

# Double Double Dominoes(DDD)

Ghețoiu Gheorghe-Laurențiu

Concepte si Aplicatii in Vederea Artificiala - Tema 1  
01 dec 2023

## 1 Problema si etapele de rezolvare

Scopul temei este implementarea unui program de calculare a scorului pentru o varianta a jocului Double Double Dominoes folosind tehnici de Computer Vision.

1. Prelucrarea dimensiunii imaginilor si extragerea chenarului
2. Identificarea pozitiilor pieselor
3. Recunoastere numere
4. Calcularea scorului
5. Rezultate

### 1. Prelucrarea dimensiunii imaginilor si extragerea chenarului

Prima observatie cu privire la setul de date este ca imaginile au o dimensiune suficient de mare, dar doar o portiune din fiecare imagine ne intereseaza. Pentru a extrage portiunea esentiala din imagine am ales sa gasesc pentru o imagine coordonatele aproximative in care este incadrat chenarul central pe care se pun piesele. Prima data am deschis imaginile cu Opencv facand resize la fereastra in care este afisata imaginea(nu la imagine) pentru a avea coordonatele nealterate. Am folosit un callback care permite afisarea coordonatelor la care se afla cursorul la momentul apasarii unui buton. Codul de mai jos exemplifica modul de folosire al callback-ului.

Listing 1: mouse\_callback

```
1 def mouse_callback(event, x, y, flags, param):  
2     if event == cv.EVENT_LBUTTONDOWN:  
3         print(f"Coordonate: ({x}, {y})")
```

Listing 2: show\_img\_win\_resize

```
1 def show_img_win_resize(img):  
2     cv.namedWindow("img", cv.WINDOW_KEEPRATIO)
```

```

3   cv.resizeWindow("img", 900, 800)
4   cv.imshow("img", img)
5   cv.setMouseCallback("img", mouse_callback)
6   cv.waitKey(0)
7   cv.destroyAllWindows()

```

In urma apelului functiei se va deschide imaginea neredimensionata si atunci cand se apasa click stanga sunt afisate in consola coordonatele cursorului la acel moment. Pentru ca este dificil de lucrat cu imaginea in fereastra deschisa de Opencv, dar si pentru a verifica rezultatul am folosit si o aplicatie de editare a imaginilor care ofera o astfel de functionalitate(Gimp, Photoshop, Paint). Coordonatele obtinute de mine sunt

```

1160:2912 - height
732:2456 - width
-----
1724 x 1752 - noua dimensiune

```

Acestea sunt alese undeva la mijloc intre chenarul central si traseu, mai aproape de traseu. Astfel, programul nu este sensibil la mici miscari ale tablei. Din testele facute, functioneaza chiar si daca raman in imagine mici portiuni din traseul lateral.

Mai departe, pentru a elimina marginile ramase cu scopul de a obtine o tabla patrata cu dimensiuni fixe am folosit codul din laborator. Diferenta este ca am cautat contururile pe o masca obtinuta din spatiul HSV. Pentru masca din spatiul HSV am folosit codul din laboratorul de Filtrarea culorilor, considerand potrivite urmatoarele valori:

```

low_yellow = (49, 0, 0)
high_yellow = (120, 255, 255)

```

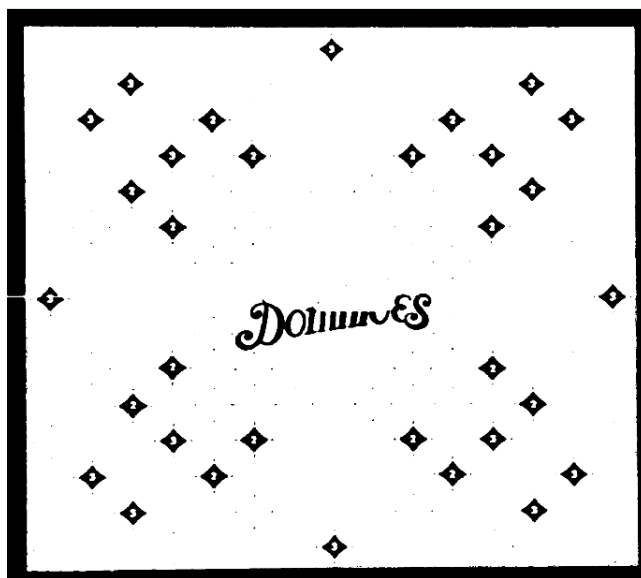


Figure 1: Masca pentru prima imagine din antrenare

Dupa cum se poate observa in **Figura 1** exista pixeli albi ce pot contribui negativ la extragerea contururilor. De exemplu linia alba din marginea stanga. Pentru a avea rezultate mai bune am aplicat operatori morfologici pe masca. Ce a functionat cel mai bine a fost eroziunea urmata de deschidere cu parametrii vizibili mai jos:

```
kernel = np.ones((3, 3), np.uint8)
erosion = cv.erode(imagine, kernel, iterations=2)
kernel = np.ones((5, 5), np.uint8)
opening = cv.morphologyEx(erosion, cv.MORPH_OPEN, kernel, iterations=2)
```

**Masca obinuta(Figure 2)** este fara linia alba de pe margine si fara cifrele din interiorul romburilor.

Pe aceasta masca am extras colturile conform conturului maxim si am folosit *getPerspectiveTransform* pentru a avea o imagine cu dimensiuni fixe. Dimensiunile au fost alese astfel incat sa fie relativ apropiate de cele obtinute dupa primul crop si sa fie divizibile cu 15 (numarul de linii si coloane ale tablei de joc). Am ales ca acestea sa fie 1635 x 1635. Imaginea rezultata in urma aplicarii celor de mai sus arata **astfel(Figure 3)**. Pentru ca mai raman margini cu pixeli complet negri vom adauga un border alb. Acesta ajuta in cazul in care o piesa este plasata pe margine, neafectand negativ suma pe care o calculam in cele ce urmeaza.

Urmatorul pas este transformarea imaginii in grayscale si apoi intr-o imagine binara folosind doua praguri in functie de valoarea din spatiul HSV. Acestea sunt obtinute in urma unor teste pe cateva dintre imagini:

Listing 3: Valori de prag

```
1  if np.sum(image_to_hsv[:, :, 2])// 10 ** 7 <= 100:  
2      warped_bin = cv.threshold(warped_gray, 130, 255,  
3          cv.THRESH_BINARY)[1]  
4  else:  
      warped_bin = cv.threshold(warped_gray, 180, 255,  
          cv.THRESH_BINARY)[1]
```

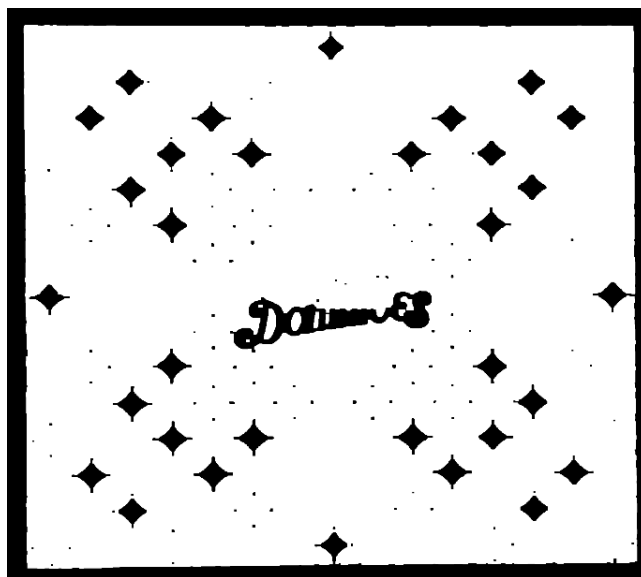


Figure 2: Masca dupa aplicarea operatorilor morfologici

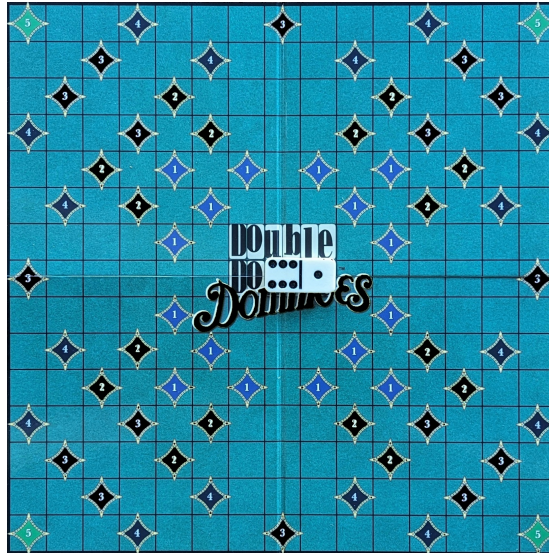


Figure 3: Imaginea transformata

## 2. Identificarea pozitiilor pieselor

Pentru a identifica pozitiile pieselor am folosit **imaginea binara** obtinuta anterior, parcurgand celula cu celula (din 109 in 109 pixeli). Asadar, imaginea este alcatuita majoritar din pixeli de culoare neagra, iar domino-urile pot sa fie separate, acestea fiind albe. In primul rand, am verificat daca o celula este candidat pentru a fi ocupata de un domino folosind suma pixelilor. Cum pixelii albi au valori mari, am trunchiat suma la o valoare impartita la  $10^4$ .

O problema in aceasta abordare este reprezentata de celulele din centrul tablei. In special celula de pe pozitia 7I(cea pe care scrie 'le') contine suficienti pixeli albi astfel incat sa fie confundata cu un domino. Pentru a evita acest lucru am implementat un double check, adaugand si o verificare pe celula din imaginea color, fiind mai usor diferentiabila. Ajuta si faptul ca nuanta de pe tabla este mai inchisa ca cea de pe domino. Pentru a face mai multe verificari si teste, am salvat suma pixelilor intr-un dictionar, folosit ulterior si pentru verificare. Pentru celulele deja identificate ca domino am folosit un set. In codul de mai jos se pot observa valorile de prag pentru o celula candidat si verificarile pentru celula 7I:

Listing 4: Identifica pozitie

```

1 # alegem celula curenta de pe tabla
2 domino = warped_bin[j:j + 109, i:i + 109]
3 # calculam suma pixelilor de pe pozitia curenta
4 sum_domino = np.sum(domino)
5 dict_val[(linii[l_ind] + 1, coloane[c_ind])] = sum_domino
6 # daca depaseste aceasta valoare este posibil sa fie un

```

```

domino
7 if sum_domino > 1600000:
8     # pentru aceasta pozitie tabla este destul de alba incat
      sa fie confundata cu o piesa
9     if (linii[l_ind] + 1, coloane[c_ind]) == (7, 'I'):
10         colored_patch = warpedf[j:j + 109, i:i + 109]
11         possible_covered = np.sum(colored_patch[:, :, :], 1)
12         if np.sum(possible_covered) < 6000000 and 160 <=
            dict_val[(7, 'I')] // 10000 <= 187:
13             continue
14         if (linii[l_ind] + 1, coloane[c_ind]) in seen:
15             continue

```

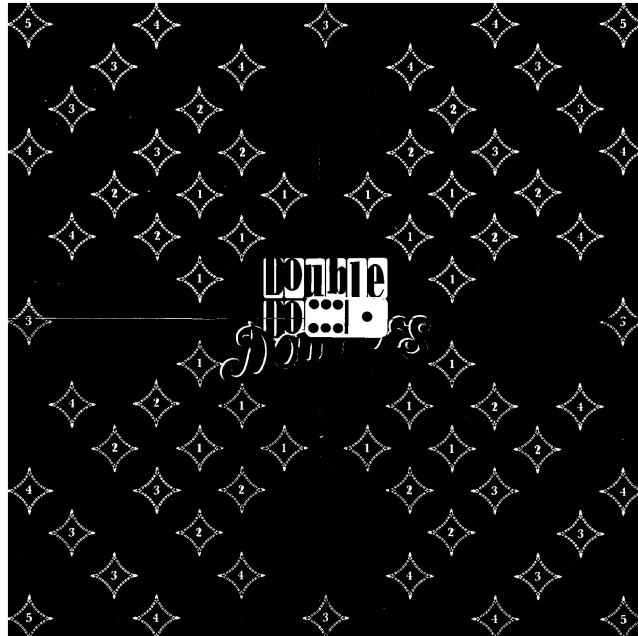


Figure 4: Imaginea binara

### 3. Recunoasterea numerelor

Pentru a recunoaste numerele de pe piese am folosit template matching. Pentru a crea template-urile am folosit imaginile auxiliare din care am extras, folosind codul de mai sus, jumatatile de domino. Am salvat aceste imagini intr-un folder si am folosit o functie pentru a identifica numarul de cercuri de pe fiecare jumătate de domino. Singura preprocesare folosita aici a fost un blur de medie.

```
img_template = cv.medianBlur(img_template, 5)
```

```
circles = cv.HoughCircles(img_template, cv.HOUGH_GRADIENT, 1, 20, param1=300,
param2=28, minRadius=8, maxRadius=100)
```

Atat rezultatele functiei care identifica numarul de cercuri cat si imaginile in sine au fost verificate manual si corectate eventualele greseli. In ceea ce priveste numarul de cercuri, a fost serializat și salvat un dictionar continand numerele corectate. In ceea ce priveste imaginile, au fost eliminate cele care nu sunt complete sau sunt inclinate.



Figure 5: Exemplu de template eliminat

Pentru template matching am mai adaugat un padding de 5 pixeli in fiecare directie pentru un domino/patch. Astfel, daca un domino nu este identificat complet intr-un patch, template matching-ul tot o sa functioneze. Pe imaginea obtinuta se aplica mai multe filtre precum blur gaussian, blur de medie, inchidere. Valorile, forma si dimensiunea pentru kernel si numarul de aplicari se regasesc in codul urmator:

Listing 5: Procesare patch

```
1 # aplicam preprocesare pe jumatatea de domino extrasa
2 imagine_cifra = cv.GaussianBlur(imagine_cifra, (11, 11),
   100)
3 imagine_cifra = cv.medianBlur(imagine_cifra, 5)
4 imagine_cifra = cv.normalize(imagine_cifra, None, alpha=0,
   beta=255, norm_type=cv.NORM_MINMAX)
5 kernel = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
6 imagine_cifra = cv.morphologyEx(imagine_cifra,
   cv.MORPH_CLOSE, kernel, iterations=3)
7 # aplica_template returneaza fisierul cu template-ul
   potrivit, conform acestuia alegem numarul corespunzator
   de pe domino
8 cifra_identificata =
   dict_nr_circles[aplica_template(imagine_cifra)]
```

Acest cod se afla in program in continuarea celui de la Listing 4. O procesare similara se aplica si pe imaginile folosite ca template, alaturi de un resize pe mai multe valori:

Listing 6: Template

```

1 scales = [0.88, 0.92, 0.95, 0.97, 1]
2 for scale in scales:
3     img_res = cv.resize(img_template.copy(), (0, 0),
4         fx=scale, fy=scale)
5     # aplica procesare pe template
6     img_res = cv.GaussianBlur(img_res, (11, 11), 100)
7     img_res = cv.normalize(img_res, None, alpha=0,
8         beta=255, norm_type=cv.NORM_MINMAX)
9     kernel = cv.getStructuringElement(cv.MORPH_RECT,
10         (3, 3))
11     img_res = cv.morphologyEx(img_res, cv.MORPH_CLOSE,
12         kernel, iterations=3)
13     img_res = cv.morphologyEx(img_res, cv.MORPH_OPEN,
14         kernel, iterations=3)

```

Confuzia in cazul acesta se facea intre domino-urile cu 4 si 6. Cele cu 6 fiind identificate ca fiind cu 4. Prin aplicarea acestor filtre culoarea devine mai difuza, iar 6 avand mai mult negru si gri devine suficient de diferit de 4. Metoda folosita pentru template matching este *TM\_CCOEFF\_NORMED*.

```
corr = cv.matchTemplate(domi, img_res, cv.TM_CCOEFF_NORMED)
```

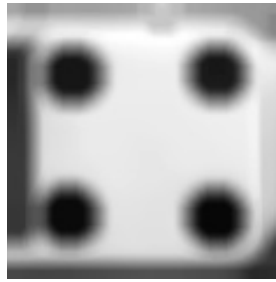


Figure 6: Patch dupa procesare

#### 4. Calcularea scorului

Pentru a calcula scorul am hardcodat o lista cu traseul de pe marginea tablei. De asemenea am hardcodat romburile de pe jumatatea de sus a tablei, celelalte fiind usor de obtinut deoarece sunt simetrice. Este retinuta pozitia fiecarui jucator, initial fiind 0.

Enumerand rezultatele din fisierele create la cerintele anterioare, verificam daca pozitia unui jucator de pe traseu coincide cu un domino plasat in runda curenta (indiferent de ce jucator face mutarea). Daca da, mutam jucatorul cu trei pozitii mai in fata. Daca jucatorul curent este unul dintre cei care primesc bonus, adaugam 3 puncte la scorul final al runde. Retinem acest lucru intr-o variabila pentru a putea verifica ulterior (1 daca jucatorul a primit bonus in decursul runde)



Listing 7: Calculeaza scor

```

1 bonus_j1 = 0
2 bonus_j2 = 0
3 if first_half[1] == str(configuratie_laterala[pozitie_j1]):
4     pozitie_j1 += 3
5     bonus_j1 = 1
6 elif second_half[1] ==
7     str(configuratie_laterala[pozitie_j1]):
8     pozitie_j1 += 3
9     bonus_j1 = 1
10 if first_half[1] == str(configuratie_laterala[pozitie_j2]):
11     pozitie_j2 += 3
12     bonus_j2 = 1
13 elif second_half[1] ==
14     str(configuratie_laterala[pozitie_j2]):
15     pozitie_j2 += 3
16     bonus_j2 = 1

```

Daca avem un domino plasat pe un romb, verificam daca este un domino dublu. Corespunzator adunam la scorul rundeii numarul de puncte de pe romb, sau dublul acestuia.

Listing 8: Calculeaza scor

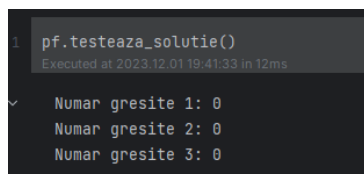
```

1 f = open(os.path.join(rezultate_dir, fisier), 'r')
2 first_half = f.readline().split()
3 second_half = f.readline().split()
4 if first_half[1] == second_half[1]:
5     nr_puncte_runda += (dictionar_pct[first_half[0]] +
6     dictionar_pct[second_half[0]]) * 2
7 else:
8     nr_puncte_runda += dictionar_pct[first_half[0]] +
9     dictionar_pct[second_half[0]]

```

Adunam si la pozitia jucatorului acelasi numar ca cel de puncte. Pentru ca avem mai multe jocuri si stim ca fiecare are 20 de runde, folosim indicele din enumerare pentru a reseta pozitia jucatorilor.

**5. Rezultate** Programul nu produce greseli pe datele de antrenare nemodificate.



```

1 pf.testeaza_solutie()
   Executed at 2023.12.01 19:41:33 in 12ms
   Numar gresite 1: 0
   Numar gresite 2: 0
   Numar gresite 3: 0

```

Figure 7: Rezultate

În schimb la o ajustare a luminozității în hsv cu  $\pm 0.3$ , acesta face o greșeală.

```
pf.testeaza_solutie()
Executed at 2023.12.02 01:04:39 in 56ms

Gresit!! C2: 3_08.txt
False
Numar gresite 1: 0
Numar gresite 2: 1
Numar gresite 3: 2
```

Figure 8: Rezultate

La schimbări mai mari de  $\pm 0.3$  programul produce multiple greșeli.

## 2 Hardcoded

Listing 9: Puncte romburi

```
1 # Prima linie
2     dictionar_puncte[(1, 'A')] = 5
3     dictionar_puncte[(1, 'D')] = 4
4     dictionar_puncte[(1, 'H')] = 3
5     dictionar_puncte[(1, 'L')] = 4
6     dictionar_puncte[(1, 'O')] = 5
7     # A doua linie
8     dictionar_puncte[(2, 'C')] = 3
9     dictionar_puncte[(2, 'F')] = 4
10    dictionar_puncte[(2, 'J')] = 4
11    dictionar_puncte[(2, 'M')] = 3
12    # A treia linie
13    dictionar_puncte[(3, 'B')] = 3
14    dictionar_puncte[(3, 'E')] = 2
15    dictionar_puncte[(3, 'K')] = 2
16    dictionar_puncte[(3, 'N')] = 3
17    # A patra linie
18    dictionar_puncte[(4, 'A')] = 4
19    dictionar_puncte[(4, 'D')] = 3
20    dictionar_puncte[(4, 'F')] = 2
21    dictionar_puncte[(4, 'J')] = 2
22    dictionar_puncte[(4, 'L')] = 3
23    dictionar_puncte[(4, 'O')] = 4
24    # A cincea linie
25    dictionar_puncte[(5, 'C')] = 2
26    dictionar_puncte[(5, 'E')] = 1
27    dictionar_puncte[(5, 'G')] = 1
28    dictionar_puncte[(5, 'I')] = 1
```

```

29     dictionar_puncte[(5, 'K')] = 1
30     dictionar_puncte[(5, 'M')] = 2
31     # A sasea linie
32     dictionar_puncte[(6, 'B')] = 4
33     dictionar_puncte[(6, 'D')] = 2
34     dictionar_puncte[(6, 'F')] = 1
35     dictionar_puncte[(6, 'J')] = 1
36     dictionar_puncte[(6, 'L')] = 2
37     dictionar_puncte[(6, 'N')] = 4
38     # A saptea linie
39     dictionar_puncte[(7, 'E')] = 1
40     dictionar_puncte[(7, 'K')] = 1
41     # A opta linie
42     dictionar_puncte[(8, 'A')] = 3
43     dictionar_puncte[(8, 'O')] = 3

```

Listing 10: Traseu lateral

```

1     configuratie_laterală = [-1, 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]

```

Listing 11: Linii/coloane

```

1     linii = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
        15]
2     coloane = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
        'J', 'K', 'L', 'M', 'N', 'O']

```

### 3 Bibliografie

1. Cod laborator
2. Documentatie Opencv