domino (va fi detaliat ulterior).

Voi continua prin a prezenta continutul fisierului “main.py”, explicand pasii pe care algoritmul meu ii urmeaza si furnizand bucatile de cod aferente:

1. Import librariile necesare: numpy si opencv.

```
import numpy as np
import cv2 as cv
```

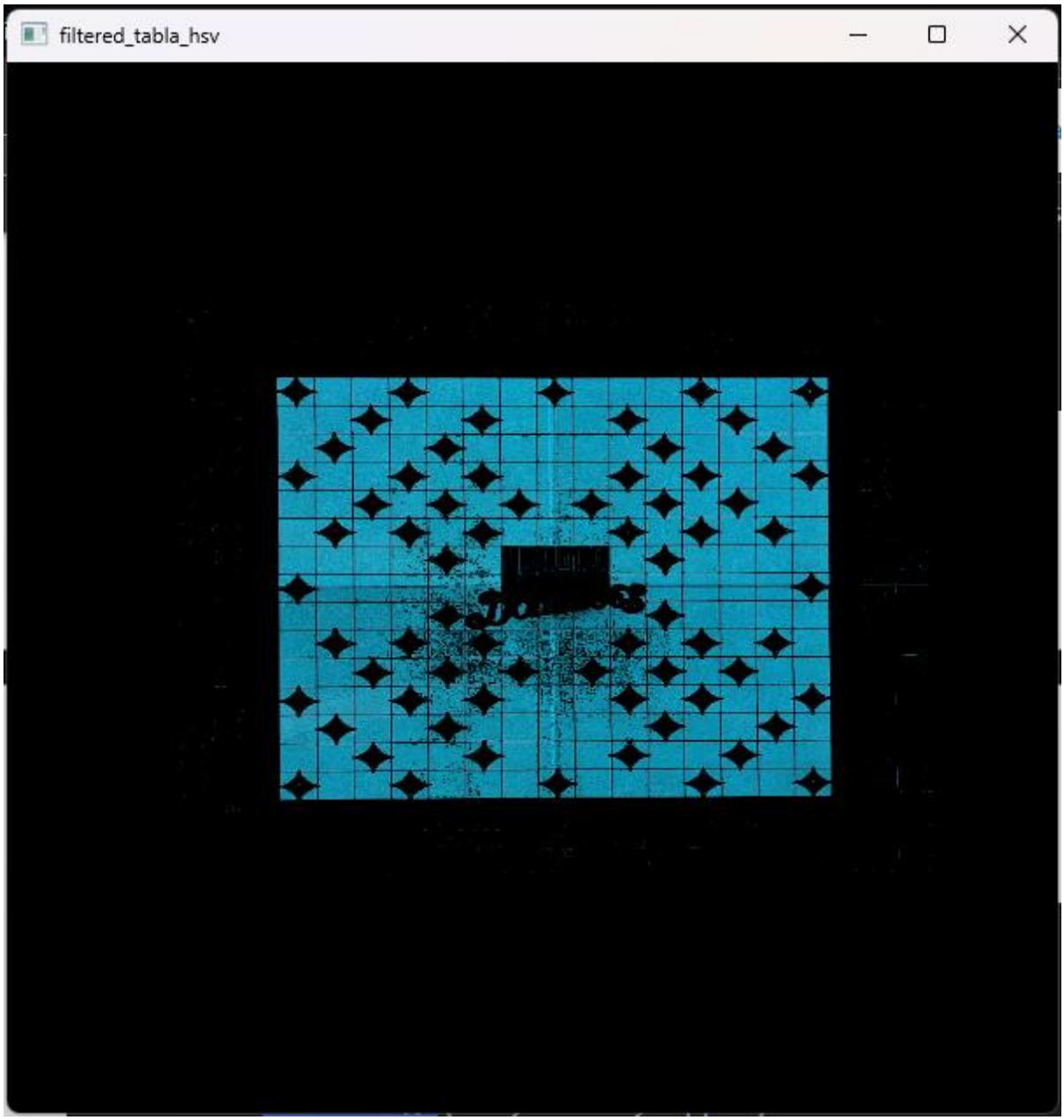
2. Functia show_image cu 2 argumente (titlul imaginii, continutul imaginii) afiseaza imaginea “image” pe ecran.

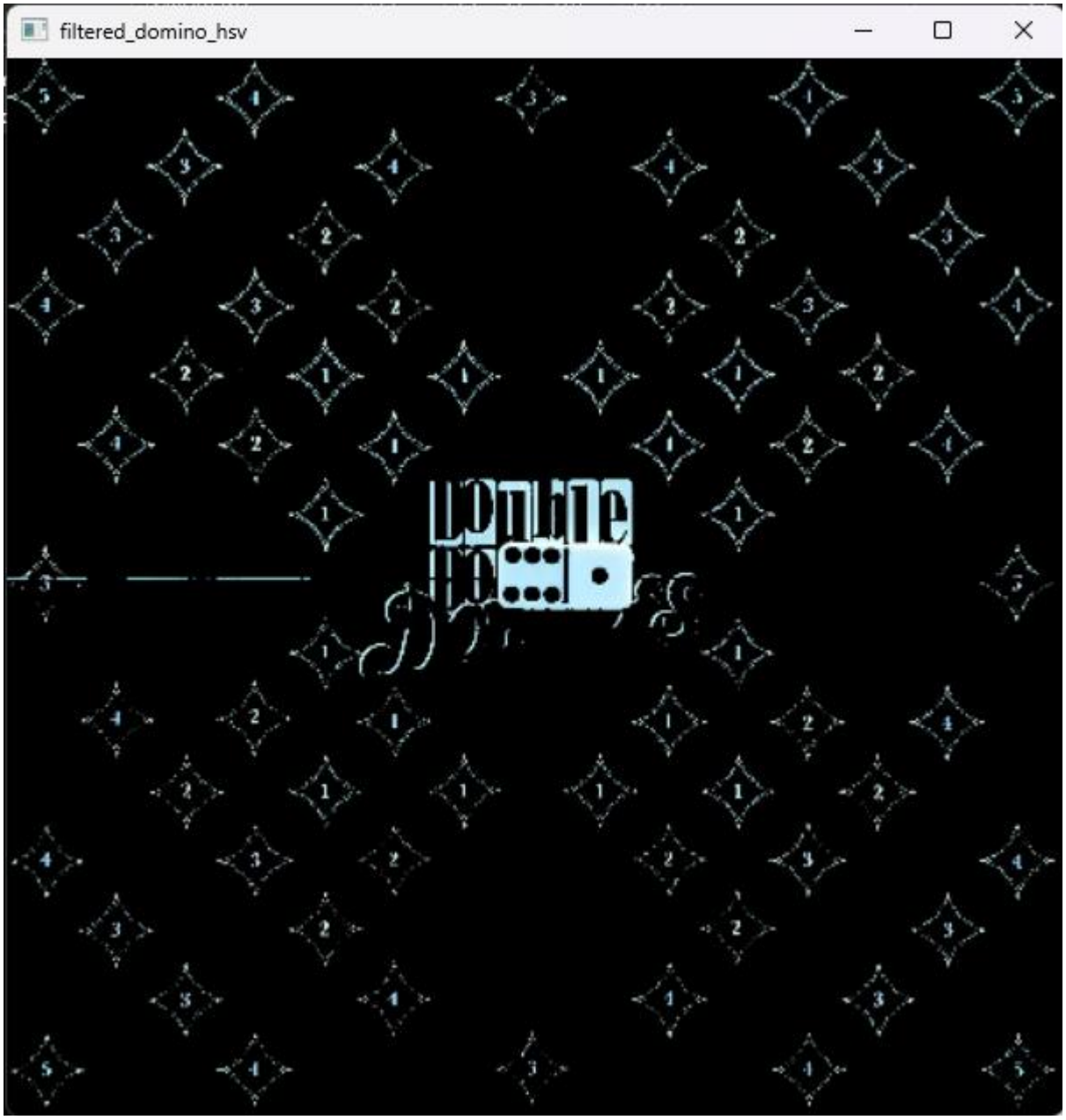
```
def show_image(title,image):
    image = cv.resize(image, (0,0), fx=0.3, fy=0.3)
    image = cv.resize(image, (600, 600))
    cv.imshow(title,image)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

3. Functiile tabla_hsv si domino_hsv primesc ca argument o image BGR si aplicand o masca HSV vor ramane doar pixelii care se incadreaza in intervalul specificat pentru nuanta dorita de albastru (tabla_hsv – pentru prelucrarea tablei de joc si gasirea colturilor; domino_hsv – pentru detectarea valorii piesei de domino).

```
def tabla_hsv(image):

    hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    lower = np.array([95, 120, 0])
    upper = np.array([100, 255, 255])
    mask = cv.inRange(hsv, lower, upper)
    filtered = cv.bitwise_and(image, image, mask= mask)

    return filtered

def domino_hsv(image):
```

```
    hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    lower = np.array([50, 0, 220])
    upper = np.array([155, 105, 255])
    mask = cv.inRange(hsv, lower, upper)
    filtered = cv.bitwise_and(image, image, mask= mask)

    return filtered
```



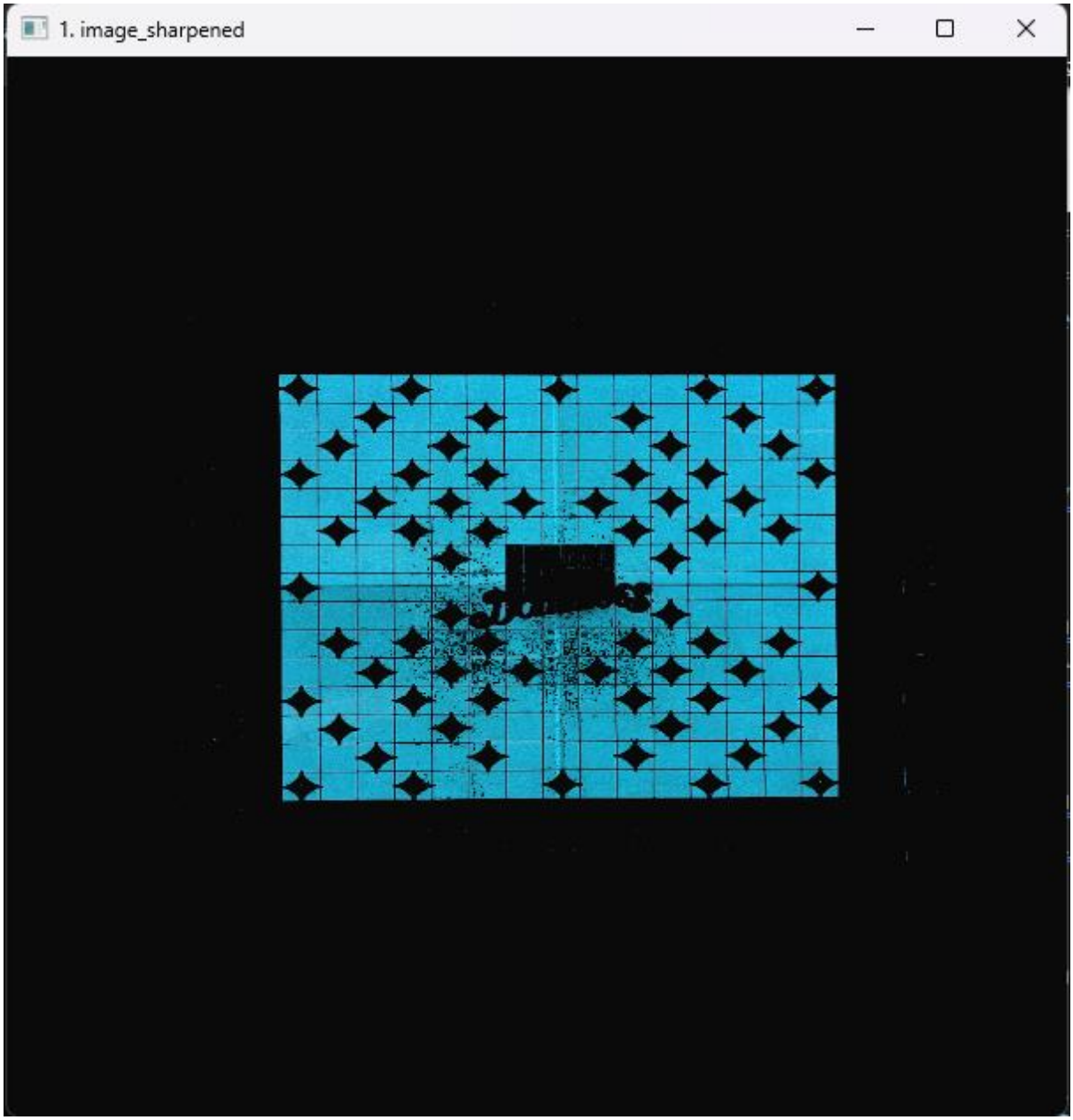


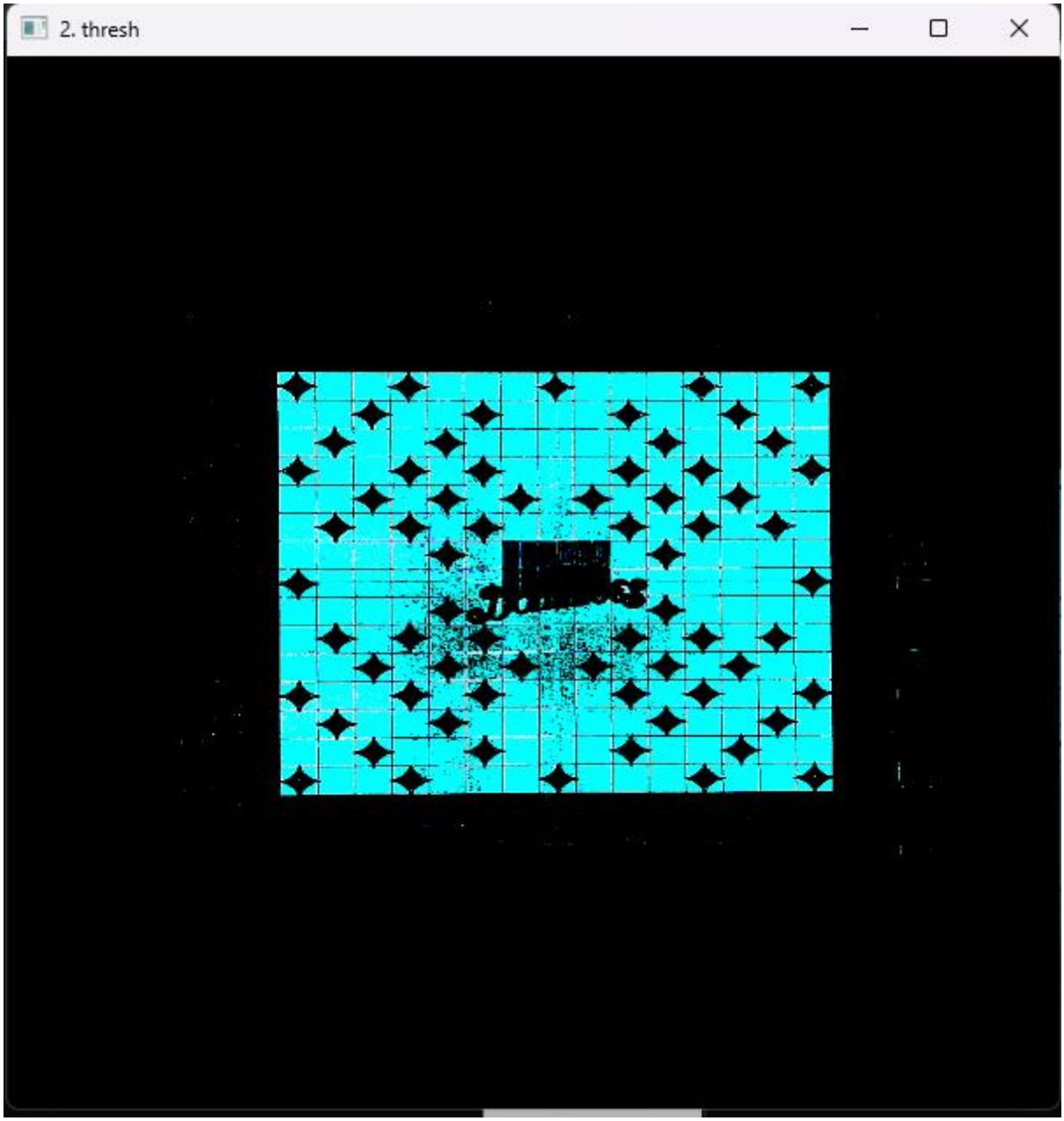
4. Functia extrage careu are scopul de a detecta colturile tablei. Argumentul path preia calea imaginii pe care dorim sa o prelucram, iar corners_coord va fi o lista care contine coordonatele colturilor detectate in imaginea precedenta.

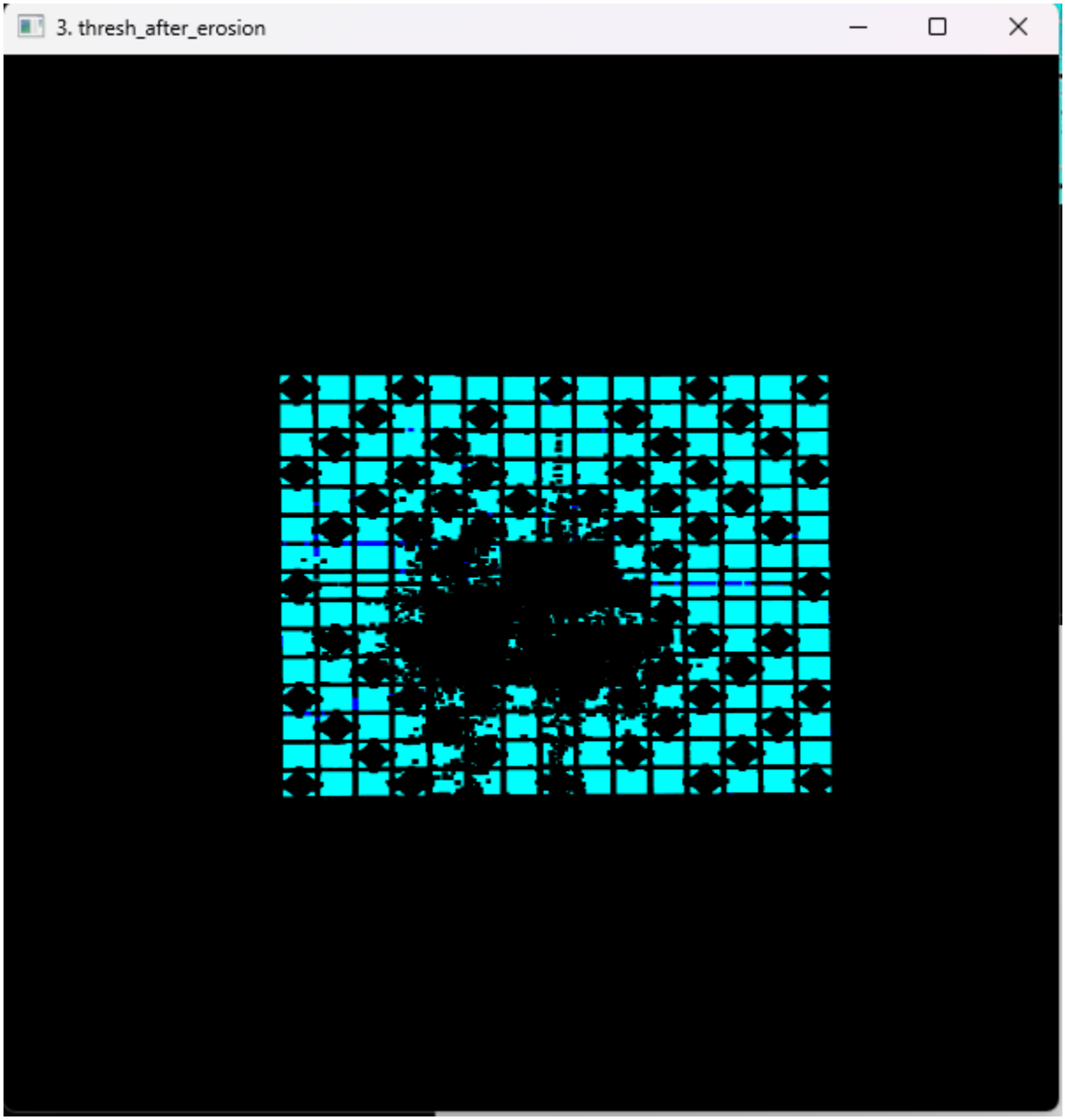
```
def extrage_careu(path, corners_coord):

    image = cv.imread(path)
    image = tabla_hsv(image)
```

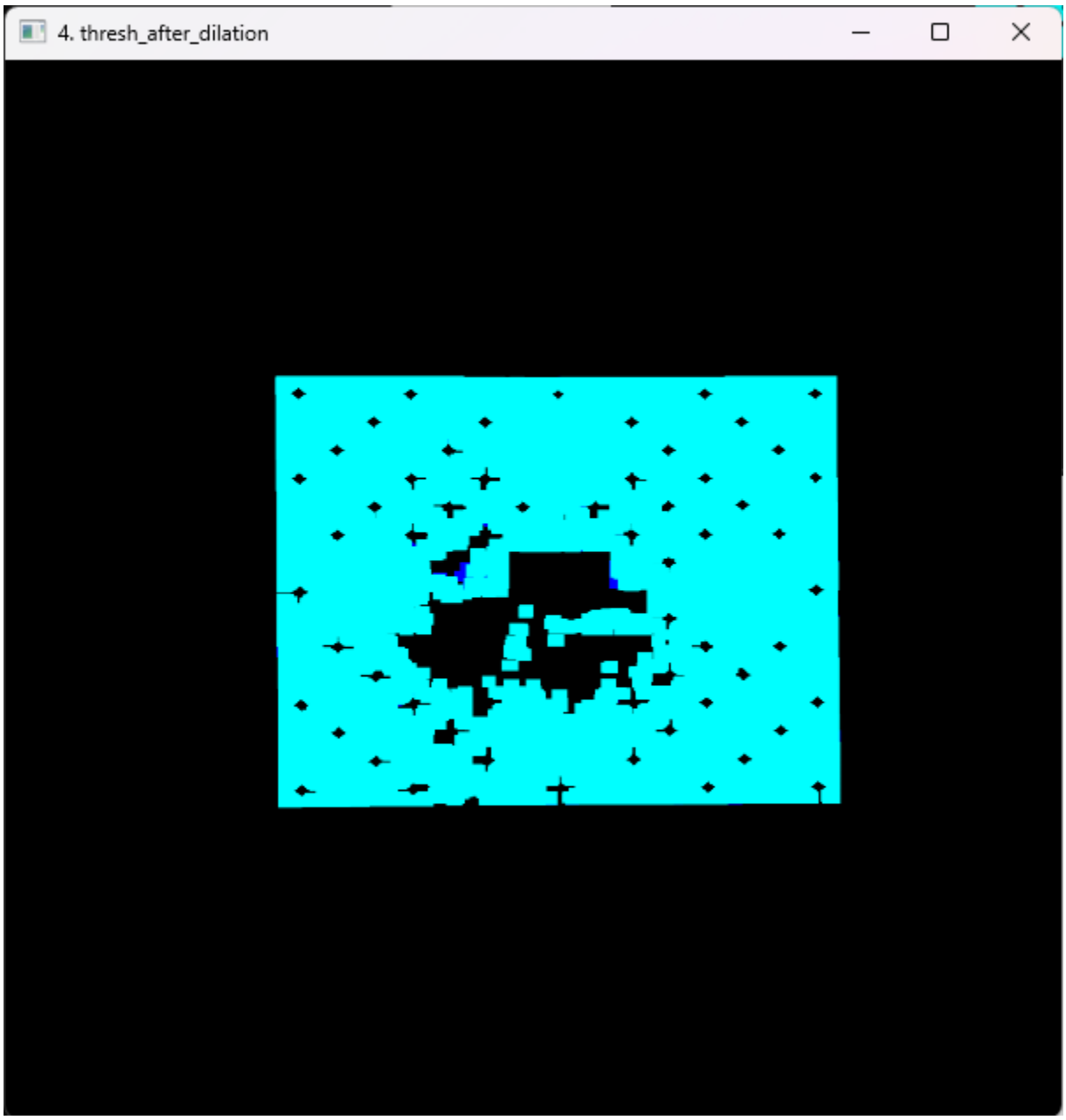
Aplic diverse filtre pentru a putea detecta muchiile potrivite, care ne vor permite ulterior sa gasim colturile bune.

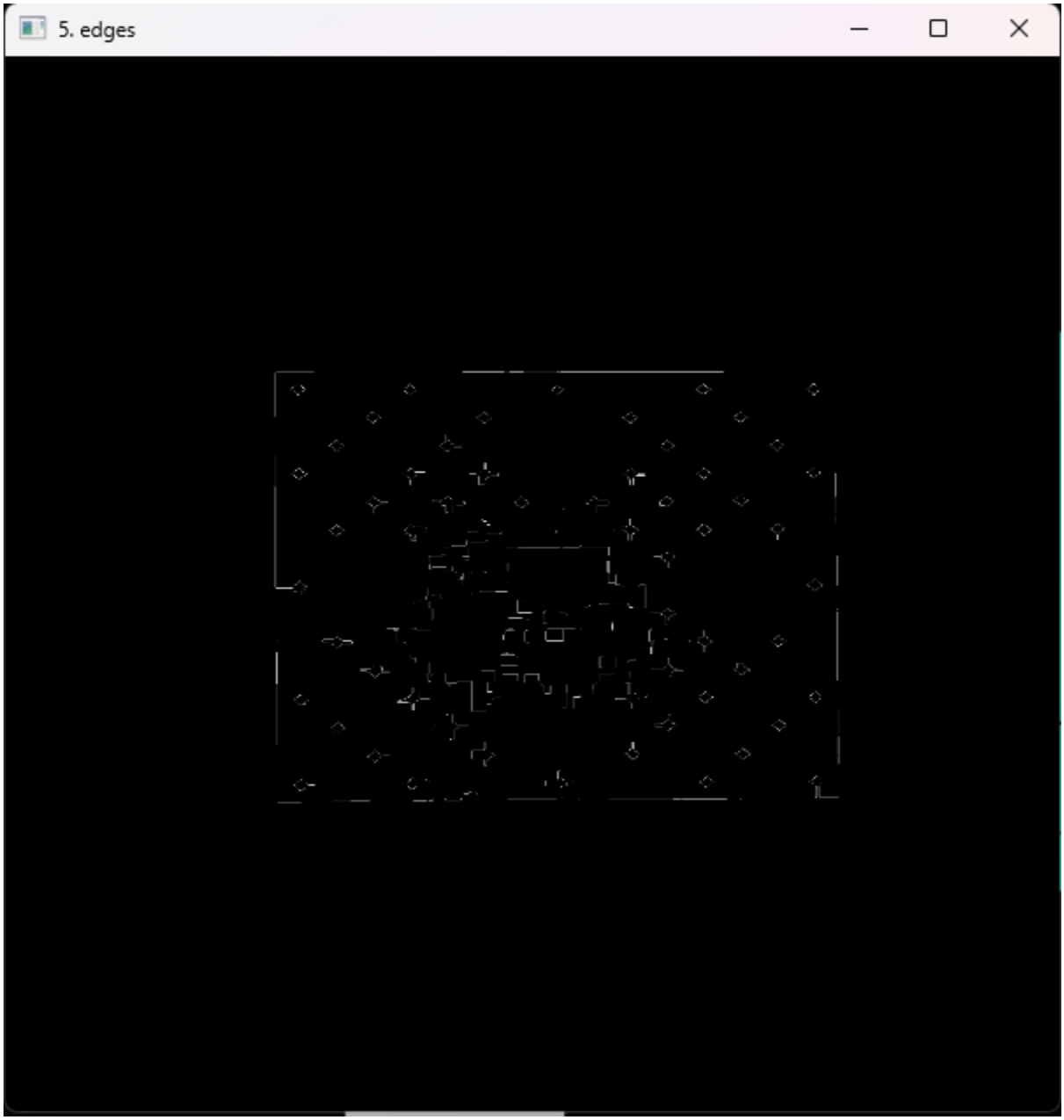
```
 image_m_blur = cv.medianBlur(image, 3)
image_g_blur = cv.GaussianBlur(image_m_blur, (3, 3), 10)
image_sharpened = cv.addWeighted(image_m_blur, 1.7, image_g_blur, -0.6, 10)
show_image('1. image_sharpened',image_sharpened)

_, thresh = cv.threshold(image_sharpened, 60, 255, cv.THRESH_BINARY)
show_image('2. thresh', thresh)

kernel = np.ones((2, 2), np.uint8)
thresh = cv.erode(thresh, kernel, iterations=15)
show_image('3. thresh_after_erosion', thresh)

kernel = np.ones((3, 3), np.uint8)
thresh = cv.dilate(thresh, kernel, iterations=20)
show_image('4. thresh_after_dilation', thresh)

edges =  cv.Canny(thresh, 50, 500)
show_image('5. edges', edges)
contours, _ = cv.findContours(edges,  cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
```


1. image_sharpened


2. thresh


3. thresh_after_erosion


4. thresh_after_dilation



In lista de contururi obtinuta anterior cautam extremitatile si le adaugam corespunzator in variabilele top_left, top_right, bottom_left, bottom_right

```
top_left = None
    bottom_right = None
    top_right = None
    bottom_left = None
    for i in range(len(contours)):
        if(len(contours[i]) > 3):
            for point in contours[i].squeeze():

                if top_left is None or point[0] + point[1] < top_left[0] + top_left[1]:
                    top_left = point
```

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

                if top_right is None or point[1] - point[0] > top_right[1] - top_right[0]:
                    top_right = point

                if bottom_left is None or point[0] - point[1] > bottom_left[0] -
bottom_left[1] :
                    bottom_left = point
```

Mereu prima imagine (chiar primele imagini) vor avea piese in mijlocul tablei, departe de colturi. Filtrele ne detecteaza bine atunci cand nu exista piese in coltul tablei. Pentru a detecta bine colturile si in acest caz, am retinut coordonatele colturilor imaginii precedente in lista corners_coord. Astfel, cand se detecteaza faptul ca o piesa a fost pusa in colt si acesta se deregleaza, vom reveni la coordonatele precedente.

```
    margin = 50
    if corners_coord == []:
        corners_coord = [top_left, top_right, bottom_left, bottom_right]
    else:
        old_top_left, old_top_right, old_bottom_left, old_bottom_right = corners_coord

        if top_left[0] - top_right[0] > margin:
            top_left = old_top_left

        if top_right[0] - top_left[0] > margin:
            top_right = old_top_right

        if bottom_left[0] - bottom_right[0] > margin:
            bottom_right = old_bottom_right

        if bottom_right[0] - bottom_left[0] > margin:
            bottom_left = old_bottom_left

        if top_left[1] - bottom_left[1] > margin:
            top_left = old_top_left

        if bottom_left[1] - top_left[1] > margin:
            bottom_left = old_bottom_left

        if top_right[1] - bottom_right[1] > margin:
            bottom_right = old_bottom_right

        if bottom_right[1] - top_right[1] > margin:
            top_right = old_top_right
```

Afisam in imagine colturile cu buline rosii.

```
    width = 1500
    height = 1500

    image_copy = cv.imread(path)
    cv.circle(image_copy,tuple(top_left),20,(0,0,255),-1)
    cv.circle(image_copy,tuple(top_right),20,(0,0,255),-1)
    cv.circle(image_copy,tuple(bottom_left),20,(0,0,255),-1)
    cv.circle(image_copy,tuple(bottom_right),20,(0,0,255),-1)
    show_image(f"6. Detected corners: {path[-8:]}",image_copy)
```

Decupam imaginea cu colturile detectate, care va fi un patrat de dimensiune 1500x1500, obtinand perspectiva ce ne va ajuta sa detectam pozitia pieselor de domino pe tabla.

```
puzzle_corners = np.array([[top_left],[top_right], [bottom_right],[bottom_left]],
dtype=np.float32)
    destination_of_puzzle = np.array([[0,0],[0,width],[height,width],[height,0]],
dtype=np.float32)
    perspective_transform = cv.getPerspectiveTransform(puzzle_corners,
destination_of_puzzle)
    image = cv.imread(path)
    result = cv.warpPerspective(image, perspective_transform, (width, height))

    return result, corners_coord
```

5. Functia clasifica_cifra primeste ca argument fasia de imagine cu jumatate de piesa de domino. Folosind tehnica de pattern matching, functia va returna valoarea casutei de domino.

```
def clasifica_cifra(patch):
        maxi=-np.inf
        poz=-1
        for j in range(1, 7):
            img_template = cv.imread('./templates/'+str(j)+'.jpg')
            img_template = cv.cvtColor(img_template, cv.COLOR_BGR2GRAY)

            corr = cv.matchTemplate(patch, img_template,  cv.TM_CCOEFF_NORMED)
            corr = np.max(corr)
            if corr>maxi:
                maxi=corr
                poz=j

            img_template = cv.rotate(img_template,cv.ROTATE_90_CLOCKWISE)
            corr = cv.matchTemplate(patch, img_template,  cv.TM_CCOEFF_NORMED)
            corr=np.max(corr)
            if corr>maxi:
                maxi=corr
                poz=j

            img_template = cv.rotate(img_template,cv.ROTATE_90_CLOCKWISE)
            corr = cv.matchTemplate(patch, img_template,  cv.TM_CCOEFF_NORMED)
            corr=np.max(corr)
            if corr>maxi:
                maxi=corr
                poz=j

            img_template = cv.rotate(img_template,cv.ROTATE_90_CLOCKWISE)
```

```
            corr = cv.matchTemplate(patch, img_template,  cv.TM_CCOEFF_NORMED)
            corr=np.max(corr)
            if corr>maxi:
                maxi=corr
                poz=j

        return poz
```

6. Functia determina_configuratie_careu_ocifre returneaza o matrice 15x15 care reprezinta pozitiile de pe tabla jocului. Elementele matricei sunt = ‘o’ daca nu este nicio piesa de domino pe pozitia respectiva, iar in caz contrar un numar 0-6 corespunzator valorii domino-ului.

```
def determina_configuratie_careu_ocifre(img, thresh, lines_horizontal, lines_vertical,
nr_patches):

    matrix = np.empty((15, 15), dtype='str')
    matrix.fill('o')
    l = []
    offset = 20

    for i in range(len(lines_horizontal) - 1):
        for j in range(len(lines_vertical) - 1):

            y_min = lines_vertical[j][0][0] + offset
            y_max = lines_vertical[j + 1][1][0] - offset
            x_min = lines_horizontal[i][0][1] + offset
            x_max = lines_horizontal[i + 1][1][1] - offset

            patch = thresh[x_min:x_max, y_min:y_max].copy()
            medie_patch = np.mean(patch)
            l.append((medie_patch, i, j))

    l.sort(key= lambda x: x[0])

    for k in range(nr_patches):

        _, i, j = l[k]

        y_min = lines_vertical[j][0][0] + offset
        y_max = lines_vertical[j + 1][1][0] - offset
        x_min = lines_horizontal[i][0][1] + offset
        x_max = lines_horizontal[i + 1][1][1] - offset

        patch_orig = img[x_min-offset:x_max+offset, y_min-offset:y_max+offset].copy()
        patch_orig = cv.cvtColor(patch_orig,cv.COLOR_BGR2GRAY)
```

```
        patch = thresh[x_min:x_max, y_min:y_max].copy()
        medie_patch = np.mean(patch)

        if medie_patch < 1:
            matrix[i][j] = 0
        else:
            matrix[i][j] = clasifica_cifra(patch_orig)

    return matrix
```

7. Functia main:
   Initial declaram pozitiile romburilor de pe tabla si traseul pionilor care sunt la fel in toate jocurile, respectiv folder-ul din care citim imagini si directorul in care vom scrie output-ul.

Vom folosi o matrice care ne va ajuta sa extragem ultima piesa adaugata pe tabla.

```
for nr_joc in range(1, 6):

        poz_pion_1 = -1
        poz_pion_2 = -1

        evolution_matrix = np.empty((15, 15), dtype='str')
        evolution_matrix.fill('o')
        corners_coord = []
        mutari_path = f'./{folder_imagini}/{nr_joc}_mutari.txt'
        fisier_mutari = open(mutari_path, 'r')
```

Detectam pentru fiecare poza matricea cu domino-uri din care vom “elimina” elementele care se afla si in evolution_matrix, astfel ramanem fix cu piesa noua de domino.

```
for i in range(1, 21):
            if i < 10:
                nume_fara_extensie = f'{nr_joc}_0{i}'
            else:
                nume_fara_extensie = f'{nr_joc}_{i}'
            path = f'./{folder_imagini}/{nume_fara_extensie}.jpg'

            result, aux_corners = extrage_careu(path, corners_coord)

            corners_coord = aux_corners

            lines_horizontal = []
            for j in range(0, 1500, 99):
```

```
                l=[]
                l.append((0, j))
                l.append((1500, j))
                lines_horizontal.append(l)
            lines_vertical = []
            for j in range(0, 1501, 100):
                l=[]
                l.append((j, 0))
                l.append((j, 2000))
                lines_vertical.append(l)

            result = domino_hsv(result)

            _, thresh = cv.threshold(result, 127, 255, cv.THRESH_BINARY_INV)

            matrix = determina_configuratie_careu_ocifre(result, thresh, lines_horizontal,
lines_vertical, i * 2)

            flag = False
            gasit_domino_1 = False
            solutie = ''
            scor_runda = 0
            puncte_pion_1 = 0
            puncte_pion_2 = 0
```

Gasim cele 2 casute ale piesei de domino plasate pe tabla si le salvam valoarea si pozitia din matrice. Pentru a indrepta anumite erori provocate de inacuratetea template matching-ului folosim informatia ca 2 piese invecinate trebuie sa aiba aceeasi valoare. Astfel, daca detectam ca vecinul piesei plasate la pasul curent are alta valoare, atunci valoarea piesei noastre curente va prelua valoarea vecinului.

```
            for j in range(15):
                for k in range(15):
                    if matrix[j][k] != 'o' and evolution_matrix[j][k] == 'o':
                        evolution_matrix[j][k] = matrix[j][k]
                        if not flag:
                            flag = True
                            save_pos = (j, k)
                            if j > 0 and evolution_matrix[j-1][k] != 'o' and
evolution_matrix[j-1][k] != matrix[j][k]:
                                evolution_matrix[j][k] = evolution_matrix[j-1][k]

                            if j < 14 and evolution_matrix[j+1][k] != 'o' and
evolution_matrix[j+1][k] != matrix[j][k]:
                                evolution_matrix[j][k] = evolution_matrix[j+1][k]
```

```
                            if k > 0 and evolution_matrix[j][k-1] != 'o' and
evolution_matrix[j][k-1] != matrix[j][k]:
                                evolution_matrix[j][k] = evolution_matrix[j][k-1]

                            if k < 14 and evolution_matrix[j][k+1] != 'o' and
evolution_matrix[j][k+1] != matrix[j][k]:
                                evolution_matrix[j][k] = evolution_matrix[j][k+1]

                        else:

                            if j > 0 and evolution_matrix[j-1][k] != 'o' and
evolution_matrix[j-1][k] != matrix[j][k] and (j-1, k) != save_pos:
                                evolution_matrix[j][k] = evolution_matrix[j-1][k]

                            if j < 14 and evolution_matrix[j+1][k] != 'o' and
evolution_matrix[j+1][k] != matrix[j][k] and (j+1, k) != save_pos:
                                evolution_matrix[j][k] = evolution_matrix[j+1][k]

                            if k > 0 and evolution_matrix[j][k-1] != 'o' and
evolution_matrix[j][k-1] != matrix[j][k] and (j, k-1) != save_pos:
                                evolution_matrix[j][k] = evolution_matrix[j][k-1]

                            if k < 14 and evolution_matrix[j][k+1] != 'o' and
evolution_matrix[j][k+1] != matrix[j][k] and (j, k+1) != save_pos:
                                evolution_matrix[j][k] = evolution_matrix[j][k+1]

                        solutie += f'{j+1}{chr(65 + k)} {evolution_matrix[j][k]}\n'

                        if not gasit_domino_1:
                            gasit_domino_1 = True
                            lin_domino_1, col_domino_1 = j, k
                            val_domino_1 = int(evolution_matrix[j][k])
                        else:
                            lin_domino_2, col_domino_2 = j, k
                            val_domino_2 = int(evolution_matrix[j][k])
```

In ultima faza, aflam din fisierul de mutari al cui este randul (jucatorul 1 sau 2), calculam punctajul conform instructiunilor jocului si afisam rezultatele in fisierul text corespunzator.

```
              linie_fisier = fisier_mutari.readline()
            start_index = linie_fisier.find('player') + 6
            jucator_rand = linie_fisier[start_index:start_index+1]

            if (romburi_tabla[lin_domino_1][col_domino_1] +
romburi_tabla[lin_domino_2][col_domino_2]) > 0:
                if val_domino_1 == val_domino_2:
```

```
                    coeff = 2
                else:
                    coeff = 1
                scor_runda += (romburi_tabla[lin_domino_1][col_domino_1] +
romburi_tabla[lin_domino_2][col_domino_2]) * coeff

            if poz_pion_1 > -1 and (traseu_pioni[poz_pion_1] == val_domino_1 or
traseu_pioni[poz_pion_1] == val_domino_2):
                puncte_pion_1 = 3

            if poz_pion_2 > -1 and (traseu_pioni[poz_pion_2] == val_domino_1 or
traseu_pioni[poz_pion_2] == val_domino_2):
                puncte_pion_2 = 3

            if jucator_rand == '1':
                scor_runda += puncte_pion_1
                poz_pion_1 += scor_runda
                poz_pion_2 += puncte_pion_2
            else:
                scor_runda += puncte_pion_2
                poz_pion_2 += scor_runda
                poz_pion_1 += puncte_pion_1

            solutie += str(scor_runda)

            result_path = f'./{folder_rezultate}/{nume_fara_extensie}.txt'
            with open(result_path, 'w') as file:
                file.write(solutie)
```