

Descrierea soluției pentru jocul Double Double Dominoes

Udrea Iulia-Maria

Grupa 463

1 Introducere

Documentația de față oferă o explicație a implementării algoritmului pentru jocul Double Double Dominoes utilizând tehnici de Computer Vision în Python, cu ajutorul bibliotecilor OpenCV și NumPy. Scopul acestui proiect este de a dezvolta o soluție pentru identificarea poziției pieselor de domino, detectarea tipului acestora și calcularea scorului corespunzător fiecărei mutări în cadrul jocului.

2 Task I - Detectarea poziției piesei

Sunt necesari doi pași premergători, care vor fi utilizați în rezolvarea tuturor cerințelor: extragerea chenarului și prelucrarea imaginii pentru a scoate în evidență piesele de domino.

2.1 Extragerea careului

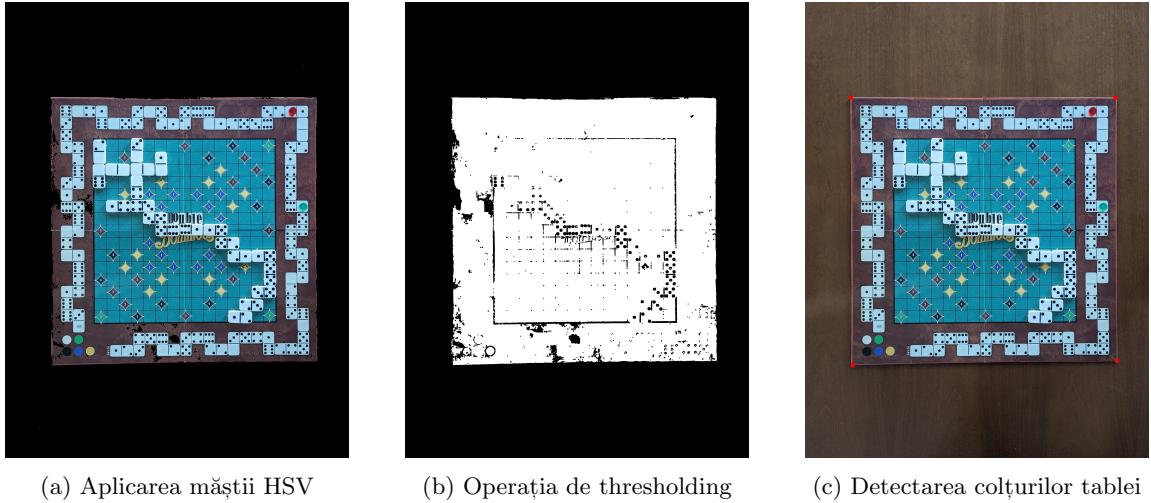
Prima etapă a soluției constă în izolarea chenarul de joc și eliminarea elementelor nedorite, cum ar fi masa și marginile tablei. În acest scop, am aplicat o mască HSV, astfel încât pixelii din intervalul specificat rămân neschimbați, în timp ce restul devin negri. Se evidențiază doar zona tablei de joc și se elimină informațiile referitoare la masă, așa cum se poate observa în Figura 1a.

Procesul de eroziune, aplicat pe imaginea grayscale rezultată, asigură eliminarea oricărora pixeli albi rămași în dreptul mesei, pentru a evita ca ulterior acestia să fie considerați colțurile tablei. Prin aplicarea unei operații de thresholding cu pragul 10, obținem o imagine binară în care tabla de joc este predominant albă, în timp ce restul pixelilor sunt negri (Figura 1b).

Listing 1: Prelucrarea imaginilor pentru determinarea colțurilor tablei

```
1 frame_hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
2 l = np.array([25, 0, 0])
3 u = np.array([255, 255, 255])
4 mask_table_hsv = cv.inRange(frame_hsv, l, u)
5 res = cv.bitwise_and(frame, frame, mask=mask_table_hsv)
6
7 res = cv.cvtColor(res, cv.COLOR_BGR2GRAY)
8 kernel = np.ones((7, 7), np.uint8)
9 res = cv.erode(res, kernel)
10
11 _, thresh = cv.threshold(res, 10, 255, cv.THRESH_BINARY)
```

Am preluat apoi codul din laborator, care utilizează *findContours* și caută colțurile tablei în funcție de aria maximă găsită, colțuri care vor fi folosite pentru a extrage o imagine cu tabla printr-o transformare de perspectivă (Figura 2a).



(a) Aplicarea măștii HSV

(b) Operația de thresholding

(c) Detectarea colțurilor tablei

Figura 1: Pași intermediari în extragerea careului de joc

Cu toate acestea, marginea tablei, care cuprinde traseul de scor, nu contribuie la rezolvarea cerințelor. Prin urmare, această margine este eliminată cu ajutorul unor coordonate fixe, iar imaginea rezultată (Figura 2b) este redimensionată la 1500x1500 pixeli.



(a) Transformarea perspectivă

(b) Eliminarea marginilor tablei

Figura 2: Extragerea chenarului de joc

2.2 Prelucrarea imaginilor

După ce chenarul de joc a fost izolat, imaginile rezultate păstrează culorile inițiale. Acestea sunt supuse unei noi etape de prelucrare.

Se folosește o nouă mască HSV cu scopul de a suprima culorile predominante ale tablei și a scoate în evidență piesele de domino (Figura 3a), după care se aplică un threshold cu pragul 140. Se obține în final o imagine binară (Figura 3b), care facilitează procesul ulterior de detectare a poziției pieselor.

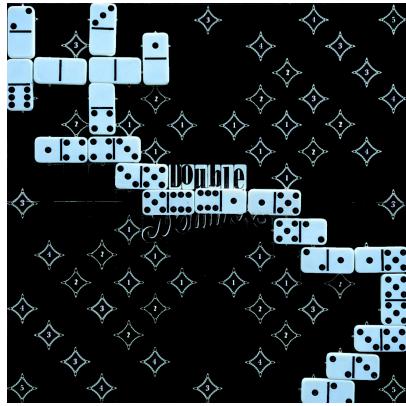
Listing 2: Prelucrarea imaginilor pentru evidențierea pieselor

```

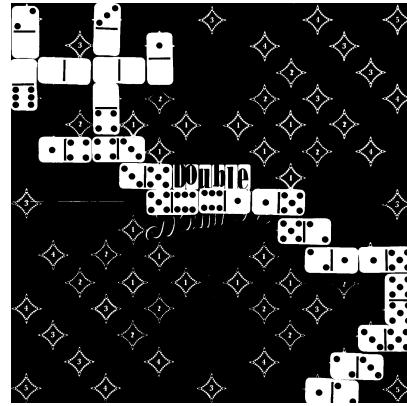
1 frame_hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
2 l = np.array([75, 0, 0])
3 u = np.array([120, 120, 255])
4 mask_table_hsv = cv.inRange(frame_hsv, l, u)
5 res = cv.bitwise_and(frame, frame, mask=mask_table_hsv)
6
7 res = cv.cvtColor(res, cv.COLOR_BGR2GRAY)
8 _, thresh = cv.threshold(res, 140, 255, cv.THRESH_BINARY)

```

Toate imaginile de testare, dar și imaginea de start (care nu conține nicio piesă), sunt transformate utilizând cele două operații descrise în subsecțiunile 2.1 și 2.2.



(a) Aplicarea măștii HSV



(b) Operația de thresholding

Figura 3: Pași în prelucrarea imaginilor

2.3 Detectarea poziției piesei adăugate

Pentru a detecta poziția piesei plasate la o anumită rundă se va calcula diferența dintre imaginea curentă și cea anterioară, iar în cazul primei mutări din fiecare set se va folosi imaginea de start. Imaginele sunt binare (de tipul *uint8*), ceea ce înseamnă că în matricea diferenței nu vor exista valori negative, aceasta fiind de asemenea o imagine binară (Figura 4).

Deoarece toate imaginile au dimensiunea standard de 1500x1500 de pixeli, iar chenarul este alcătuit din 15x15 patrățele, fiecare patrățel este poziționat la coordonate fixe, având latura de 100 de pixeli.

Algoritmul parcurge matricea diferenței în secțiuni de dimensiunea patrățelor și identifică două pozitii cu cele mai mari sume ale diferențelor. Acestea reprezintă coordonatele (pozițiile) unde s-a adăugat piesă.

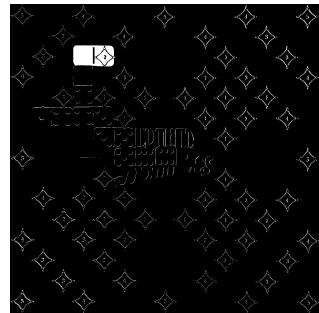


Figura 4: Exemplu de matrice diferență

3 Task II - Recunoașterea numerelor de pe piesă

Numerele de pe piesă adăugată vor fi detectate pentru fiecare jumătate a acesteia, adică se vor decupa individual pătrătelele detectate anterior. Deoarece piesele nu sunt întotdeauna așezate perfect pe tablă, am adăugat 5 pixeli la toate laturile posibile ale pătratului, acolo unde nu se depășește marginea imaginii întregi.

Numerele de pe piesă sunt reprezentate sub formă de buline care au aproximativ aceeași formă și dimensiune, indiferent de orientarea piesei (verticală, orizontală, rotită etc.). Pentru a număra bulinile, am utilizat metoda *HoughCircles*, care identifică pozițiile cercurilor din imagine.

Fiind o imagine binară, aceasta este destul de granulată, având contururi abrupte. De aceea am aplicat în prealabil un filtru median pentru a reduce asperitățile și pentru a obține o imagine mai uniformă, urmată de un detector Canny care evidențiază contururile acum rotunjite.

Listing 3: Parametrii funcției HoughCircles

```
1 img = cv.medianBlur(img, ksize=5)
2 img = cv.Canny(img, 50, 120)
3
4 circles = cv.HoughCircles(
5         img, cv.HOUGH_GRADIENT,
6         dp=1.7, minDist=7,
7         param1=75, param2=30,
8         minRadius=8, maxRadius=14)
```

Parametrii funcției au fost aleși ținând cont de următoarele aspecte:

- HoughCircles poate detecta mai multe cercuri în dreptul aceleiași buline. Distanța minimă dintre două cercuri detectate trebuie să fie suficient de mare pentru a evita acest lucru.
- Raza bulinelor (cercurilor) căutate este în general aceeași. Totuși, creșterea razei maxime asigură faptul că acestea sunt detectate chiar și în cazul în care piesa nu este vizibilă complet în pătratul decupat.
- Raza maximă trebuie să fie suficient de mică încât să nu fie detectate marginile rotunjite ale piesei intregi ca fiind cercuri.

Ceilalți parametri au fost determinați experimental.

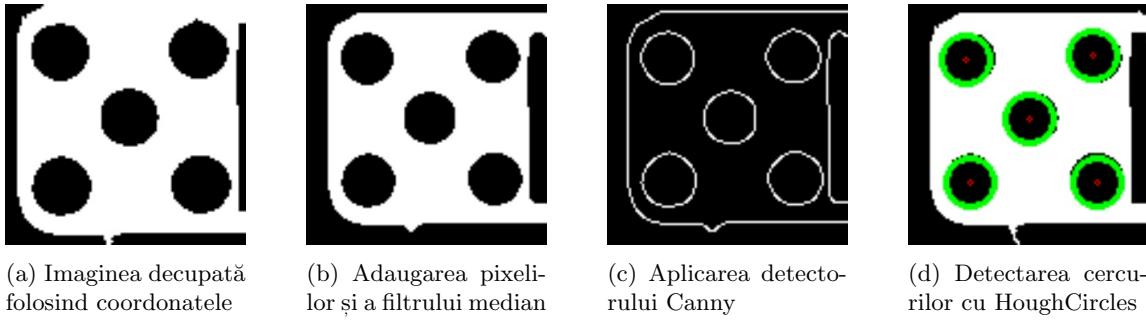


Figura 5: Pașii în recunoașterea numerelor de pe piesă

4 Task III - Calcularea scorului

Jucătorul curent este identificat din fișierele de mutări. Pentru a determina scorul mutării, se adaugă punctajele asociate fiecărei poziții de pe tabla de joc unde a fost plasată piesa, ținându-se cont dacă aceasta este o piesă dublă, caz în care scorul se dublează. Punctajele de pe tabla de joc au fost predefinite manual într-o matrice (*scoruri_tabla*)

Listing 4: Calcularea scorului folosind datele extrase la pașii anteriori

```
1 jucator_curent = int(mutari[j][-1]) - 1
2 jucator_opus = 1 - jucator_curent
3
4 # Se adauga scorul mutarii
5 scor_mutare = scoruri_tabla[coord1[0]][coord1[1]] +
6             scoruri_tabla[coord2[0]][coord2[1]]
7
8 # Se verifica daca e piesa dubla, caz in care scorul se dubleaza
9 if buline[0] == buline[1]:
10     scor_mutare *= 2
```

Jocul implică și parcurgerea unui traseu cu piese de domino, care poate aduce un bonus oricărui jucător, nu doar celui curent. Scorul fiecărui jucător marchează și poziția acestuia pe traseu. Astfel, pentru a determina bonusurile, se verifică dacă oricare dintre numerele de pe piesa plasată coincid cu cele de pe traseu, la pozițiile corespunzătoare scorurilor.

```
1 if (scor[jucator_curent] > 0 and
2     traseu[scor[jucator_curent] - 1] in buline):
3     scor_mutare += 3
4
5 if (scor[jucator_opus] > 0 and
6     traseu[scor[jucator_opus] - 1] in buline):
7     scor[jucator_opus] += 3
8
9 scor[jucator_curent] += scor_mutare
```

Rezultatele obținute la aceste 3 taskuri se scriu în fișiere de output .txt, respectând formatul impus.