UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

DEPARTAMENTUL DE CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

DISCIPLINA CONCEPTE SI APLICATII IN VEDEREA ARTIFICIALA

Calculator automat de scor pentru jocul Double Double Dominoes

- PROIECT -

STUDENT: HOLMANU ANTONIO-MARIUS

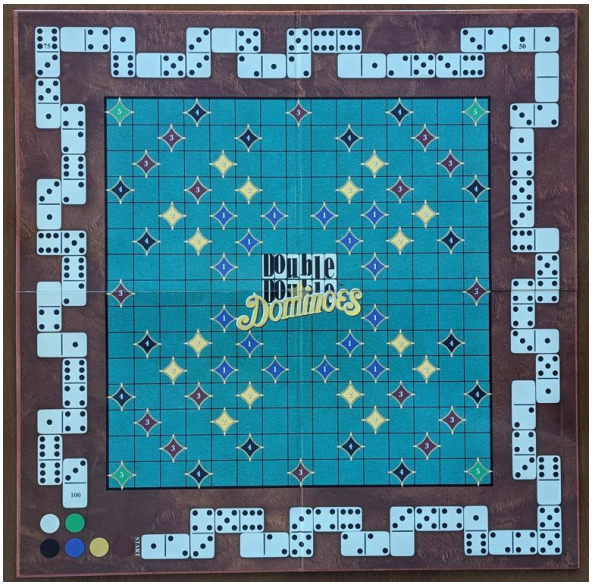
GRUPA: 462

BUCUREȘTI

2023

1. Operatii preliminare task-urilor

1.1. Extragerea informatiilor de pe tabla de joc



Am inceput proiectul prin “hardcodarea” tablei de joc si a drumului pe care trebuie sa il parcurga pionii jucatorilor.

```
#  drumul jucatorilor hardcodat
drum = [1, 2, 3, 4, 5, 6, 0, 2, 5, 3, 4, 6, 2, 2, 0, 3, 5, 4, 1, 6, 2, 4, 5, 5, 0, 6, 3, 4, 2, 0, 1,
        5, 1, 3, 4, 4, 4, 5, 0, 6, 3, 5, 4, 1, 3, 2, 0, 0, 1, 1, 2, 3, 6, 3, 5, 2, 1, 0, 6, 6, 5, 2,
        1, 2, 5, 0, 3, 3, 5, 0, 6, 1, 4, 0, 6, 3, 5, 1, 4, 2, 6, 2, 3, 1, 6, 5, 6, 2, 0, 4, 0, 1, 6,
        4, 4, 1, 6, 6, 3, 0]
```

```
#  tabla de joc hardcodata
tabla = [
    [5, 0, 0, 4, 0, 0, 0, 3, 0, 0, 0, 4, 0, 0, 5],
    [0, 0, 3, 0, 0, 4, 0, 0, 0, 4, 0, 0, 3, 0, 0],
    [0, 3, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 3, 0],
    [4, 0, 0, 3, 0, 2, 0, 0, 0, 2, 0, 3, 0, 0, 4],
    [0, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0],
    [0, 4, 0, 2, 0, 1, 0, 0, 0, 1, 0, 2, 0, 4, 0],
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3],
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [0, 4, 0, 2, 0, 1, 0, 0, 0, 1, 0, 2, 0, 4, 0],
    [0, 0, 2, 0, 1, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0],
    [4, 0, 0, 3, 0, 2, 0, 0, 0, 2, 0, 3, 0, 0, 4],
    [0, 3, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 3, 0],
    [0, 0, 3, 0, 0, 4, 0, 0, 0, 4, 0, 0, 3, 0, 0],
    [5, 0, 0, 4, 0, 0, 0, 3, 0, 0, 0, 4, 0, 0, 5]
]
```

1.2. Spatiul de culoare RGB **Vs** HSV

Spatiul de culori HSV reprezinta cele trei componente definitorii ale unei imagini (**H**ue, **S**aturation, **V**alue). Intrucat in imaginile pe care le avem de analizat exista obiecte (ex. pisele de domino), detectarea acestora tine de modul in care ele reflecta lumina, de materialul acestora ori de cat de intens sunt colorate.

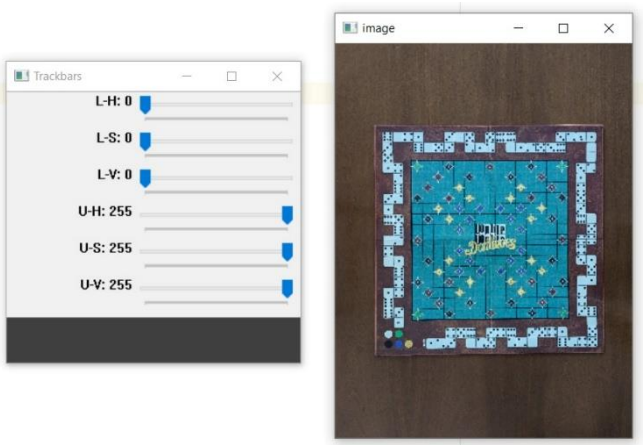
Asadar, convertirea imaginilor in acest spatiu de culori este un prim pas spre construirea calculatorului dorit.

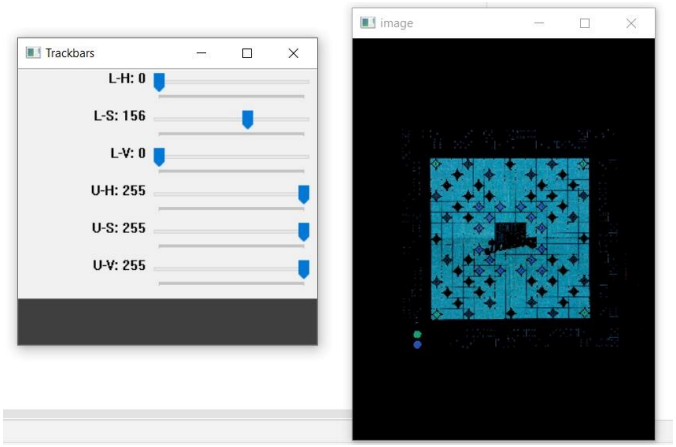
In solutia propusa de mine, am utilizat urmatoarea functie pentru acesta conversie:

```
def convertToHSV(image, lower_white, upper_white):
    hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv_image, lower_white, upper_white)
    white_parts = cv.bitwise_and(image, image, mask=mask)

    return white_parts
```

Evident, in functie de intervalele selectate: ***lower_white***, ***upper_white***, putem obtine diverse variatiuni ale imaginilor procesate. In cazul proiectului meu, pentru a gasi parametrii potriviti, am utilizat un script care m-a ajutat la ajustarea in timp real a acestora.





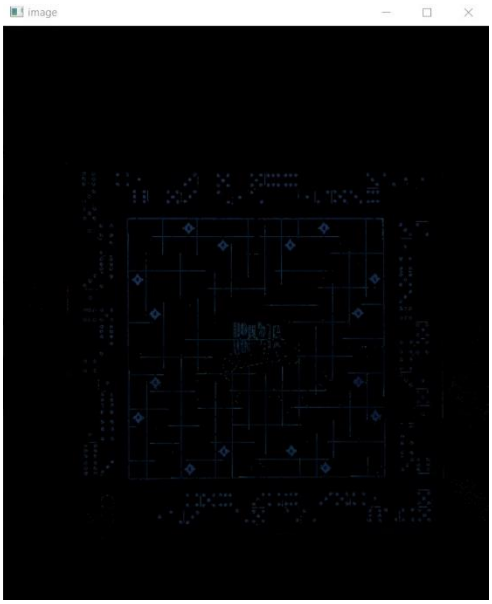
1.3. Extragerea careului de joc

Pentru gasirea celor patru puncte ce delimiteaza dreptunghiul reprezentat de careul de joc, am aplicat diverse filtre si operatori morfologici ca in exemplul dat la laborator.

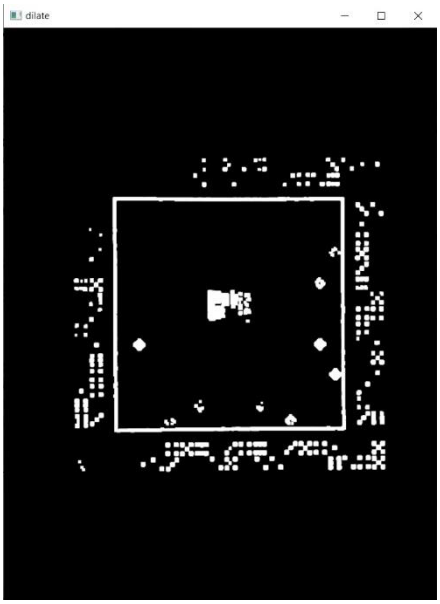
```
def findBoardCorners(image):
    #  am aplicat cateva filtre pentru detectia colturilor tablei de joc
    gray_image = cv.cvtColor(convertToHSV(image, lower_white_board, upper_white_board), cv.COLOR_BGR2GRAY)
    image_m_blur = cv.medianBlur(gray_image, ksize: 11)
    image_g_blur = cv.GaussianBlur(image_m_blur, ksize: (0, 0), sigmaX: 11)
    image_sharpened = cv.addWeighted(image_m_blur, alpha: 0.9, image_g_blur, -0.8, gamma: 0)
    _, thresh = cv.threshold(image_sharpened, thresh: 1, maxval: 255, cv.THRESH_BINARY)
    kernel = np.ones( shape: (19, 19), np.uint8)
    thresh = cv.dilate(thresh, kernel)

    contours, _ = cv.findContours(thresh, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    max_area = 0
```

Rezultatul filtrelor aplicate este urmatorul:



Conversie la HSV

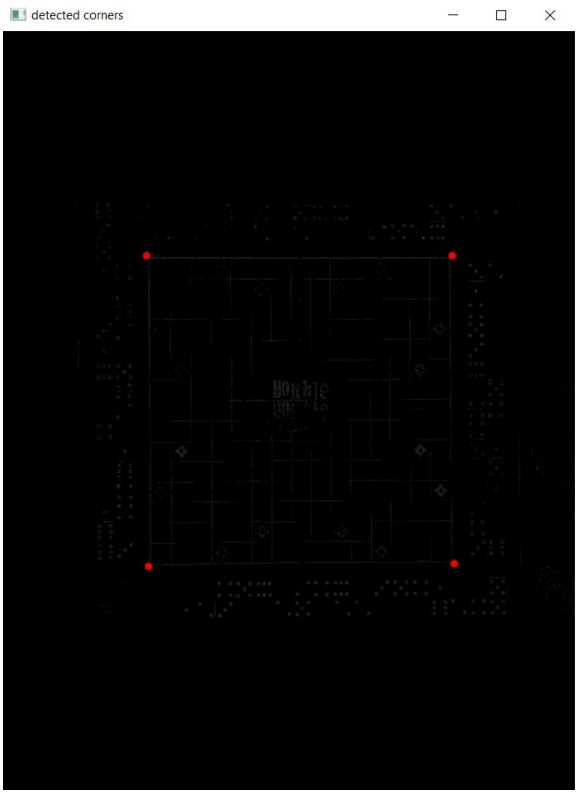


Operatie de dilatare

Prin incercari repetate in alegerea filtrelor, cat si a intervalelor de reprezentare a imaginii in spatiul de culoare HSV, am reusit sa gasesc cu precizie coordonatele careului de joc.

In continuare, pentru extragerea careului, am utilizat o schimbare de perspectiva prin functiile: **cv2.getPerspectiveTransform()** si **cv2.warpPerspective()**.

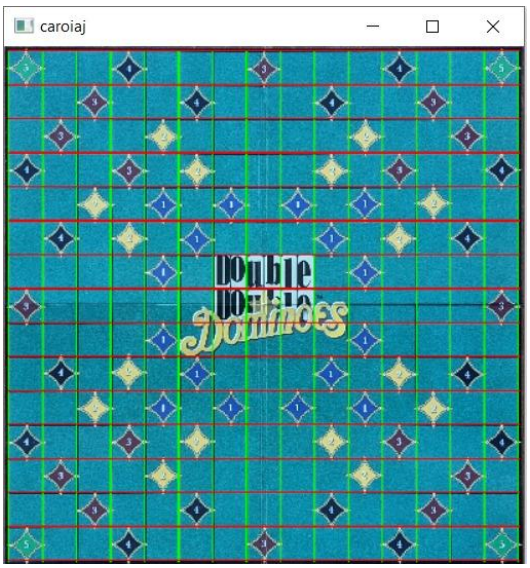
Tinand cont de dimensiunea careului din imagine (15x15 patrate), alegerea dimensiunii noului careu s-a facut ca multiplu de 15 (2250x2250).



Calculul conturului maxim

De asemenea, tot in cadrului acestui pas de extragere a careului de joc, a fost necesar sa cunosc pozitiilor liniilor verticale si orizontale pentru care se face delimitarea patratelor pe care sunt asezate piesele de domino.

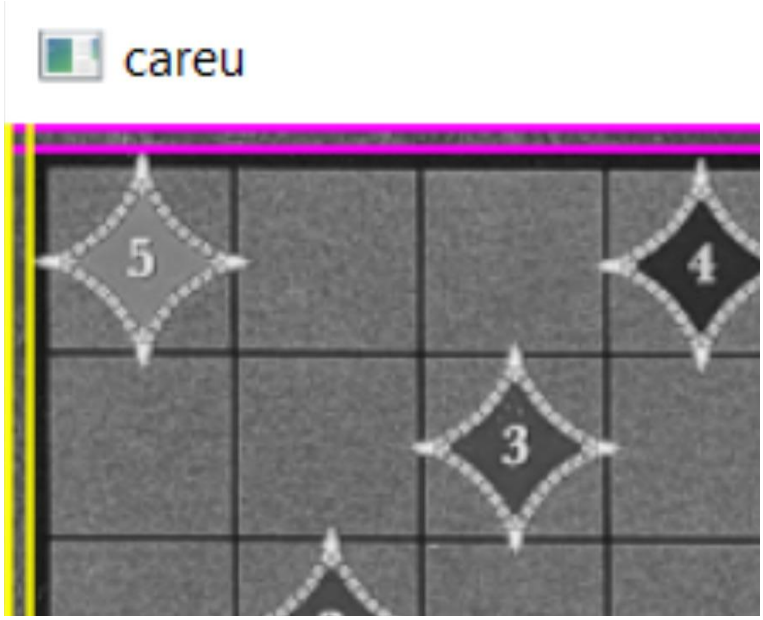
Astfel, am “hardcodat” pozitia liniilor (verticale si orizontale), iar rezultatul este urmatorul:



Este de mentionat faptul ca am avut in vedere sa las un spatiu de aproximativ 20 px pentru fiecare latura a careului fata de fereastra decupata si astfel sa incep pozitionarea liniilor decalat cu aproximativ 20 px fata de marginea ferestrei, din 2 motive:

1. Detectia colturilor careului a permis si includerea marginii careului de joc (aprox. 20 pixeli) prin aplicarea unui kernel de dimensiune 19x19 la operatia de dilatare;

2. Exista posibilitatea ca in cadrul procedurii de detectie a bulinelor de pe fiecare domino in parte, piesa sa fie pozitionat in asa fel incat o mare parte din bulina sa acopere muchia careului. Asadar, este util sa putem decupa un patch mai mare atunci cand pozitia dominoului se afla pe marginea careului.



2. Task-ul 1

2.1. Comparatie BEFORE and AFTER mutare

Strategie: Pentru a afla pozitia unei piese de domino la fiecare mutare, am hotarat sa selectez cele 15x15 patrate, individual, din imaginea de dinaintea mutarii/dupa mutare si sa compar histogramele asociate lor. Daca histogramele difera semnificativ (fata de un prag stabilit), atunci inseamna ca s-a petrecut ceva cu acel patch, ceea ce indica cu o probabilitate destul de mare ca acolo s-a asezat un capat al piesei de domino.

Cum sunt destul de multe patch-uri de comparat voi tine cont de pozitia patch-urilor pentru care s-a stabilit ca acolo s-a pus deja o piesa de domino. Am implementat aceasta metoda cu ajutorul unei matrice ce stocheaza daca la acea pozitie se afla sau nu un domino dupa multiple comparatii intre imagini. Astfel, prin aceasta abordare pot reduce numarul de comparatii efectuate si timpul de executare a programului.

De asemenea, histogramele fiecarui patch se construiesc dupa ce se converteste patch-ul in spatiul de culoare HSV. Este de mentionat faptul ca din cauza schimbarilor de lumina din mediul ambiant in care s-au facut pozele, comparatia prin histograme poate sa difere in functie de masca aplicata in transformarea BGR → HSV. Pentru o abordare mai sigura, am ales o metoda de vot prin care rezultatul final este dat de output-ul a trei masti distincte de conversie BGR → HSV, aplicate acelui patch.

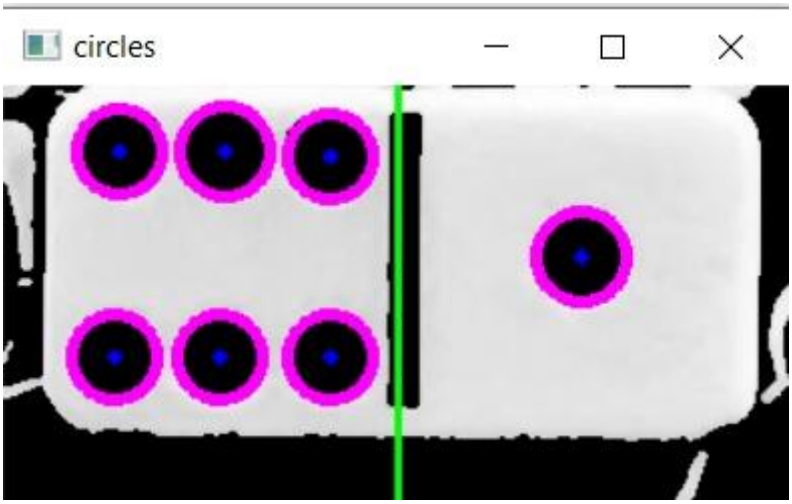
2.2. Procesarea predictiilor pozitiilor pieselor de domino

Dupa cum am mentionat, ca exista posibilitatea sa existe detectii de tip FALSE POSITIVE (din cauza a mai multor factori), doresc sa elimin pe cat posibil aceste detectii.

In functia **proceseazaPredictii (multime_puncte)** am calculat distanta Manhattan dintre punctele prezise (Manhattan – deoarece vorbim despre un caroiaj), iar acele puncte cu distanta mai mare decat **1** intre ele, vor fi considerate eliminate. Desigur, daca exista un conglomerat de puncte cu distanta 1 intre ele, voi considera ca pozitie valida pentru acel patch (patrat din careu) doar primul set de puncte gasit. Intr-adevar, nu este o solutie tocmai buna aceasta situatie, dar o solutie mai buna implica implementarea a altor mecanisme de detectie mai complicate.

3. Task-ul 2

Pentru solutionarea task-ului am utilizat transformarea Hough pentru detectia cercurilor. Astfel, pentru fiecare domino gasit la task-ul anterior, am aplicat functia **cv2.HoughCircles()**. Functia returneaza numarul de puncte gasite de pe fiecare domino, iar impartirea pentru fiecare capat al domino-ului s-a facut tinand cont de pozitia liniei delimitatoare (pe care am aproximat-o ca jumatate din dimensiunea patch-ului).



4. Task-ul 3

Rezolvarea acestui task a fost o problema de algoritmica mai mult, in care am avut nevoie de pozitia fiecarui punct bonus de pe tabla de joc si de drumul jucatorilor. Astfel, am “hardcodat” tabla de joc si drumul jucatorilor in elaborarea algoritmului.