Calcularea automata a scorului in jocul "Double Double Dominoes"

Tindeche Alexandru
Grupa 331

December 4, 2023

Contents

1 Preluarea datelor

Primul lucru pe care il fac este sa declar o variablia "base_path" care este radacina caii catre folderul cu imagini. Apoi, intr-un string, citesc tot fisierul cu mutari, din folderul a carei caii am declarat-o mai devreme. Striu ca in acest fisier text, am pe fiecare linie denumirea imaginii si jucatorul care va face mutarea, separate printr-un spatiu. Acest spatiu va fi inlocuit cu caracterul "-" si voi split-ui acest string dupa "\n" pentru a obtine o lista cu mutari. Mai tarziu, stiind ca fiecare dintre aceste elemente ale listei este ceva de forma imagine-jucatorN, voi prelua doar ultimul caracter pentru a vedea mutarea.

Apoi, intr-un loop cu 20 de iteratii, voi face toti pasii necesari pentru a calcula scorul fiecarei runde.

Logica algoritmului este urmatoarea: fiecare imagine, o compar cu cea anterioara. Astfel ca pentru fiecare imagine apelez o functie numita comparare_matrici, cu doi parametrii: imaginea curenta (img0) si imaginea de comparat (img). Daca avem de a face cu prima imagine, img0 va avea valoarea Null.

2 Functia "comparare_matrici"

Logica functiei presupune procesarea separata a celor doua imagini date ca parametru.

Imaginile vin ca string-uri si sunt citite din fisier cu ajutorul librariei OpenCV, apoi extragem careul central.

2.1 Identificarea careului central

Pentru a putea calcula scorul unei runde este nevoie, in primul rand, sa identificam piesele de domino puse pe careul central. Iar pentru acest lucru trebuie sa identificam corect careul central. Astfel ca apelam functia "extrage_careu" care are ca parametrii imaginea citita, adresa si un parametru boolean care imi spune daca poza este prima din set sau nu.

Pentru a putea extrage careul central am scris functia "extrage_careu" care face diferite modificari asupra imaginii pentru a putea identifica careul central precum:
1. Modificarea luminozitatii imaginii la o valoare constanta:

```
def adjust_luminosity(image, target_brightness=83.7):
  # Calculate the mean brightness of the image
  mean_brightness = np.mean(image)

  # Calculate the adjustment factor
  adjustment_factor = target_brightness - mean_brightness

  # Adjust the brightness of the image
  adjusted_image = image + adjustment_factor

  # Clip the values to be in the range [0, 255]
  adjusted_image = np.clip(adjusted_image, 0, 255).astype(np
   .uint8)

  # show_image('adjusted_image',adjusted_image)

  return adjusted_image
```

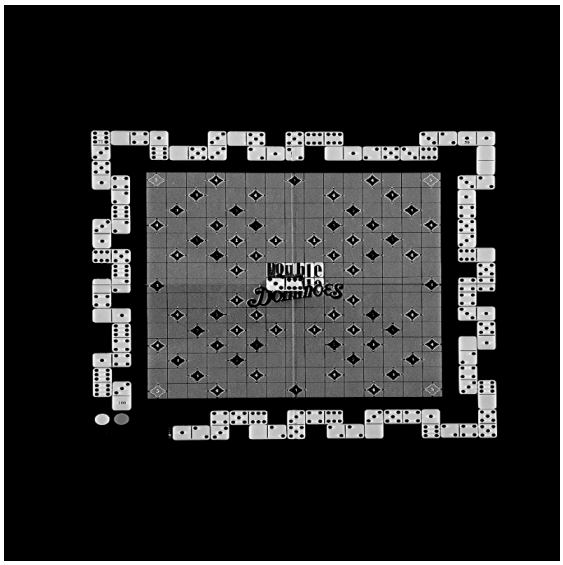
2. Se aplica o filtrare a culorilor HSV, incercand sa pastrez culorile cat mai apropiate de cea albastra a tablei de joc; astfel ca am eliminat culoarea maro a mesei pe care se afla tabla, si culoarea maro a marginii tablei de joc

```
def filter_blue(image):
  # Convert BGR to HSV
  hsv = cv.cvtColor(image, cv.COLOR_BGR2HSV)

  # # Convert specific blue from RGB to HSV
```

```
    blue_rgb = np.uint8([[[2, 143, 178]]]) # RGB color
    blue_hsv = cv.cvtColor(blue_rgb, cv.COLOR_RGB2HSV)[0][0]

    # Define a small range around the converted HSV blue color
    lower_blue = np.array([blue_hsv[0] - 25, 50, 50])
    upper_blue = np.array([blue_hsv[0] + 10, 255, 255])


    # Threshold the HSV image to get only blue colors
    mask = cv.inRange(hsv, lower_blue, upper_blue)
    res = cv.bitwise_and(image, image, mask=mask)

    return res
```

3. Am aplicat un filtru de sharpen pentru a scoate in evidenta detaliile, apoi un threshold binar care are ca rol evidentierea careului din centru, si diferite filtre de dilate, erode si close pentru a scoate in evidenta marginile careului.

Figure 1: Imaginea originala dupa filtrul HSV si sharpen

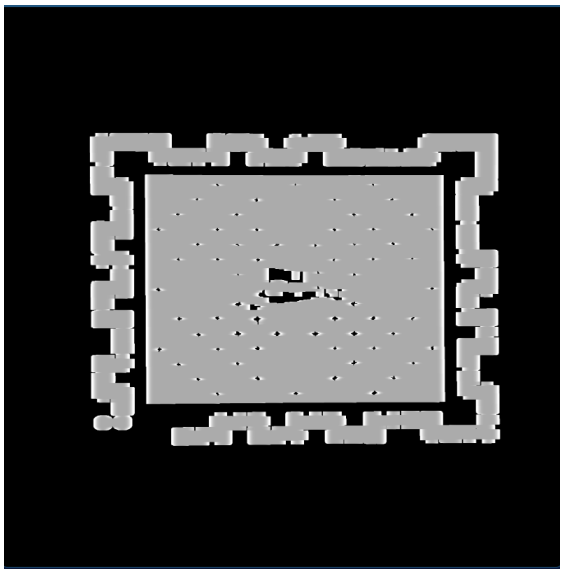


4. Peste marginile acum scoase in evidenta, am mai aplicat cateva filtre si un threshold calculat dinamic pentru a facilita extragerea de edge-uri

```
    md = np.median(thresh)
    lower = int(max(0, (1.0 - 0.33) * md))
    upper = int(min(255, (1.0 + 0.33) * md))
```

5. Am aplicat un filtru Canny pentru a extrage edge-urile, iar apoi am folosit functia OpenCV pentru gasirea contururilor, cu parametrii RETR _ EXTERNAL si CHAIN _ APPROX _ SIMPLE

Figure 2: Imaginea originala dupa erode si dilate



6. Contururile acum detectate sunt filtrate printr-un list comprehention, pentru a pastra contururile care au o forma cat mai apropiata de un patrat, folosind functia OpenCV boundingRect care returneaza un tuplu de forma (x, y, w, h), unde (x, y) sunt coordonatele dreptunghiului din stanga sus, w este latimea si h inaltimea dreptunghiului. Astfel ca folosesc raportul `cv.boundingRect(cnt)[2]/cv.boundingRect(cnt)[3]`

7. Precum in laborator, intr-un for loop, parcurg contururile si retin colturile conturului cu aria cea mai mare, care stiu ca este careul din centru pe care il caut. In acest loop, iau fiecare contur, si daca are mai mult de 3 puncte, adica stiu ca poate fi ceva de forma macar patrulatera si, iterand prin puncte, si gasind potentiale puncte care sunt "mai la dreapta" sau "mai la stanga" decat punctele gasite anterior

8. Dupa aceea, am implementat un sistem care, pentru prima imagine retine coordonatele, iar altfel, daca a existat o eroare in detectarea colturilor si unul ditre puncte este deviat cu mai multe de 150 de pixeli, atunci o sa pastreze punctele de la prima imagine, fapt indicat de parametrul boolean "first_image", astfel avand o plasa de siguranta.

9. Iau punctele detectate anterior si cu ajutorul cv.getPerspectiveTransform, "decupez" careul din centru

10. Un ultim lucru pe care il fac este sa filtrez culoarea bej a patratelelor cu numere care sunt foarte apropiate ca si culoare de culoarea dominourilor, pentru a evita detectarea fals pozitiva in algoritmul de identificare a patratelelor ocupate

Rezultatul, o imagine, este pasata inapoi spre functia apelanta, si stocata in parametrul "result".

2.2 Trasarea caroiajului

Careul central este dispus ca o matrice 15x15, astfel ca am "desenat de mana", adica hardcodat, liniile verticale si orizontale sub forma de linii OpenCV. Acum ca am cele 15x15 casute stabilite, pot sa determin care dintre acestea sunt ocupate, si care nu sunt ocupate.

```
1   off = 0
2
3   lines_horizontal0=[]
4   for i in range(10,1520,97):
5     l=[]
6     l.append((0,i + off))
7     l.append((1500 ,i + off))
8     lines_horizontal0.append(l)
9     if len(lines_horizontal0) == 8:
10      off = 5
11
12  off_vertical = 5
13  lines_vertical0=[]
14  for i in range(23,1525,97):
15    l=[]
16    l.append((i + off_vertical,0))
17    l.append((i + off_vertical,2000))
18    lines_vertical0.append(l)
19    if len(lines_vertical0) == 8:
20      off_vertical = 0
```

Se observa in codul de mai sus ca liniile orizontale pornez de la linia 10 pana la 1520 din 97 in 97 de pixeli, iar cele orizontale de la 23 la 1525 tot din 97 in 97 de pixeli. La un calcul simplu vedem ca vor fi 15 linii orizontale si verticale. La abele tipuri de linii avem un offset, care se aplica de la jumatate, pentru a plasa cat mai aproape de liniile reale ale careului de joc, deoarece imaginea este putin deformata si liniile se departeaza usor-usor.

Mai apoi, implementez un threshold binary inverted pentru a incerca sa filtrez culorile si a face zonele ocupate de piese de domino cat mai intunecate, atfel incat sa pot implemeta o conditie, ca daca media intensitatii culorilor (in greyscale, cu cat este mai mare valoarea unui pixel cu atat acesta este mai luminos), este mai mica decat o valoare, sa stiu ca acea casuta e clar ocupata de o piesa.

```
1   _, thresh = cv.threshold(result, 190, 230, cv.
     THRESH_BINARY_INV)
```

2.3 Detectarea casutelor ocupate

Dupa logica descrisa anterior, voi sti ca o casuta este ocupata de o piesa, daca media intensitatilor pixelilor este mai mica decat o medie prestabilita. Totusi, in experimentele mele, am descoperit ca imaginile pot diferi in luminozitate, iar daca pe o imagine am o medie fixa care functioneaza, aceasta poate sa nu mai functioneze si pe alte imagini. Asadar, am facut doua functii "stabilire_medie" si "medie_totala" care stabilesc in mod dinamic valoarea de comparat. Cum fac asta? Functia "stabilire_medie" este folosita pentru prima imagine si calculeaza mean-ul celor 9 patchuri de pe mijloc si le sorteaza crescator. Apoi pasam catre functia care detecteaza casutele ocupate ca mean superior cea de-a 3-a medie, pentru ca primele doua medii ca intensitate o sa fie a celor 2 casute ocupate de piese, iar a 3a este, in principiu, a 3a casuta pe care sunt scrise literele "le" din cuvantul Double, pentru ca textul este foarte apropiat ca si culoare si numar de pixeli alb-negrii de o piesa de domino. Cea de-a doua functie, "medie_totala", calculeaza media tuturor casutelor de pe tabla, iar ca medie superioara pasam numarul rundei * 2. Facem acest lucru, pentru ca stim ca in fiecare runda avem exact nr rundei piese de domino, iar o piesa de domino ocupa 2 patratele. De exemplu la runda 2, avem 2 piese de domino, adica 4 casute ocupate.

```
1   def stabilire_medie(image, lines_horizontal,
     lines_vertical):
2   # Luam mediile celor 9 patratele din centru si le sortam
3   offset = 20
4   medii = []
5   for i in range(6, 9):
6     for j in range(6, 9):
7       y_min = lines_vertical[j][0][0] + offset
8       y_max = lines_vertical[j + 1][1][0] - offset
9       x_min = lines_horizontal[i][0][1] + offset
10      x_max = lines_horizontal[i + 1][1][1] - offset
11      patch = image[x_min:x_max, y_min:y_max].copy()
12      medie_patch = np.mean(patch)
13      medii.append(medie_patch)
14  medii.sort()
15  return medii
```

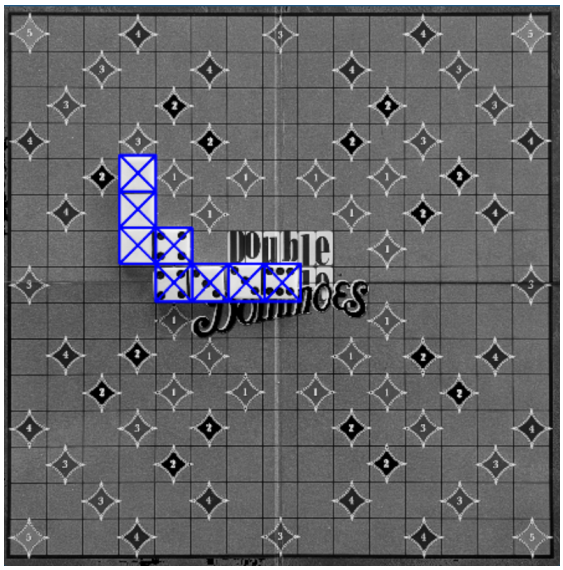
```
1   def medie_totala(image, lines_horizontal, lines_vertical):
2   # Calculez media tuturor patratelelor din care e alcatuita
      tabla
3   # Sortez mediile si iau primele num_piese
4   offset = 20
5   medii = []
6   for i in range(len(lines_horizontal)-1):
7     for j in range(len(lines_vertical)-1):
```

```
        y_min = lines_vertical[j][0][0] + offset
        y_max = lines_vertical[j + 1][1][0] - offset
        x_min = lines_horizontal[i][0][1] + offset
        x_max = lines_horizontal[i + 1][1][1] - offset
        patch = image[x_min:x_max, y_min:y_max].copy()
        medie_patch = np.mean(patch)
        medii.append(medie_patch)
    medii.sort()
    return medii
```

Aceste date le pasam catre functia "determina_configuratie_careu_ox" luata din laborator si modificata, pentru a folosi media calculata dinamic anterior. Aceasta functie parcurge fiecare casuta delimitata de liniile trasate anterior, face mean-ul fiecarei casute, iar daca aceasta este mai mica decat o anumita medie o considera ocupata, altfel libera. La inceputul functiei declaram o matrice 15x15 care mimeaza tabla de joc, iar pe pozitiile care sunt ocupate punem x, altfel o.

Daca se doreste vizualizarea configuratiei, se va decomenta apelul functiei "vizualizare_configuratie", care "deseneaza" pe imagine un X peste casuta ocupata de o piesa, un sainty check.

Figure 3: Imaginea cu careul extras si identificarea casutelor ocupate



2.4 Determinarea numarului de pe dominouri

Pentru a calcula scorul dintr-o runda, ne intereseaza sa vedem ce piesa noua a fost adaugata de jucator. Astfel ca o sa detectam casutele ocupate din aceasta runda si vom compara cu matricea unde am marcat casutele ocupate din runda trecuta. Pentru prima

imagine, nu avem cu ce runda sa comparam, asa ce ne uitam doar pe pozitiile in care avem marcat cu litera x ca fiind ocupata.

Pentru determinarea numarului reprezentat pe dominouri este folosita functia "determina_configuratie_cifre" care primeste ca parametru careul original, imaginea pe care am aplicat threshold-ul pentru evidentierea pieselor, liniile, matricea unde sunt marcate casutele ocupate de piese si pozitiile i, j unde am detectat o casuta ocupata, care la runda de dinainte era libera.

Astfel, intr-un for pentru fiecare casuta, verificam daca aceasta este ocupata in runda curenta, dar nu a fost in cea anterioara, iar daca este nou ocupata, apelam functia care mentionata mai sus care sa ne indice ce numar de pe domino se afla in casuta aceea.

In cadrul functiei declaram o noua matrice 15x15 in care notam numerele corespunzatoare numarelor de pe piesa de domino, iar apoi luam toate patratelele, ca in laborator, si verificam daca aceasta este ocupata si este si la coordonatele indicate prin parametrii, atunci aplicam patchului o alta functie denumita "pattern_matching" care are rolul de a gasi numarul.

Altfel, punem -1 in matrice.

2.4.1 Pattern Matching

In aceasta functie care primeste ca parametru un patch, voi aplica tehnica de multiscale template matching, care presupune a avea un template, pe care incep sa il suprapun, in mai multe dimensiuni, peste o imagine, si a vedea cat de bine se coreleaza, astfel stiind daca acel template se regaseste sau nu in imaginea in care vreau sa il gasesc.

Asadar, in algoritmul scris de mine, am aplicat mai intai un blur gaussian pentru a elimina zgomotul, apoi am marit rezolutia imaginii cu cv.resize .

```
1   patch = cv.cvtColor(patch, cv.COLOR_BGR2GRAY)
2   patch = patch.astype('uint8')
3   patch = cv.GaussianBlur(patch, (0, 0), 3)
4   # Raise resolution
5   patch = cv.resize(patch, (0, 0), fx=2, fy=2)
```

Apoi, parcurg imaginile-template - care au fost extrase decupand piesele din imaginilie auxiliare prezente in datele de antrenare si distribuite in functie de numarul piesei in foldere diferite numerotate de la 1 la 6 - pe care aplic acelasi blur gaussian si resize si intr-un loop in care iau template-ul curent si il incerc in diferite dimensiuni cu cv.matchTemplate, retin cifra care a avut cea mai mare corelatie, apoi o returnez. Astfel gasesc cifra din patc-ul respectiv.

```
def pattern_matching (patch):
  patch = cv.cvtColor(patch, cv.COLOR_BGR2GRAY)
  patch = patch.astype('uint8')
  patch = cv.GaussianBlur(patch, (0, 0), 2)
  # Raise resolution
  patch = cv.resize(patch, (0, 0), fx=2, fy=2)
  # show_image('patch',patch)
  base_path_identifying_numbers = 'imagini_auxiliare\\'

  max_correlation = -np.inf
  poz = None

  for i in range(0, 7):
    folder = os.path.join(base_path_identifying_numbers, f'{i}')
    # show each image in the folder
    with os.scandir(folder) as entries:
      for entry in entries:
        image_path = os.path.join(folder, entry.name)
        if not os.path.exists(image_path):
          print(f"Image path {image_path} does not exist.")
        image = cv.imread(image_path)
        if image is None:
          print(f"Image at {image_path} could not be loaded.")
        # show_image('image',image)
        image_template = image.copy()
        image_template = cv.cvtColor(image_template, cv.COLOR_BGR2GRAY)
        image_template = cv.GaussianBlur(image_template, (0, 0), 2)
        image_template = cv.resize(image_template, (0, 0), fx=2, fy=2)
        # Crop the image to get only the center
        image_template = image_template[10:-10, 10:-10]
        # show_image('image_template',image_template)
        # Multiscale template matching
        for scale in np.linspace(0.2, 1.0, 20)[::-1]:
          resized = cv.resize(image_template, (0, 0), fx=scale, fy=scale)
          # show_image('resized',resized)
          # if the resized image is smaller than the template, then break from the loop
```

```
37            # if resized.shape[0] < patch.shape[0] or resized.
      shape[1] < patch.shape[1]:
38            #   print
39            #   break
40            # Apply template Matching
41            result = cv.matchTemplate(patch, resized, cv.
      TM_CCOEFF_NORMED)
42            _, max_val, _, max_loc = cv.minMaxLoc(result)
43            # check to see if the iteration should be
      visualized
44            # draw a bounding box around the detected region
45            if max_val > max_correlation:
46              max_correlation = max_val
47              poz = i
48              # print(poz)
49    # print(poz)
50    if poz != None:
51      return poz
```

Inapoi in functia "comparare_matrici", preiau aceasta matrice cu numere (in care acum am numerele identificate, pe pozitiile casutelor unde am noua piesa) si ma folosesc de aceasta pentru a calcula scorul rundei.

3 Calcularea scorului rundei curente

Dupa ce am scris in fisier pozitiile si numerele de pe noua piesa de domino, le returnez catre functia apelanta main, unde voi calcula punctajul rundei, in functie de aceste doua date.

Pentru fiecare joc, incepem prin a reseta scorul celor doi jucatori la -1. Pentru calcularea scorului am facut un array cu numerele de pe margine pentru a putea urmari trasul pionilor si pentru a acorda bonusul corespunzator. Pentru punctajul de pe tabla (casutele cu numere) am facut o matrice 15x15 care are exact casutele de pe tabla cu numere, daca este cazul, daca nu 0. Pentru fiecare runda resetam scorul rundei la 0. Apoi, verificam daca este cazul sa acordam punctaj bonus, inainte de orice: verificam daca pionul se afla pe traseu (daca scorul unui jucator este diferit de -1) si acordam punctajul bonus sau nu in functie de caz si, daca este cazul, adaugam bonusul si la scorul rundei. Apoi, daca piesa a fost asezata pe unul dintre un patratel cu punctaj, acordam punctajul celui care a facut mutarea tura aceasta, iar daca piesa este dubla, mai adaugam odata punctajul si il socotim la punctajul rundei.

La sfarsit, scriem scorul in fisier.

List of Figures