Documentație proiect jocul Mathable

Daria Pîrvulescu

Decembrie 2024

1 Scopul proiectului

Mathable este un joc ce se bazează pe formarea de ecuații valide de matematică (adunare, scădere, împărțire și înmulțire). Avem la dispoziție anumite numere de la 0 la 90 și o tablă de 14x14 casuțe care conține anumite constrângeri și bonusuri ca în figurile de mai jos:

Figura 1

Mathable se poate juca în mai mulți, dar pentru acest proiect s-a ales varianta cu doi *Playeri*.

Scopul proiectului este acela de a construi un algoritm de vedere artificială care să recunoască poziția și cifra de pe fiecare nouă piesă așezată pe tabla de Mathable (*Task 1 și Task 2*). Cu ajutorul acestora, algoritmul va prelucra scorul fiecărui jucător după fiecare rundă (*Task 3*).

Reguli:

Fiecare jucator, pornind de la numerele deja existente pe tablă trebuie să formeze ecuații matematice valide, fie pe orizontal, fie pe vertical cu numerele extrase. Ecuațiile trebuie să fie valide din cel puțin un sens (în sus, jos, stânga sau dreapta), dar există posibilitatea ca ele să fie valide din mai multe sensuri. În consecință jucatorul primește punctele de pe piesă de câte ori piesa lui satisface o ecuație. În materie de bonusuri, pătrățelele galbene și mov oferă și ele puncte în plus de x 3 sau x 2.

Jocul are și constrângeri (casuțele albastre). Este permis să plasezi o piesă pe acele casuțe doar dacă ecuația formată satisface simbolul matematic de pe acestea.

2 Analiza și prelucrarea datelor

Prima și cea mai importantă etapă în acest proiect este analiza datelor. În cele 200 de imagini de antrenare care cumulează mutările a 50 de runde în 4 jocuri putem ușor observa contrastul între tabla de *Mathable* și masă. De aceea, am ales să le separ folosind *Trackbar*-ul din laboratorul 7:

```
    cv.namedWindow("Trackbar")
    cv.createTrackbar("LH", "Trackbar", 0, 255, nothing)
    cv.createTrackbar("LS", "Trackbar", 0, 255, nothing)
    cv.createTrackbar("LV", "Trackbar", 0, 255, nothing)
    cv.createTrackbar("UH", "Trackbar", 255, 255, nothing)
    cv.createTrackbar("US", "Trackbar", 255, 255, nothing)
    cv.createTrackbar("UV", "Trackbar", 255, 255, nothing)
```

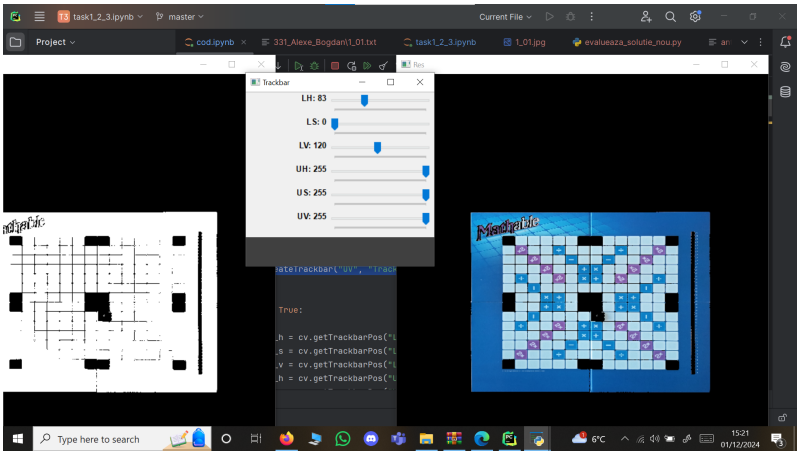

Figura 2

Se aplică apoi funcții pentru a găsi marginile, apoi colturile tablei și o transformare de perspectivă pentru ca imaginea să fie dreaptă și nedeformată, astfel separând masa de jocul *Mathable*:

```
    edges =  cv.Canny(thresh ,50,150)
    contours, _ = cv.findContours(edges,  cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    .....
    M = cv.getPerspectiveTransform(puzzle,destination_of_puzzle)
```

În figura de mai jos se poate vedea cum a detectat *cv.Canny* marginile tablei.

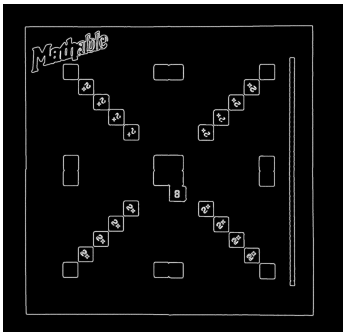

Figura 3

Se observă acum că tabla este compusă din 2 elemente: careu ce conține căsuțele și un contur cu sigla jocului. Pentru ușurință, în continuare, se va lucra doar cu tabla cea mică. Distanța de la colțurile tablei mici la cele ale tablei mari sunt invariante, astfel am prestabilit pozițiile pixelilor colțurilor pe axele Ox și Oy, decupând careul cu pătrățele (am folosit aceeași funcție ca mai sus : $cv.getPerspectiveTransform$).

3 Abordarea pentru Task1

Primul task presupune localizarea casuței (Ox si Oy) pe care s-a pus o nouă piesă în fiecare imagine. Având deja careul pregătit, cerința se poate îndeplini prin folosirea diferenței dintre imaginea anterioară și imaginea curentă. Mai exact, făcând diferența între două imagini care au aproape aceeași configurație va iesi în evidentă locul în care diferența este cea mai mare. Acesta coincide și cu locul unde a fost lipită noua piesă. Pentru prima imagine din fiecare joc m-am folosit de o imagine auxiliara cu o tablă goală Figura 1.

Este folosit un grid de 14x14 linii verticale și orizontale de 80 de *pixeli* pentru a separa careul în *patch*-uri reprezentând o casuță. Imaginile sunt transformate în tonuri de gri ($cv.COLORBGR2GRAY$) pentru simplificarea analizei, apoi în *array*-uri de $int32$ (int32 se folosește pentru evitarea unui overflow). Am tăiat din fiecare *patch* un padding de 20 de pixeli din toate direcțiile, lăsând la comparare doar o bucată centrală pentru a evita compararea elementelor nefolositoare din imagini (cum ar fi marginea albastră dintre casuțe, umbre etc). Se iterează prin toate cele 196 de casuțele și se alege patch-ul cu diferența maxima.

4 Abordarea pentru Task2

Al doilea task presupune determinarea, pe langă poziția piesei, și a numărului scris pe aceasta. Am abordat această cerința prin folosirea noțiunii de **Template Matching**.

4.1 Obținerea Șabloanelor

Pentru siguranța ca numerele să se potrivească cât mai bine, se folosește o poză auxiliară cu tabla plină de toate modelele diferite de piese Figura 1 , cu ajutorul căreia am "decupat" fiecare numar. Imaginile cu numerele din joc sunt transformate în $alb - negru$, netezindu-le. Se aplică, de asemenea, și operatorii morfologici: *Eroziune* și *Dilatare* . Am observat ca numerele cu o singură cifră trebuie "decupate" astfel încât în jurul lor să existe un spațiu liber destul de generos. În contrast, cele cu 2 cifre trebuie "decupate" cât mai strâns de număr. Acest lucru împiedică, de exemplu, confundarea clasică 1 cu 11.

```
    kernel = np.ones((2, 2), np.uint8)
    patch = get_patch(img_cropped, row, col, top, bottom, left, right)
    _, thresh = cv.threshold(cv.cvtColor(patch, cv.COLOR_BGR2GRAY), 100, 255, cv.
    THRESH_BINARY_INV)
    dilated = cv.erode(thresh, kernel)
    dilated = cv.dilate(dilated, kernel)
```

4.2 Template Matching

Temaplate Matching este o tehnică clasică de vedere artificială utilizată pentru a localiza poziția unei imagini șablon într-o imagine mai mare. OpenCV oferă funcția cv.matchTemplate() pentru această sarcina. Glisează imaginea șablonului peste imaginea de intrare (similar cu convolutia 2D) și compară șablonul cu secțiunea din imaginea de intrare de sub el.

Figura 4: Exemple de șabloane

După extragerea șabloanelor, se vor prelucra patch-urile ce vor fi comparate cu acestea (binarizare, eroziune și dilatare).

```
corr = cv.matchTemplate(dilated,img_template,  cv.TM_CCOEFF_NORMED)
```

Orice confuzie de numere (de exemplu 16 cu 18) pe care am întâlnit-o s-a putut rezolva prin micșorarea sau mărirea spațiului negru din jurul șabloanelor.

5 Abordarea pentru Task3

Al treilea task presupune calcularea scorului fiecărui jucător după fiecare rundă. Rundele în care fiecare jucător lipește o piesă sunt extrase din fișierul *turns* corespunzător fiecărui joc. Aceste runde sunt extrase sub forma unui array la care am adăugat numărul 51, astfel încât când vectorul este parcurs să nu se acceseze o zonă de memorie nealocată (motivare mai amănunțită se poate găsi în logica algoritmului de mai jos). De asemenea, am returnat si numele primului jucător pentru a putea reproduce fișierul *score.txt* corect în final.

Logica algoritmului

Toate cerințele sunt rezolvate prin rularea unui bloc de cod al cărei logici este următorea:

- Pentru fiecare joc îmi setez variabilele în starea lor neutră (ex: score player1=0, board conf=first board conf). Variabila *board conf* este o matrice de 14x14 care va ajuta la extragerea scorului și care, pentru fiecare joc, este completată la poziția corespunzătoare cu piesa găsită. Extrag *array*-ul de runde cum am descris mai sus și stabilesc cine este primul player. Vectorul va avea mereu pe prima poziție cifra 1 (fiind mereu indicele primei poze), în consecință sar peste acest prim element, deoarce vreau să scriu scorurile găsite de fiecara dată când o valoare din *array* corespunde cu un indice de poza .
- Pentru fiecare poză din fișier verific că aceasta să aparțină jocului curent.
- Când numărul din vectorul *turns* corespunde cu numărul pozei citite, înseamnă că jucătorii se interschimbă, deci trebuie scris în fișier scorul.
- Poziția și identitatea piesei le extrag cu funcțiile descrise mai sus, apoi le pun în matricea *board conf*
- Pentru **calcularea scorului** logica este urmatoarea: am o matrice generală numită *game table* în care se țin toate restricțiile și bonusurile unei table de Mathable și 5 funcții (4 care îmi verifică în sus, jos, stânga și dreapta de câte ori se alcatuiește o ecuație matematică validă, respectând constrângerile din *game table* și una pentru a vedea dacă piesa a fost pusă pe o poziție de *bonus*).

6 Concluzie

În conluzie, cele mai importante elemente din cadrul vederii artificiale pe care le-am folosit în soluționarea acestui proiect sunt *Canny Edge Detection, Geometric Image Transformations și Template Matching* . Ideeile principale ale soluției sunt: *detectarea marginilor și tăierea imaginilor* în careuri mai usor de lucrat, *diferența dintre poze consecutive* pentru obținerea poziției piesei noi puse, *prelucrarea numerelor sub formă de șabloane folosite în Template Matching* și în final calcularea scorului pe măsură ce înaintăm în citirea imaginilor pe baza unei *matrici reprezentative.*