

DOCUMENTAȚIE

-Tema 1-

Rezolvare joc Double Double Dominoes

Student: Rîncu Ștefania

December 5, 2023

Contents

1 Informatii utile	2
1.1 Spațiul HSV (Hue Saturation Value)	2
1.2 Preprocesarea imaginilor	2
2 Structura algoritmului de rezolvare	3
3 Etapa 1: Detectarea careului de joc	3
3.1 Găsirea colțurilor	3
3.2 Determinarea tablei mari	3
3.3 Augmentarea tablei mari	4
3.4 Determinarea careului de joc	5
3.5 Simulare matrice	6
4 Etapa 2: Identificarea poziției piesei plasate	6
4.1 Punct de plecare (Încercare eșuată)	7
4.2 Varianta finală	7
4.3 Importanța blurării imaginilor	10
4.4 Alegerea pragului pentru medie	11
5 Etapa 3: Determinarea numărului de pe piesă	12
5.1 Templates	12
5.2 Preprocesare	12
5.3 Template matching	12
6 Etapa 4: Calcularea scorului	13

1 Informații utile

1.1 Spațiul HSV (Hue Saturation Value)

Când am început să lucrez pentru acest proiect, am căutat să ajustez parametrii în spațiul HSV astfel încât să observ configurații ideale.

Problema era că eu modificam parametrii pentru luminozitate în majoritatea transformărilor, ceea ce făcea soluția foarte sensibilă la variații de lumină.

Ulterior, am înțeles că îmi este folositoare pentru a extrage anumite culori de interes.

Notătii pe care le voi folosi:

- **l_h, l_s, l_v:** limitele inferioare pentru intervalele hue, saturation, respectiv value; by default au valoarea 0
- **u_h, u_s, u_v:** limitele superioare pentru intervalele hue, saturation, respectiv value; by default au valoarea 255

```
l_h = l_s = l_v = 0
u_h = u_s = u_v = 255

image_hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)
mask_table_hsv = cv.inRange(image_hsv, np.array([l_h, l_s, l_v]), np.array([u_h, u_s, u_v]))
result_image = cv.bitwise_and(image_hsv, image_hsv, mask=mask_table_hsv)
```

Figure 1: *Sablonul pentru prelucrarea imaginilor în spațiul HSV*

Pasul 1: Trecem imaginea din spațiul BGR în spațiul HSV.

Pasul 2: Definim o mască binară în care valorile pixelilor sunt setate la 255 (alb) dacă acestea se încadrează în intervalul HSV specificat și la 0 (negru) în caz contrar.

Pasul 3: Aplicăm masca binară definită la pasul anterior peste imaginea HSV originală, iar astfel păstrăm doar pixelii cărora le corespunde un pixel alb în mască.

1.2 Preprocesarea imaginilor

În cadrul acestui proiect, am observat faptul că este esențial ca înainte să prelucrăm o imagine să aplicăm asupra acesteia diverse operații pentru a evidenția detaliile de care avem nevoie.

Filtrul median și cel Gaussian mi-au folosit pentru a elibera zgromotul și a netezi imaginea. Astfel, pentru detectarea muchiilor sau înainte de aplicarea unui threshold nu mai obțin atât de multe puncte împrăștiate. Aceste procedee sunt foarte bune pentru normalizarea imaginilor, deoarece o uniformizăm și nu mai avem minime sau maxime locale foarte mari.

Operatorii morfologici, dilatarea și eroziunea m-au ajutat să închid sau deschid contururi sau să elimin mici puncte din background. Pentru o imagine binarizată, determinam contururi destul de regulate astfel încât să pot extrage ceea ce caut.

În majoritatea cazurilor, adăugarea unei singure operații de preprocesare (filtrare, thresholding, eroziune, dilatare), m-a ajutat să obțin rezultate mai bune sau să rezolv problemele întâmpinate.

2 Structura algoritmului de rezolvare

1. Detectarea careului de joc
2. Identificarea poziției piesei plasate
3. Determinarea numărului de pe piesă
4. Calcularea scorului

3 Etapa 1: Detectarea careului de joc

Pentru a detecta careul de joc, prima dată extrag tabla mare, în care este încadrat acesta.

3.1 Găsirea colțurilor

În cadrul funcției **determine_corners** am folosit codul prezentat în laboratorul 6, pentru a determina colțurile celor două table prin căutarea unui contur de dimensiune maximă și memorarea extremelor stânga sus și dreapta jos. Ulterior, pe baza acestora determinăm și celelalte două puncte care alcătuiesc pătratul.

Această funcție întoarce un vector care reține cele patru seturi de coordonate ale colțurilor determinate.

3.2 Determinarea tablei mari

Pasul 1 (Prelucrare în spațiul HSV):

Pentru a detecta întreaga tablă am trecut imaginea în spațiul HSV și am crescut limita inferioară pentru hue la valoarea 24. Astfel, elimin nuanțele de maro închis și obțin o mască ce are pixeli negri acolo unde se află masa pe care este poziționată tabla.

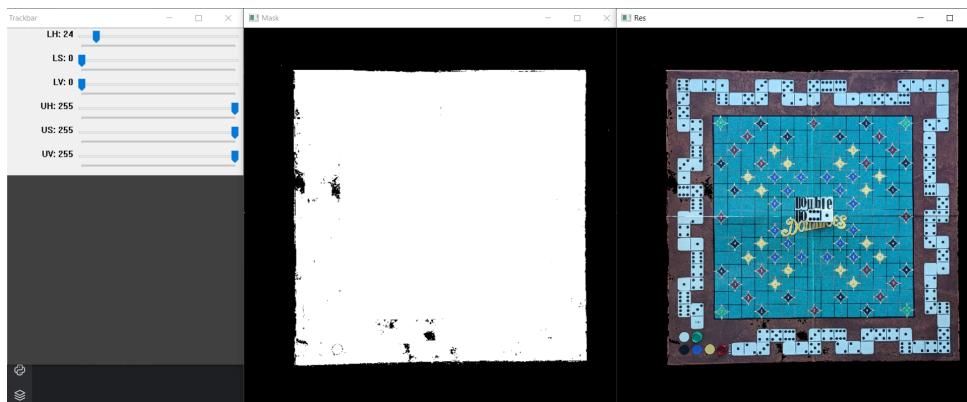


Figure 2: Mască și imaginea rezultată pentru $L_h = 24$

Pasul 2 (Procesare și prelucrare imagine):

Am preluat rezultatul și l-am trecut în format Grayscale. Prima dată am folosit un **filtru median**, pentru a reduce zgomotul din imagine. După aceea, am aplicat un **filtru Gaussian** pentru a blura imaginea și a estompa detaliile subtile, obținând o imagine mai netedă. Ulterior, pentru a accentua detaliile (**filtru de accentuare**), combin cele două imagini filtrate median și Gaussian.

Folosind un **threshold** am binarizat imaginea accentuată pentru a evidenția marginile

tablei. Peste imaginea binară, aplic **operatorul Canny** pentru a detecta muchiile. Deoarece am sesizat că muchiile detectate sunt foarte segmentate (vezi: *Figure 3a*) am aplicat peste acestea operația de **dilatare**, pentru a mări zonele de pixeli albi și a le conecta (vezi: *Figure 3b*).

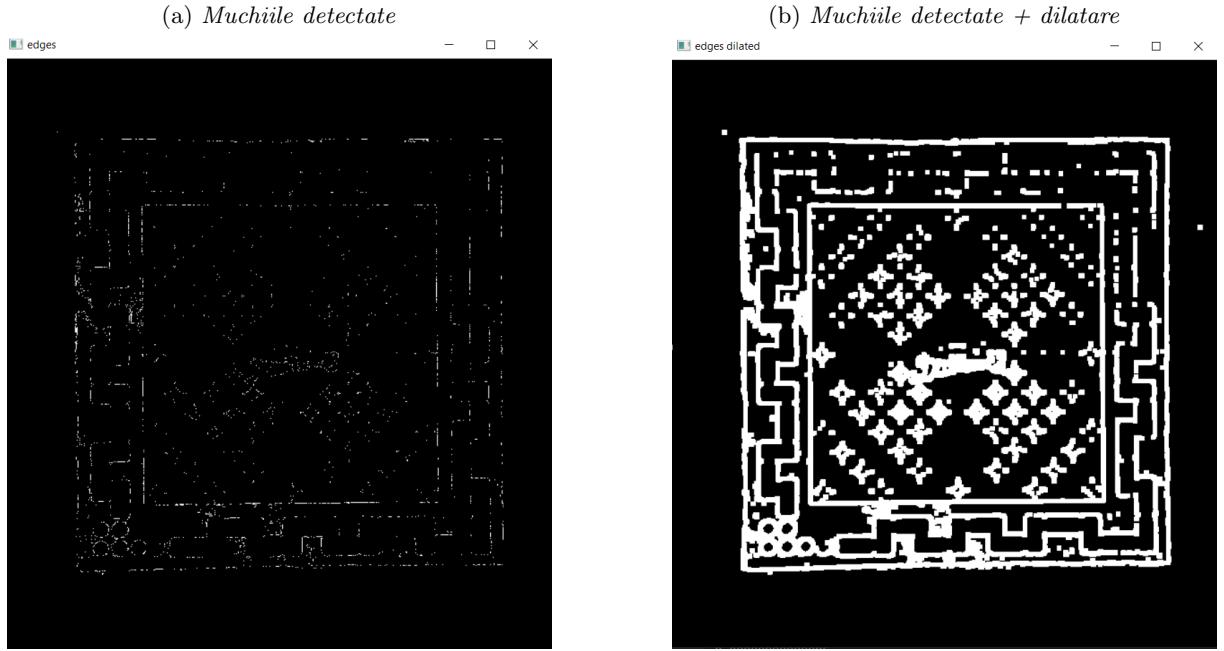


Figure 3: *Muchii extrase și muchii dilatate*

Pasul 3 (Extragere contur și determinare colțuri):

Am apelat funcția ***determine_corners*** descrisă mai sus.

3.3 Augmentarea tablei mari

Am observat faptul că pe marginea tablei mari, este prezent un traseu de scor (vezi: *Figure 4a*) care ulterior va duce la multe rezultate eronate în etapa de detecție a careului de joc. De aceea, tai mai mulți pixeli din exteriorul tablei (vezi: *Figure 4b*)

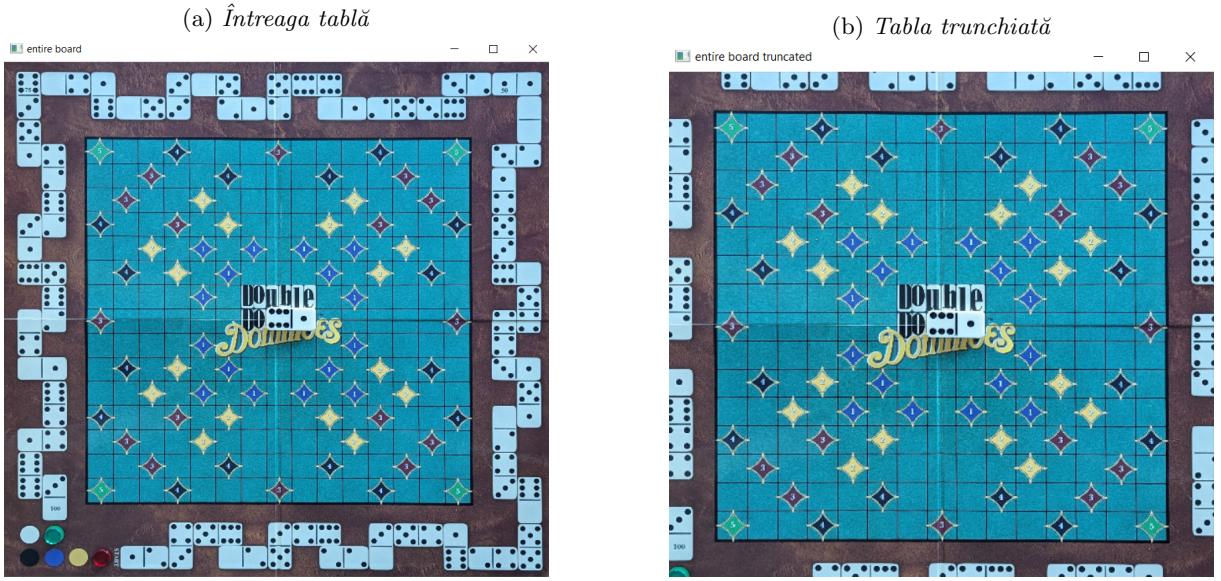


Figure 4: *Întreaga tablă vs preluarea conținutului de interes*

3.4 Determinarea careului de joc

Pasul 1 (Prelucrare în spațiul HSV):

Pentru a detecta careul de joc, preiau tabla din *Figure 4b*, peste care aplic un filtru median, urmat de un filtru Gaussian, cu scopul de a **normalize și netezi** imaginea.

De data aceasta, abordarea a fost de a păstra doar nuanțele din sfera albastru-verde, adică centrul tablei de joc. De aceea, am limitat intervalul nuantă la [30, 134].

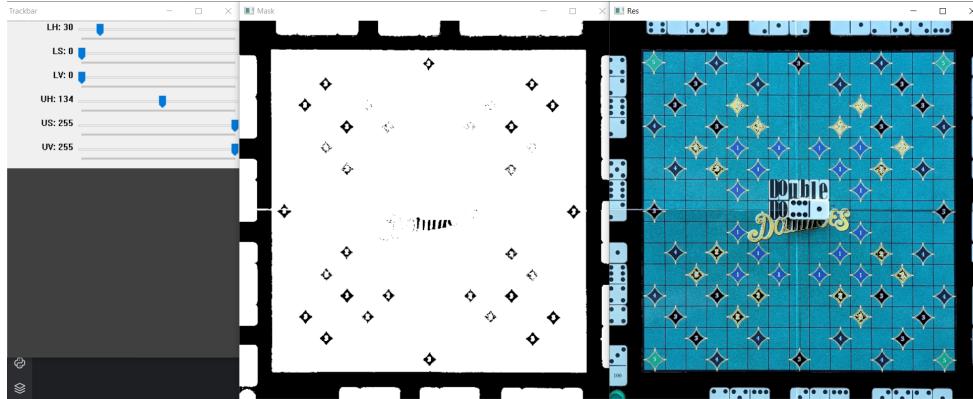


Figure 5: *Masca și imaginea rezultată pentru $l_h = 30$ și $u_h = 134$*

Pasul 2 (Procesare și prelucrare imagine):

Am urmat aceeași tipar ca la extragerea tablei mari. Diferă doar anumiți parametrii, cum ar fi alegera threshold-ului. În plus, aplic încă un filtru median peste imaginea dilatătă, deoarece am sesizat cazuri în care tabela de scor se atinge de marginea tablei de joc după dilatare (vezi: *Figure 6*)

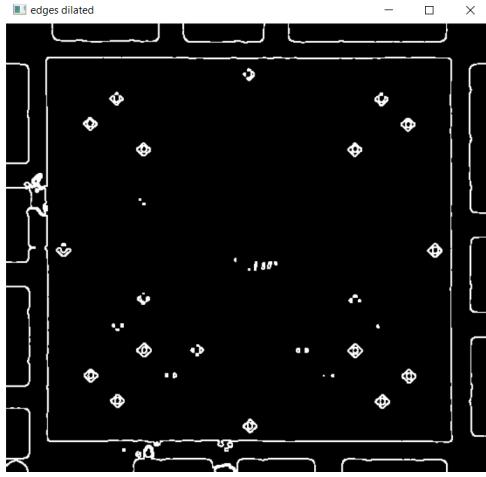


Figure 6: Muchiile detectate + dilatare

Pasul 3 (Extragere contur și determinare colțuri):

Am apelat funcția ***determine_corners*** descrisă mai sus.

3.5 Simulare matrice

Pe parcursul rezolvării algoritmului, este util să împărțim imaginea tablei de joc într-o matrice de 15x15. Astfel, putem identifica în ce celule a fost plasată o piesă. Pentru aceasta, când preiau imaginea careului de joc, o redimensionez la 150x150.

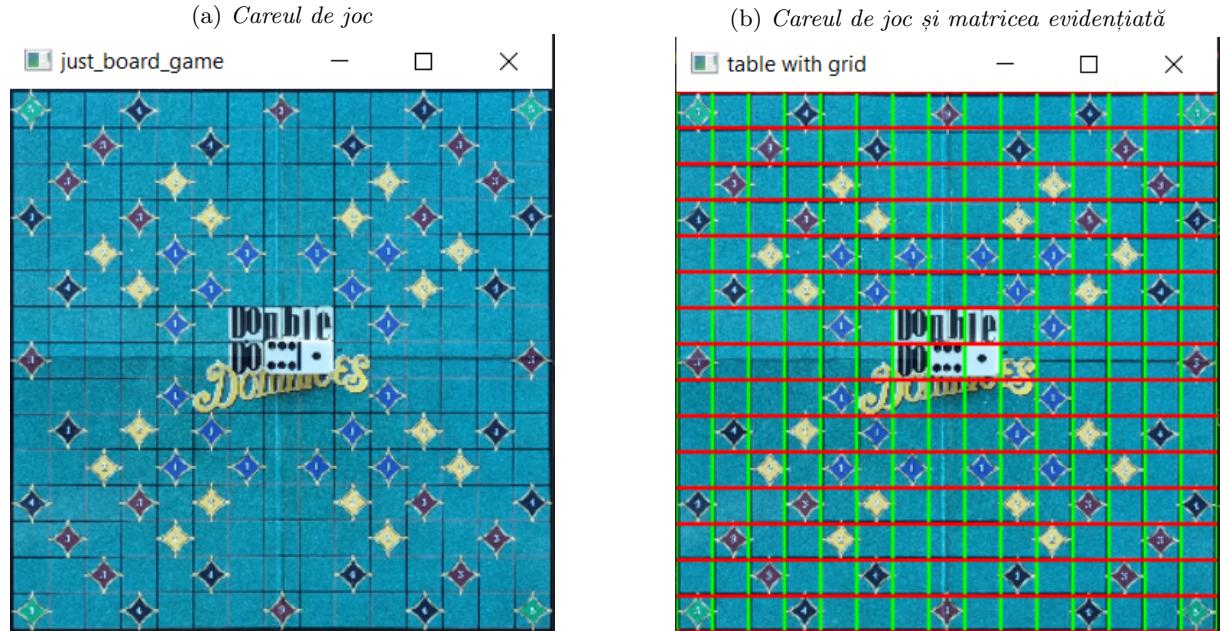


Figure 7: Evidențierea careului de joc și a celulelor sale

4 Etapa 2: Identificarea poziției piesei plasate

Găsirea poziției unde a fost asezată fiecare piesă pe tabla de joc a reprezentat cea mai dificilă cerință din rezolvarea temei, deoarece am detectat foarte multe rezultate fals pozitive (scrisul din mijlocul tablei).

Primul gând a fost să merg în spațiul HSV și să aplic o mască neagră peste piese. Dar pentru asta adaptam intervalele pentru luminositate și/sau saturatie, deci implementam un algoritm sensitiv la variații ale acestora. Am încercat să modific intervalul nuanței pentru a detecta albastrul tablei. Dar scrisul din mijlocul tablei nu se dădea bătut și apărea în rezultate.

4.1 Punct de plecare (Încercare eșuată)

Una dintre variantele care aproape a funcționat, avea la bază ideea că atunci când **scad din tabla curentă tabla de joc anterioară**, determin celulele în care a fost pusă nouă piesă. Mai exact, preiau **diferența** dintre două imagini consecutive, aplic un **filteru median** peste aceasta și apoi o **binarizez** prin impunerea unui threshold. După aceea, foloseam un **operator morfologic de deschidere** pentru a elimina pixelii din foreground și a umple spațiile albe unde se află nouă piesă. Media pentru patch pe care o setasem trebuia să depășească **0**, deoarece aveam imaginea aproape neagră pentru anumite configurații.

Totuși, în această rezolvare apăreau **probleme**:

1. Din prima tablă de joc scădeam tabla auxiliară (cea fără piese), iar orice variație a luminosității se observa destul de ușor. De aceea, pentru foarte multe jocuri, la prima mutare identificam 4 sau 5 piese. Iar dacă modificam un threshold pentru imagini luminoase, atunci nu mai găseam piesele pentru imagini întunecate.
2. Când piesele se mutau mai sus sau mai jos de la o rundă la alta (jocul 4) rămâneau pixeli în foreground pe care îi număram.

4.2 Varianta finală

Soluția pe care am implementat-o preia ideea de bază de mai sus și aduce în plus anumite îmbunătățiri.

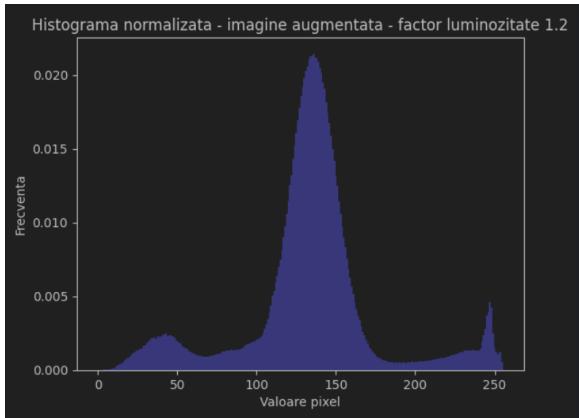
În primul rând, am observat că luminositatea și modul în care pică aceasta se modifică și în cadrul unui joc, de la o rundă la alta. De aceea, am considerat că pot lua tabla auxiliară (cea fără piese) ca punct de referință pentru toate celelalte imagini, fiind un **element fix**. Deci, **pe tot parcursul unui joc**, voi face **diferența dintre tabla curentă și tabla goală**.

Pentru a trata complet cazul în care luminositatea variază, am adăugat o funcție în cadrul căreia caut o **interpolare liniară** între imaginea tablei auxiliare și cea a primei table dintr-un joc. Astfel, determin **histogramele pentru intensitățile pixelilor** din fiecare imagine și le **normalizez** prin împărțirea rezultatului la 2550000, adică dimensiunea imaginii (știu sigur că e fixă). Pentru fiecare histogramă normalizată, calculez **suma cumulativă**, cu scopul de a determina distribuția cumulativă a intensității pixelilor. Pe baza acestora, stabilesc o corespondență între intensitățile pixelilor dintre cele două imagini, iar astfel reușesc să egalizez histogramele.

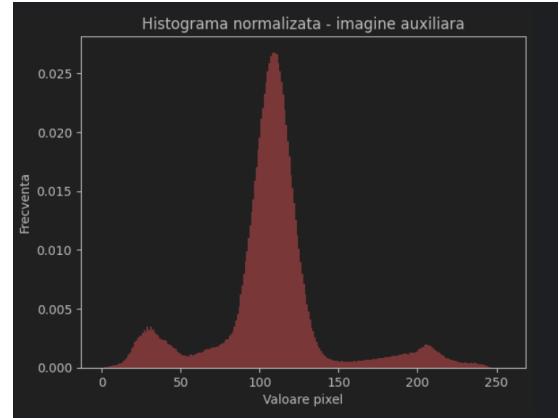
Mai exact, pentru fiecare primă tablă dintr-un joc realizez **ajustarea histogramei** pentru a face ca distribuția intensităților pixelilor să se potrivească cu distribuția din tabla auxiliară.

Pe baza acestei interpolări pe care o determin la prima rundă a fiecărui joc, aplic transformări de tipul **Look-Up-Table (LUT)** asupra imaginilor viitoare, cu scopul de a corecta intensitățile pixelilor.

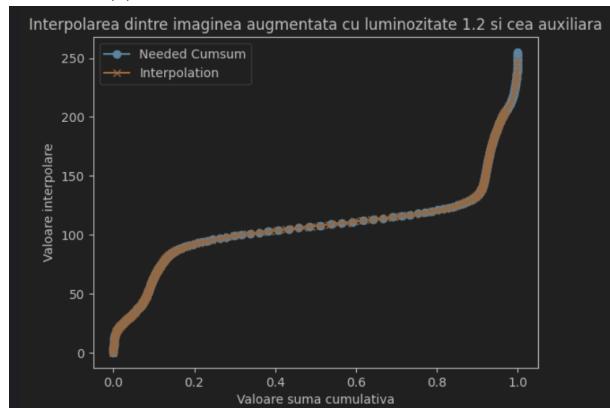
(a) Histogramă normalizată - imagine augmentată - 1.2 factor luminositate



(b) Histogramă normalizată - imaginea auxiliară



(c) Grafic interpolare pentru cele două



(d) Imaginea augmentată - factor 1.2 luminozitate

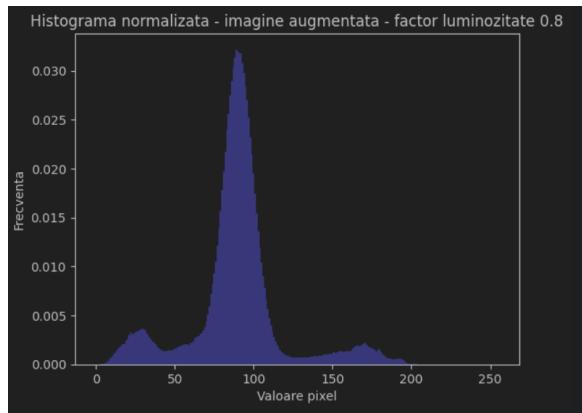


(e) Imaginea după egalizarea histogramei

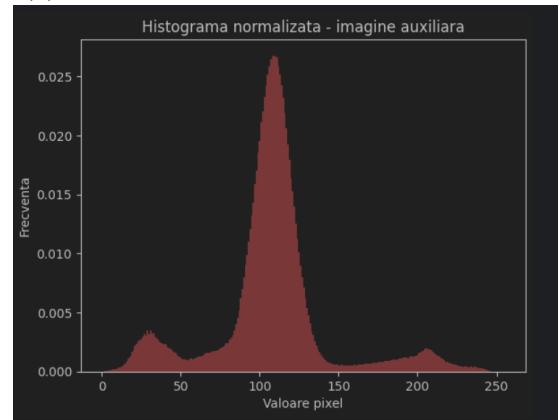


Figure 8: Vizualizare preprocesare imagine mult prea luminoasă

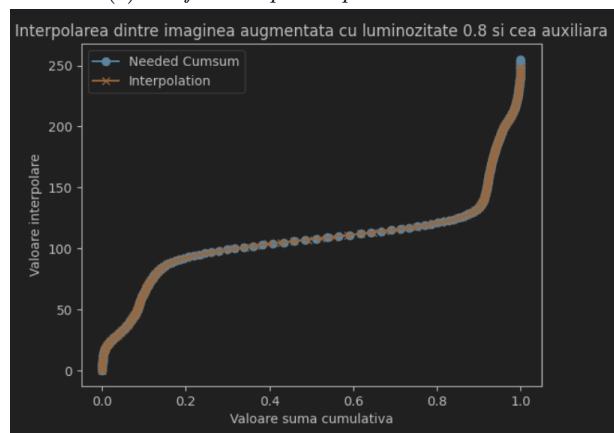
(a) Histogramă normalizată - imagine augmentată - 0.8 factor luminositate



(b) Histogramă normalizată - imaginea auxiliară



(c) Grafic interpolare pentru cele două



(d) Imaginea augmentată - factor 0.8 luminositate



(e) Imaginea după egalizarea histogramei



Figure 9: Vizualizare preprocesare imagine mult prea întunecată

Așadar, pe parcursul unui joc, preiau fiecare imagine și o modific astfel încât să aibă o distribuție a intensităților pixelilor asemănătoare tablei auxiliare. După aceea, realizez diferența dintre tabla curentă, pe care am augmentat-o și cea auxiliară, iar pentru fiecare patch calculez media. Dacă această medie depășește un prag, atunci consider că în celula respectivă a fost plasată o piesă.

4.3 Importanța blurării imaginilor

Initial, după ce aplicam interpolarea găsită pentru egalizarea histogramei imaginii curente, efectuam direct diferența dintre cele două table. Algoritmul mergea perfect pe toate teste, dar media din patch-uri era destul de mare și pentru celule care nu prezintau interes (vezi: *Figure 10*).

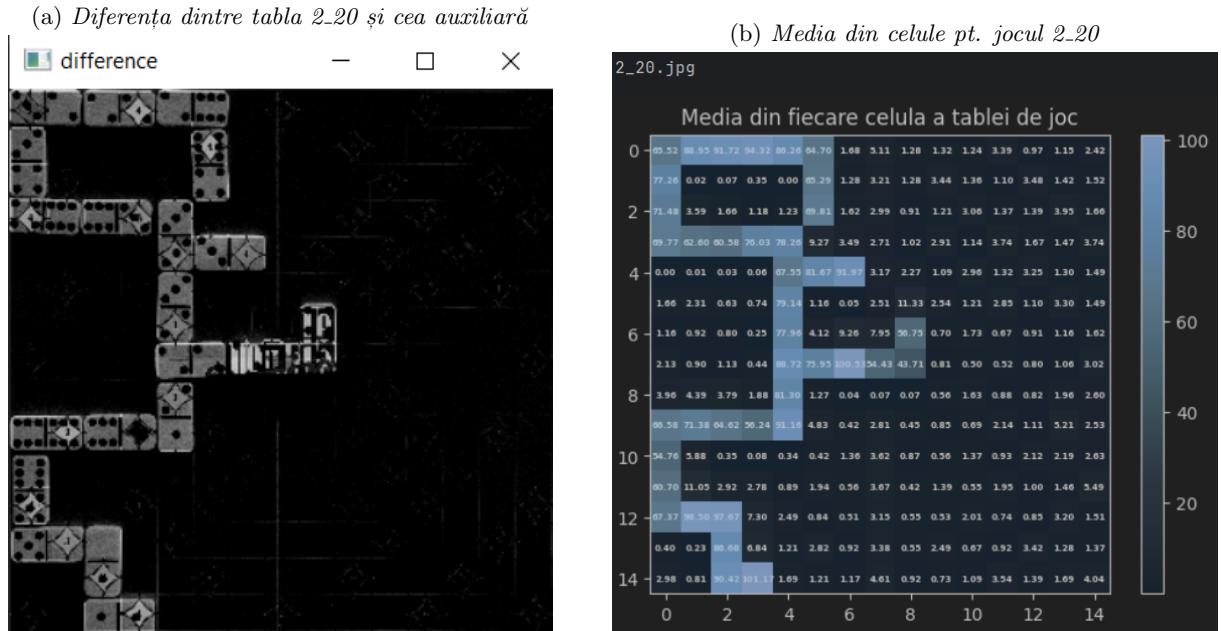


Figure 10: Vizualizarea tablei de joc și a matricei de medie

Pentru a scădea semnificativ aceste medii, înainte de a efectua diferență am aplicat un **filtru median** peste ambele imagini, cu scopul de a elimina zgomotul. Apoi, peste diferență am folosit un **operator morfologic de deschidere**, pentru a **elimina** pixelii albi din **background**. Deoarece pe imaginile închise la culoare aveam configurații pentru care nu găseam piesele, am adăugat și o **dilatare**. Din acest motiv, media nu o mai efectuez pentru întreg patch-ul, ci restrâng intervalul spre mijlocul celulei.

(a) Diferența dintre tabla 2_20 și cea auxiliară blurată



(b) Diferența dintre tabla 2_20 și cea auxiliară + operator de deschidere



(c) Media din celule pt. jocul 2_20 + blurare

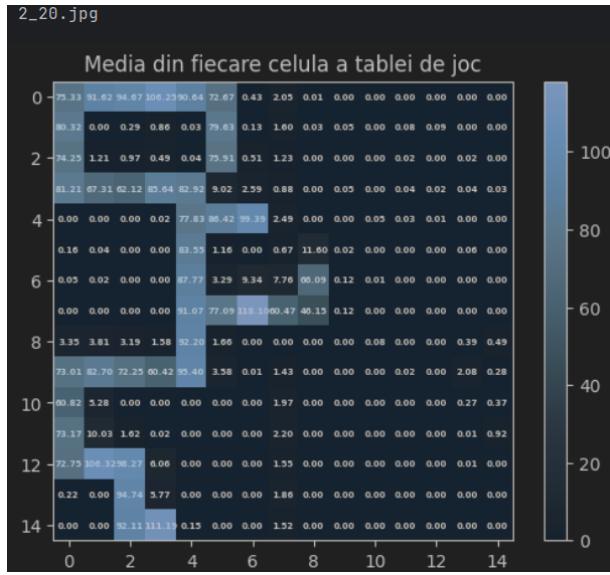


Figure 11: Vizualizarea tablei de joc și a matricei de medie pentru imagini blurate

Dacă analizăm Figure 10a și Figure 11c observăm diferența dintre matricele de medii obținute pentru același joc, și felul în care folosind o filtrare reușim să reducem semnificativ media pentru celulele din exterior, care nu prezintă interes.

4.4 Alegerea pragului pentru medie

Pragul pe care l-am ales pentru a determina dacă patch-ul curent conține sau nu o piesă de joc a fost determinat în **mod exclusiv (forță brută)**. Am afișat pentru fiecare joc, cea mai mare medie pe care o găsesc pe tablă și nu ar trebui inclusă în rezultate. Pentru imagini luminoase, am obținut valoarea 21. Pe de altă parte, pentru imagini intunecate, cel mai mic prag pentru care o piesă nu era detectată este 31. Astfel, am ales să pun limita pentru media patch-ului **29**.

5 Etapa 3: Determinarea numărului de pe piesă

5.1 Templates

Pentru a clasifica numărul de pe piesă de domino, am selectat 17 template-uri care corespund tuturor orientărilor posibile pe care le pot avea piesele. Pentru piesele 0, 1, și 2 am selectat mai multe template-uri deoarece pe acestea le identificam greșit cel mai frecvent.

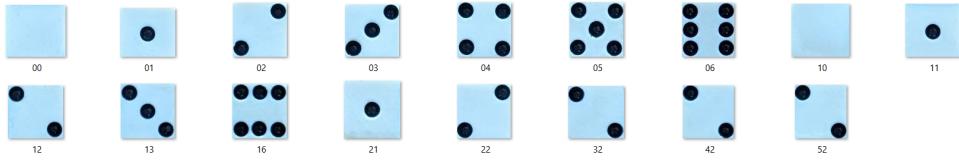


Figure 12: Template-uri folosite

Din *Figure 12* se poate observa faptul că am numit sugestiv template-urile, astfel încât ultima cifră să corespundă cifrei de pe piesă.

5.2 Preprocesare

Atât template-urilor, cât și patch-urilor ce conțin jumătatea de piesă de domino pentru care vreau să identific numărul asociat le aplic aceeași preprocesare, și anume le trec în spațiul HSV, și reduc intervalul specific pentru nuanță la [82, 255], iar pentru saturăție la [0, 108]. Astfel, păsterez din imaginea originală, regiunile ce conțin nuanțe cu o intensitate mai scăzută și păstrează culorile cu tonuri reci.

Deși intuitiv, nu ar mai trebui să adaptăm intervalul pentru nuanță, deoarece piesa este alb negru, dar am considerat cazul în care atunci când iau un patch, preiau și puțin din tablă.

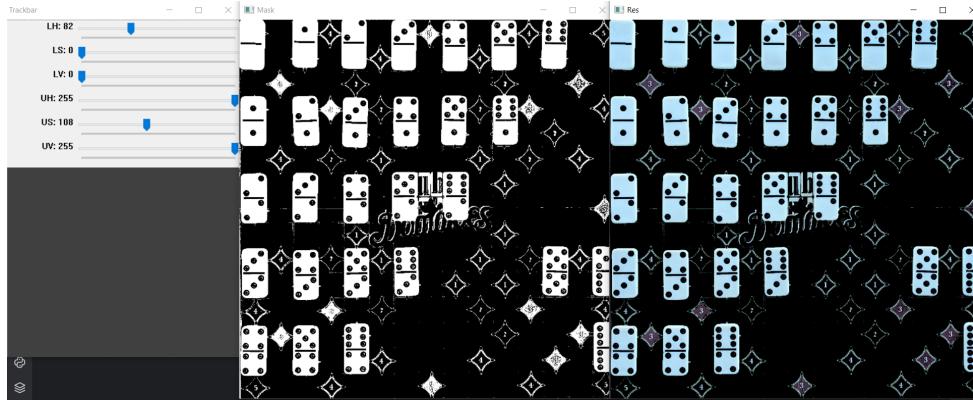


Figure 13: Masca și imaginea rezultată pentru $Lh = 82$ și $us = 108$

Nu am eliminat transformarea în spațiul HSV, deoarece după ce realizez transformarea LUT asupra fiecărei table de joc, intervalele se potrivesc. De asemenea, template-urile sunt extrase din imaginile auxiliare care conțin toate piesele, care se aseamană cu cea goală.

5.3 Template matching

Pentru a determina numărul de pe piesă de domino, prima dată preiau patch-ul care se află la coordonatele primite. Îi aplic transformarea în spațiul HSV prezentată mai sus,

iar după aceea, îl convertesc în Grayscale. Asupra acestui rezultat intermediu, aplic un filtru median, apoi eroziune, thresholding și un operator morfologic de închidere. La final, pentru a netezi imaginea rezultată, aplic un filtru Gaussian.

După aceea, iterez prin imaginile de template prezentate în *Figure 12* și le aplic preprocesarea din spațiul HSV prezentată. La fiecare iterare compar patch-ul extras (și modificat) cu template-ul curent.

Pentru funcția de Template Matching folosesc **corelația** dintre patch și template, metoda utilizată pentru măsurarea corelației fiind **cv.TM_CCOEFF_NORMED**. Pentru a găsi cea mai bună potrivire, cauți valoarea maximă din matricea de corelație rezultată și o compar cu maximele anterioare. Astfel, reușesc să determin care template este cel mai asemănător patch-ului și memorez cifra acestuia.

6 Etapa 4: Calcularea scorului

Pentru calcularea scorului, rețin o listă ce conține două liste: una pentru poziția jucătorului 1 pe tabela de scor, iar cealaltă pentru jucătorul 2.

La finalul fiecărei runde, transmit unei funcții lista pozițiilor jucătorilor pe tabla de scor, indicele jucătorului curent, coordonatele piesei adăugate în matricea de joc și numerele de pe piesă.

Accesez valorile din matricea hardcodată, care conține punctele bonus (**table_points**) corespunzătoare coordonatelor specificate și le adun pentru a calcula punctele primite. Dacă am adăugat o piesă dublă, atunci înmulțesc acest punctaj cu 2. Apoi verific dacă ultima poziție a jucătorului curent în harta de scor corespunde uneia dintre cele două numere de pe piesă, iar în caz afirmativ adaug 3 puncte. Dacă numărul total de puncte este mai mare decât 0, atunci simulez mutarea pionului pe tabela de scor prin înaintarea cu ”numărul de puncte căsuțe” și actualizez poziția jucătorului curent prin adăugarea noii poziții calculate (**ultima_poziție + nr_puncte**)

De asemenea, pentru oponent, verific dacă ultima poziție a sa în harta de scor corespunde uneia dintre cele două numere de pe piesa curentă, iar în caz afirmativ îi adaug 3 puncte prin actualizarea vectorului său de poziții.

```
def calculate_score(players_positions, current_player, coord, nr_on_piece):
    points = table_points[coord[0][0]][coord[0][1]] + table_points[coord[1][0]][coord[1][1]]

    if nr_on_piece[0] == nr_on_piece[1]:
        points *= 2

    if (outside_points[players_positions[current_player - 1][-1]] == nr_on_piece[0]
        or outside_points[players_positions[current_player - 1][-1]] == nr_on_piece[1]):
        points += 3

    if points > 0:
        players_positions[current_player - 1].append(players_positions[current_player - 1][-1] + points)

    if (outside_points[players_positions[(current_player - 2) % 2][-1]] == nr_on_piece[0]
        or outside_points[players_positions[(current_player - 2) % 2][-1]] == nr_on_piece[1]):
        players_positions[(current_player - 2) % 2].append(players_positions[(current_player - 2) % 2][-1] + 3)

    return points
```

Figure 14: Cod calculare scor