

Calcularea scorului în jocul Mathable folosind Vedere Artificială

Alexandru Miclea

Facultatea de Matematică și Informatică

An universitar 2024-2025

Grupa 331

30 noiembrie 2024

Cuprins

1 Introducere	3
1.1 Descrierea proiectului	3
1.2 Structurarea proiectului	3
2 Implementare	3
2.1 Extragerea careului dintr-o poză	3
2.2 Extragerea şabloanelor cu piese de joc	5
2.3 Extragerea configuraţiei unui joc	5
2.4 Extragerea coordonatelor plasării unei piese	7
2.5 Calculul punctelor obţinute într-o tură	8
2.6 Calculul punctelor obţinute de un jucător	8
3 Discuţie asupra optimalităţii soluţiei	9
4 Concluzie	9

1 Introducere

1.1 Descrierea proiectului

Acest proiect a fost realizat pentru disciplina "Concepțe și Aplicații în Vederea Artificială". Tema proiectului este analiza mai multor jocuri Mathable. Jocul se presupune a fi jucat de către 2 persoane, aceștia plasând un număr de piese pe tabla de joc.

Informațiile care se doresc a fi extrase sunt următoarele:

- Poziția plasată de către un jucător în timpul unei ture
- Numărul de pe piesa plasată de către un jucător în timpul unei ture
- Scorul total primit de un jucător într-o tură

1.2 Structurarea proiectului

După citirea cerinței, am hotărât să structurez codul în două clase Python:

- **GameState**: Clasa care stochează piesa și poziția aferente unei mutări produse în joc, metode de extras poziția unde a fost plasată o piesă și de calcul a punctelor pe care mutarea a adus-o
- **Game**: Clasa care stochează o listă cu toate stările de joc, metode de prelucrat imaginile oferite, scrierea fisierelor .txt
- **Constants**: Două metode care întorc o matrice de dimensiune 14*14 cu numerele pieselor pe tablă, respectiv regulile care se află pe acea poziție
- **Main**: Locul în care definesc căile către directoarele de train/test, alături de numărul de jocuri de procesat și folder-ul de şabloane cu piese și tabla de joc goală.

2 Implementare

2.1 Extragerea careului dintr-o poză

Careul este porțiunea din tabla de joc în care se plasează piesele. Extrag doar această porțiune pentru a minimiza porțiunile din imagine în care poate apărea zgomot (masă, porțiunea albastră a tablei etc.).

Procesul de extragere îmbină mai multe procedee clasice de Computer Vision. Pașii aplicati sunt următorii:

- Iau imaginea aşa cum este ea în directorul de train/test
- Duc imaginea din BGR în HSV și adun 22 la valorile aferente Hue (modulo 180)
- Duc imaginea din HSV în Grayscale
- Aplic un filtru de dimensiune 3*3 cu medie 1 peste imaginea de la pasul anterior (astfel mai "curăț" din muchiile pe care urmeaza sa le obtin la pasul următor)

- Fac un threshold pentru pixelii care au valoarea <165, astfel incăt aceştia să aibă valoarea 0 (negru)
- Aplic un filtru de dilatare de dimensiune 15*15 peste imagine. Aici careul poate fi uşor detectat de metoda `cv.findContours()`

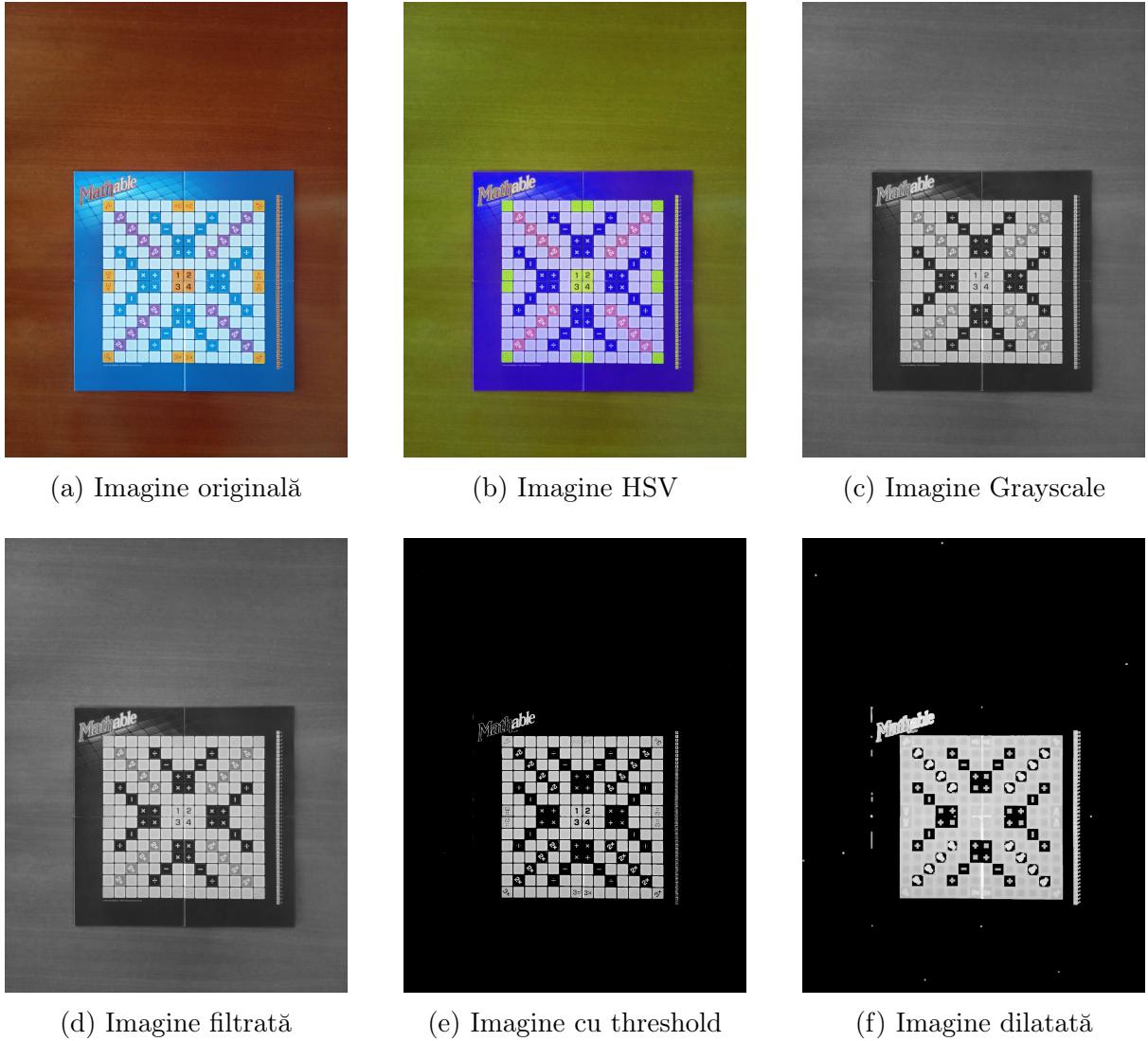


Figura 1: Ilustrarea paşilor pentru extragerea careului

Intuiţia pe care am avut-o în abordarea prezentată mai sus pornea de la faptul că formula de convertire a unei imagini din BGR în Grayscale este următoarea:

$$\text{Grayscale} = 0.299 * R + 0.587 * G + 0.114 * B \quad (1)$$

Aşadar, dacă reușeam să fac tabla minus careu să fie un albastru "pur" iar portiunea maro a mesei să fie suficient de mică, acestea ar fi fost mai ușor eliminate de către pasul de thresholding, ceea ce s-a și întâmplat.

În urma extragerii contururilor folosind codul prezentat la laborator am observat pentru mai multe imagini că partea stânga sus poate fi luată în afara careului, distorsionând piesele. Un workaround a fost să iau coordonatele colțului stânga sus raportat la celelalte 3 colțuri (care erau identificate corect mai consistent). Totodată, experimental, am

observat că rezoluția finală a careului este 1460*1460 pixeli. Pentru a avea rezoluția maximă și a putea împărți careul în 14*14 zone, am ales ca lățimea și lungimea să fie de 1456*1456 ($1456 / 14 = 104$).

În final, careul pe care acest pipeline îl extrage arată ca în figura de mai jos:

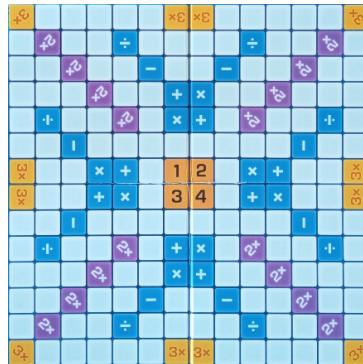
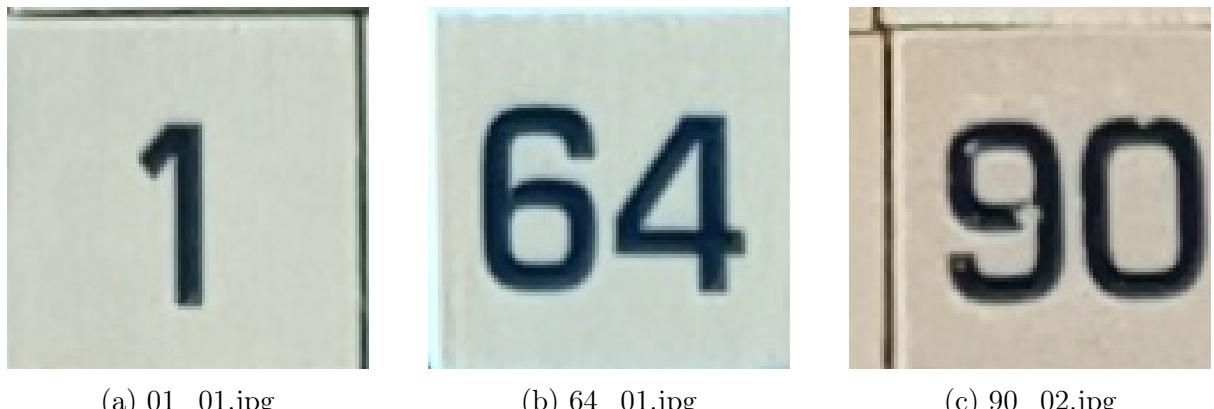


Figura 2: Careul final

2.2 Extragerea șabloanelor cu piese de joc

Folosind pipeline-ul de mai sus am salvat toate careurile din turele pe care le-am avut în directorul train, alături de 2 imagini extra care conțineau toate piesele de joc. Am împărțit careurile în secțiuni de 104*104 și am salvat acele șabloane. Ele au fost label-uite sub forma {număr_piesă}_{număr_apariție}.jpg.



(a) 01_01.jpg

(b) 64_01.jpg

(c) 90_02.jpg

Figura 3: Exemple de șabloane cu piese extrase

Aplicarea șabloanelor va fi descrisă în următoarea secțiune.

2.3 Extragerea configurației unui joc

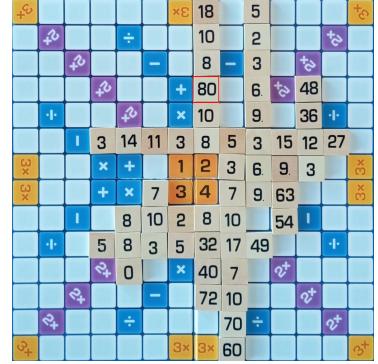
Prin configurație mă refer la poziția plasării pieselor pe durata unui joc. Deoarece acestea nu își pot schimba poziția sau să dispară, am dedus că este suficient să fac extragerea din ultima poză aferentă unui joc.

Sistemul de extragere este inspirat din alegerile care au loc în perioada redactării documentației acesteia. Ceea ce fac în mod precis este să fac Template Matching

folosindu-mă de sabloanele extrase (292 la număr) peste careul turei 50 și să găsesc pozițiile din imagine în care matching-ul are valoarea $\geq 90\%$. Pentru cifra 7 în mod specific am redus threshold-ul la 80%.



(a) Matching pentru 3



(b) Matching pentru 90

Figura 4: Ilustrarea procesului de Template Matching

Am ales aceste 2 poze pentru a ilustra următoarele aspecte:

- În cazul piesei 3, mai exact pentru piesa din mijloc, algoritmul a detectat de mai multe ori piesa ca fiind 3. Acesta este un lucru bun, pentru că orice vot contează :)
- În cazul piesei 90, aceasta a detectat în mod fals 80 ca fiind o piesă 90. Am ținut să minimizez posibilitatea ca acest lucru să se întâpte, drept urmare am încercat mai multe tipuri de Template-Matching. Cel mai bun rezultat a fost obținut de `cv.TM_CCOEFF_NORMED`.

Date fiind condițiile de mai sus, abordarea mea a fost următoarea:

- Preiau imaginea careului (ținută în memoria programului) și o transfer în Grayscale. La fel procedez și pentru fiecare şablon.
- Iterez prin fiecare şablon și aplic `cv.matchTemplate()`
- Aplic threshold-urile menționate mai sus
- Punctele rămase îmi indică colțul stânga sus unde template-ul se potrivește. Dacă aceste coordonate din 1456×1456 în 14×14 după următoarea formulă:

$$poziție = \max((punct_identificat + 52), 0) // 104 \quad (2)$$

- În fiecare astfel de bucket am o listă de piese cu care s-a făcut Template Matching. Ca să aleg piesa cu cele mai multe match-uri aplic `np.bincount(nparray).argmax()`. Dacă nu există elemente în bucket-ul respectiv, spun că acea pătrătică nu conține o piesă.

Având astfel configurația finală, pot deduce, dată poziția plasării unei piese, ce piesă a fost pusă și la ce tură

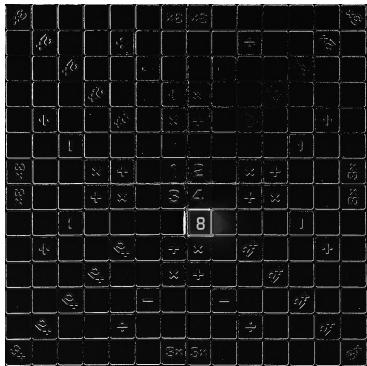
2.4 Extragerea coordonatelor plasării unei piese

Pentru a înțelege modul în care se detectează poziția unde a fost mutată o piesă sunt necesare câteva precizări:

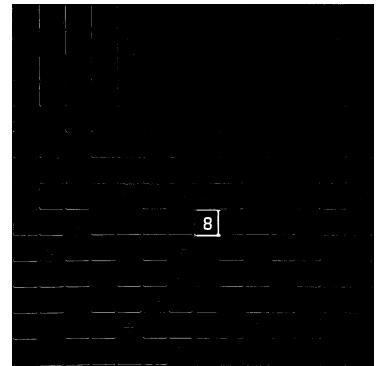
- Aplic metoda Sliding Window. Fereastra este de dimensiune 124*124 (104px cât are şablonul + 10px padding în fiecare direcție)
- Diferența dintre careuri reprezintă diferența în modul aplicată careului turei actuale și celei precedente. Dacă sunt la tură 1, tură precedentă este tabla fără piese pe ea

Procesul prin care detectez unde a fost pusă o piesă este următorul:

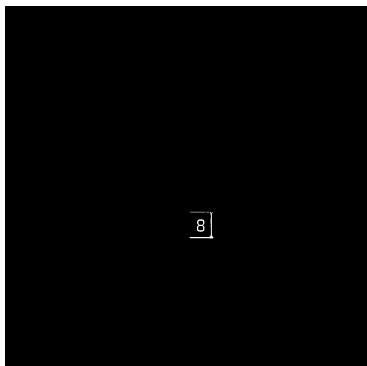
- Fac diferența dintre careuri
- Aplic un threshold în felul următor:
 - Valorile <100 și >220 sunt duse în valoarea 0
 - Valorile != 0 sunt duse în 255
- Aplic un filtru de eroziune de dimensiune 3*3 (suficient de mare cât să șteargă zgomotul reprezentat de liniile careului și suficient de mic cât să nu șteargă diferența unde se află piesa nou plasată)
- Plimb Sliding Window peste careu și fac o medie a pixelilor din ferestra. Fereastra cu media cea mai mare este returnată ca variantă de răspuns.



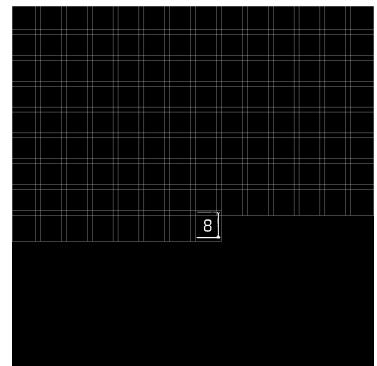
(a) Diferență Grayscale



(b) Aplicare threshold



(c) Filtru eroziune



(d) Sliding Window

Figura 5: Ilustrarea pașilor pentru extragerea coordonatelor pieselor

2.5 Calculul punctelor obținute într-o tură

Pentru a calcula câte puncte obțin într-o tură am următorul set de pași:

- Dată fiind o stare a jocului, mă uit să văd unde și ce piesă am plasat
- Determin multiplicatorul dat de căte expresii valide completez
 - Verific pentru fiecare direcție dacă am cel puțin două piese cu care pot completa o expresie
 - Pentru o astfel de direcție verific dacă am o condiție de constrângere a operației. Dacă da, atunci verific doar acea operație. Dacă nu, verific oricare operație până găsesc o operație validă.
 - Întorc 1 pentru operație validă, respectiv 0 altfel.
- Înmulțesc numărul de pe piesă cu acel multiplicator.
- Verific dacă poziția pe care am plasat piesa are multiplicator. Dacă da, înmultesc numărul de pe piesă cu acel multiplicator.

2.6 Calculul punctelor obținute de un jucător

După ce aplic algoritmul din secțiunea precedentă fiecărei ture, citesc fișierul de ture și însumez valorile obținute pentru fiecare tură.

3 Discuție asupra optimalității soluției

- Extragerea poziției se face foarte bine din cât am testat.
- Deducerea numărului unei piese poate să furnizeze un rezultat gresit / nici un rezultat. În cazul acesta voi pierde 0.35p per piesă clasificată greșit.
- Calculul punctajului a fost făcut corect. Am ținut cont de toate edge case-urile.
- Pe testul din directorul `fake_test` am obținut punctajul maxim.

4 Concluzie

Proiectul acesta a fost foarte drăguț. Mi-a plăcut faptul că l-am abordat, după părerea mea, într-un mod original. De asemenea mi-a plăcut foarte mult să experimentez și să găsesc o soluție eficientă.