Vlaicu Ana-Francesca
Gr. 463

Calculator automat de scor pentru jocul Mathable

Documentație

Introducere

Programul prezentat are scopul de a analiza și procesa imagini ce conțin o tabla de Mathable. Utilizând tehnici de prelucrare a imaginii, acesta extrage careul de joc dintr-o imagine, detectează pozițiile celulelor și determină configurația lor (ocupat/neocupat) într-un format matriceal. Apoi, după găsirea piesei adăugate în ultima rundă, programul identifică numărul de pe ea.

Surse

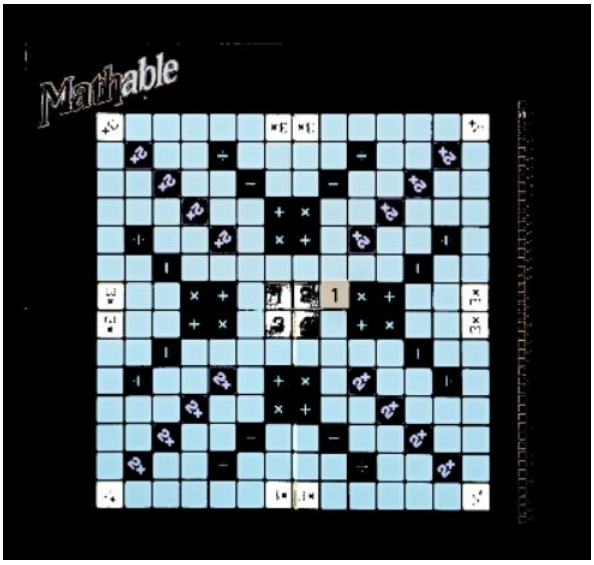
Codul de pornire a fost cel din laboratorul 6, pentru jocul de Sudoku. Din el am folosit logica pentru extragerea careului și determinarea configurației careului în formă matriceală. Pentru identificarea valorilor HSV, am folosit codul din laboratorul 7, cel cu filtrarea culorilor.

Materiale auxiliare

Pentru task-ul 2 am creat un folder *templates1* ce conține piesele posibile de joc. Numele fiecărei poze este chiar numărul din poză pentru a le prelucra și accesa facil.

Funcții pentru realizarea task-ului 1

Funcția principală pentru extragerea spațiului de joc din imaginile furnizate este *extract_gameboard*, ce primește o imagine la intrare și returneaza zona de interes. Înainte de a o procesa, asupra imaginii aplic un filtru prin funcția *hsv_filter_gameboard*, care îmi convertește zonele galbene în alb, iar apoi izolează regiunile deschise la culoare pentru a păstra conținutul de interes și a elimina fundalul. De asemene, păstrez o copie a imaginii nefiltrate.



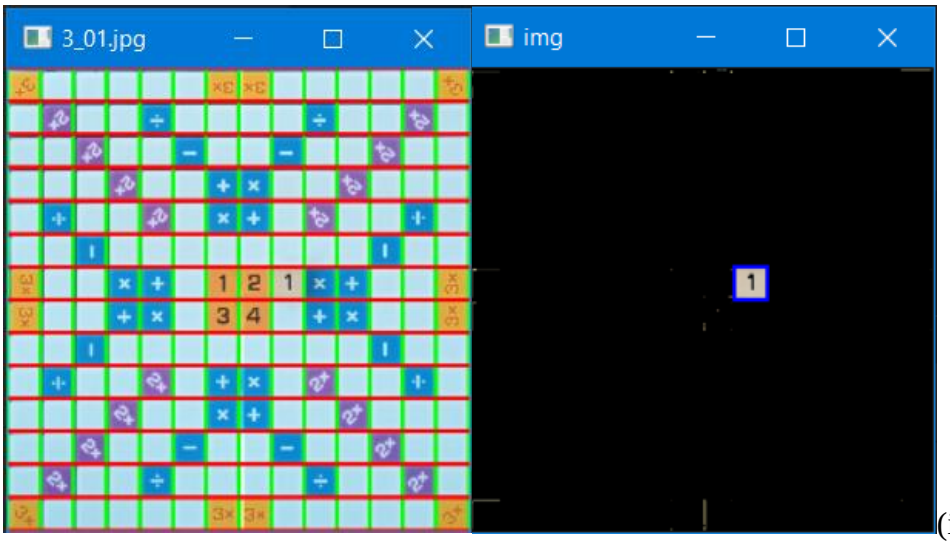
(imaginea după aplicarea filtrului hsv)

În continuare, convertesc imaginea în grayscale și aplic asupra ei un blur median, unul gaussian si un sharpen pentru a reduce zgomotul și a accentua detaliile. Imaginea este convertită în alb-negru, folosind un threshold adaptiv, și punctele mici de zgomot alb sunt reduse prin eroziune. Folosesc detecția Canny pentru a evidenția muchiile tablei de joc. Caut contururile din imagine folosind funcția cv.findContours. și păstrez doar contururile exterioare, iar contururile mici (sub 4 puncte) sunt ignorate. Pentru fiecare contur, identific cele patru colțuri ale careului (stânga-sus, dreapta-sus, dreapta-jos, stânga-jos) pe baza coordonatelor punctelor conturului. (de exeplu, colțul stânga-sus este determinat de suma minimă a coordonatelor (x + y)). Apoi, calculez aria conturului format din colțurile detectate și o compar cu aria maximă găsită. Dacă aria este mai mare, actualizez colțurile și aria maximă. În final, folosindu-mă de copia imaginii, păstrată la început, extrag careul de joc ce se află între cele patru colțuri găsite.

(colțurile detectate)

După detectarea careului, îmi definesc liniile orizontale și verticale pentru o tablă de 14x14 cu dimensiunea celulei de 114 pixeli (latura pătratului de 1600 de pixeli împărțită la 14 poziții posibile pentru așezarea piesei).

Pasul următor este analiza careului și determinarea configurației acestuia. Mai intâi, aplic asupra careului extras un alt filtru HSV ce izolează doar piesele de joc:



(izolarea pieselor)

Funcția *gameboard_ox_config* are rolul de a analiza imaginea filtrată ca mai sus și de a extrage configurația tablei de joc. Aceasta se folosește de liniile definite mai sus pentru a împărți imaginea în celule și a hotărî dacă celula conține o piesă de joc sau nu. După ce aplic un threshold asupra imaginii, intru în functie, unde creez inițial o matrice goală de 14x14 și încep să parcurg tabla, calculând coordonatele ce indică limitele fiecărei celule cu o marjă mică de eroare. Extrag zona corespunzătoare și calculez media și deviația standard a valorilor pixelilor din acest patch. Dacă media pixelilor este foarte mică (<30) și variabilitatea este scăzută (<15), celula este considerată complet neagră (fără piesă) și marcată cu **o**. Dacă zona nu este complet neagră, se analizează proporția de pixeli negri. Împart numărul de pixeli negri la dimensiunea patch-ului și, dacă imaginea mea nu este neagră, dar am o proporție mica de pixeli negri (conturul cifrei de pe piesă), marchez zona cu **x** pentru a semnala prezența piesei.

Configurațiile mele sunt salvate într-o listă și astfel, comparând matricea corespondentă imaginii curente cu cea anterioară, determin poziția unde a fost plasată piesa în runda curentă. Păstrez coordonatele piesei noi și, pentru scrierea în fișier, adun 65 la indicele coloanei și îl convertesc din int in char pentru a afișa poziția conform cerinței.

Întrucât prima imagine dată este cea cu prima mutare, altă funcție pe care o folosesc este *empty_board_config*, cu ajutorul căreia creez o matrice ce simbolizează tabla goală. Aceasta este utilă pentru a determina unde a fost amplasată prima piesă în joc.

Funcții pentru realizarea task-ului 2

După identificarea piesei așezate pe tablă în runda curentă și cu ajutorul coordonatelor găsite mai sus ale acesteia, extrag din careul de joc patch-ul ce o conține și apelez pentru ea funcția *categorize_tile*. Rolul acesteia este de a identifica numărul de pe piesa nou adăugată, însă și ea apelează mai multe funcții. La început, inițializez mai multe variabile, precum corelația maximă

găsită (pornind de la -inf), o variabilă ce reprezintă identificatorul șablonului care are cea mai mare corelație cu patch-ul (inițial -1) și o variabilă care reține șablonul care s-a potrivit cel mai bine. Aplic pe patch funcția *preprocess_image* ce include pași precum aplicarea unui filtru gaussian și conversia în tonuri de gri. În continuare, funcția iterează prin toate deplasările posibile pe verticală și orizontală ale *patch*-ului în intervalul specificat de variabila de intrare *shift_range*. Pentru fiecare pereche de deplasări, se aplică funcția *shift_patch*, ce are rolul de a muta patch-ul pe imaginea din care a fost extras. Apoi iterez prin toate șabloanele din folder-ul *templates*, și acestea fiind preprocesate cu *preprocess_image*. Scalez șablonul pentru a avea dimensiuni similare cu *patch-ul* utilizând funcția *scale_template*. Pentru fiecare șablon scalat, se testează diferite rotații în intervalul specificat (de exemplu, între -9 și +9 grade, cu pași de 3 grade). Aplic funcția *rotate_image* ce calculează centrul imaginii folosind coordonatele (width // 2, height // 2) și folosește o matrice de rotație pentru a realiza rotația în jurul centrului la un unghi dat.

Funcția *correlation_score* este utilizată pentru a calcula un scor de corelație între patch și șablonul rotit. Acest scor indică cât de bine se potrivesc cele două imagini. La intrare, *correlation_score* primește patch-ul și șablonul, ce trebuie să aibă aceeași dimensiune. După convertirea la float a datelor, calculez media pixelilor pentru fiecare imagine și deviația fiecărui pixel față de media imaginii respective. Calculez apoi numărătorul (adică produsul scalar între deviațiile pixelilor din template și patch), normele L2 (sumă pătratelor deviațiilor) pentru fiecare imagine și numitorul. Numitorul este produsul normelor celor două imagini sub radical. În final, scorul de corelație este dat de împărțirea numărătorului la numărător. O valoare apropiată de 1 înseamnă că imaginile sunt similare, iar o valoare apropiată de -1 arată contrariul. Cazul în care numitorul este 0 (una dintre imaginile de intrare are toți pixelii de aceeași valoare) este tratat, iar funcția întoarce 0.

Înapoi în funcția *categorize_tile*, dacă scorul de corelație este mai mare decât maxi (corelația maximă curentă), funcția actualizează noua valoare a corelației maxime, identificatorul șablonului curent și șablonul care a generat acest scor. La final, funcția returnează identificatorul (poz) al șablonului care s-a potrivit cel mai bine cu patch-ul. Acest identificator indică numărul piesei.

Alte observații

Logica de rulare se află în funcția run_project, întrucât întreaga rezolvare se află într-un singur fișier .py. Funcția run_project iterează prin datele de intrare, aplică funcțiile pe ele și crează fișierele de ieșire și directorul pentru ele (dacă nu există). Deși nu am implementat task-ul 3, am creat fișiere goale pentru turns si scores, deoarece am observat că script-ul de evaluare nu scrie punctajul total fără ele.