UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Concepte și Aplicatii în Vederea Artificială

Tema 1

Calculator automat de scor pentru jocul Double Double Dominoes

Iancu Florentina-Mihaela

CTI, grupa 461

Cuprins

Configurări inițiale

Proiectul a fost creat folosind un server Jupyter și Python 3 în PyCharm.

În dezvoltarea acestui program am folosit următoarele librării: os, numpy (as np), math, cv2 (as cv).

Toate path-urile utilizate sunt scrise în variabile la începutul codului.

Am utilizat câteva elemente hardcodate, precum punctajul de pe tabla de Double Double Dominoes sau traseul pionilor. De altfel, deoarece una dintre pozițiile pieselor este o literă, am utilizat un vector *num_to_letter[]* pentru a memora alfabetul și a face schimbarea pentru de afișarea rezultatului final.

```
side_game = [-1,1,2,3,4,5,6,0,2,5,3,4,6,2,2,0,3,5,4,1,6,2,4,5,5,0,6,3,4,2,0,1,5,1,3,4,4,4,5,0,6,3,5,4,1,3,2,0,0,1,1,2,3,6,3,5,
 2,1,0,6,6,5,2,1,2,5,0,3,3,5,0,6,1,4,0,6,3,5,1,4,2,6,2,3,1,6,5,6,2,0,4,0,1,6,4,4,1,6,6,3,100]

num_to_letter = ['','A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
 'V', 'W', 'X', 'Y', 'Z']

main_game =[
    [-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1],
    [-1,5,0,0,4,0,0,0,3,0,0,0,4,0,0,5],
    [-1,0,0,3,0,0,4,0,0,0,4,0,0,3,0,0],
    [-1,0,3,0,0,2,0,0,0,0,0,2,0,0,3,0],
    [-1,4,0,0,3,0,2,0,0,0,2,0,3,0,0,4],
    [-1,0,0,2,0,1,0,1,0,1,0,1,0,2,0,0],
    [-1,0,4,0,2,0,1,0,0,0,1,0,2,0,4,0],
    [-1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0],
    [-1,3,0,0,0,0,0,0,0,0,0,0,0,0,0,3],
    [-1,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0],
    [-1,0,4,0,2,0,1,0,0,0,1,0,2,0,4,0],
    [-1,0,0,2,0,1,0,1,0,1,0,1,0,2,0,0],
    [-1,4,0,0,3,0,2,0,0,0,2,0,3,0,0,4],
    [-1,0,3,0,0,2,0,0,0,0,0,2,0,0,3,0],
    [-1,0,0,3,0,0,4,0,0,0,4,0,0,3,0,0],
    [-1,5,0,0,4,0,0,0,3,0,0,0,4,0,0,5]
]
```

În ceea ce privește ordinea rulării, programul accesează folderul cu imaginile de joc și pentru fiecare imagine extrage numărul jocului și al rundei. Dacă imaginea curentă este *x_01*, reprezentând prima mutare dintr-un joc, atunci programul extrage o poză cu tabla de joc goala și o consideră ca fiind imaginea anterioară mutării curente.

Pentru fiecare imagine nouă, se folosește fișierul *x_mutări.txt* din care se extrage numărul player-ului care a efectuat mutarea.

Task 1. Poziția piesei

Pentru a determina poziția mutării din poza curentă am folosit detectarea marginilor și diferența dintre tabla anterioară și cea curentă.

În primul rând, folosind funcția *extrage_careu()* din Laboratorul 6, am modificat-o ca să extrag numai tabla centrală, cea albastră pe care sunt plasate piesele. Am folosit un interval care cuprinde toate nuanțele de albastru necesare și am folosit zona găsită pentru a crea o mască.

```
#Cautam albastru in imagine
kernel = np.ones((5, 5), np.uint8)
hsv = cv.cvtColor(img_copy, cv.COLOR_BGR2HSV)
#Interalul pentru detectarea culorii
lower_blue = np.array([10, 150, 50])
upper_blue = np.array([170, 255, 255])

blue_mask = cv.inRange(hsv, lower_blue, upper_blue)
# Mask = cv.erode(Mask, kernel, iterations=1)
# Mask = cv.morphologyEx(Mask, cv.MORPH_OPEN, kernel)
blue_mask = cv.dilate(blue_mask, kernel, iterations=1)
blue_mask = cv.bitwise_not(blue_mask)
blue_mask = cv.medianBlur(blue_mask,9)
blue_mask = cv.GaussianBlur(blue_mask, (5,5), 0)
cv.addWeighted(blue_mask, 2.0, blue_mask, -0, 0, blue_mask)
```

Restul codului din funcție determină marginile careului albastru și în final returnează coordonatele fiecărui colț. Utilizarea unei imagini obținute folosind funcția *cv.warpPerspective()* cauza câteva probleme, așa că imaginea rezultată nu a fost utilizată în versiunea finală.

După obținerea coordonatelor careului se face un crop imaginii originale. Pentru a regla câteva probleme în următorul task, am adăugat 20 pixeli în partea de jos și cea din dreapta, precum și un border de 10 pixeli în toate direcțiile.

Din măsurătorile acestea, putem obține lungimea și lățimea unui pătrățel de pe tablă (L/15).

```
square_vertical = round((bottom_right[1]-top_left[1])/15)
square_horizontal = round((bottom_right[0]-top_left[0])/15)
```

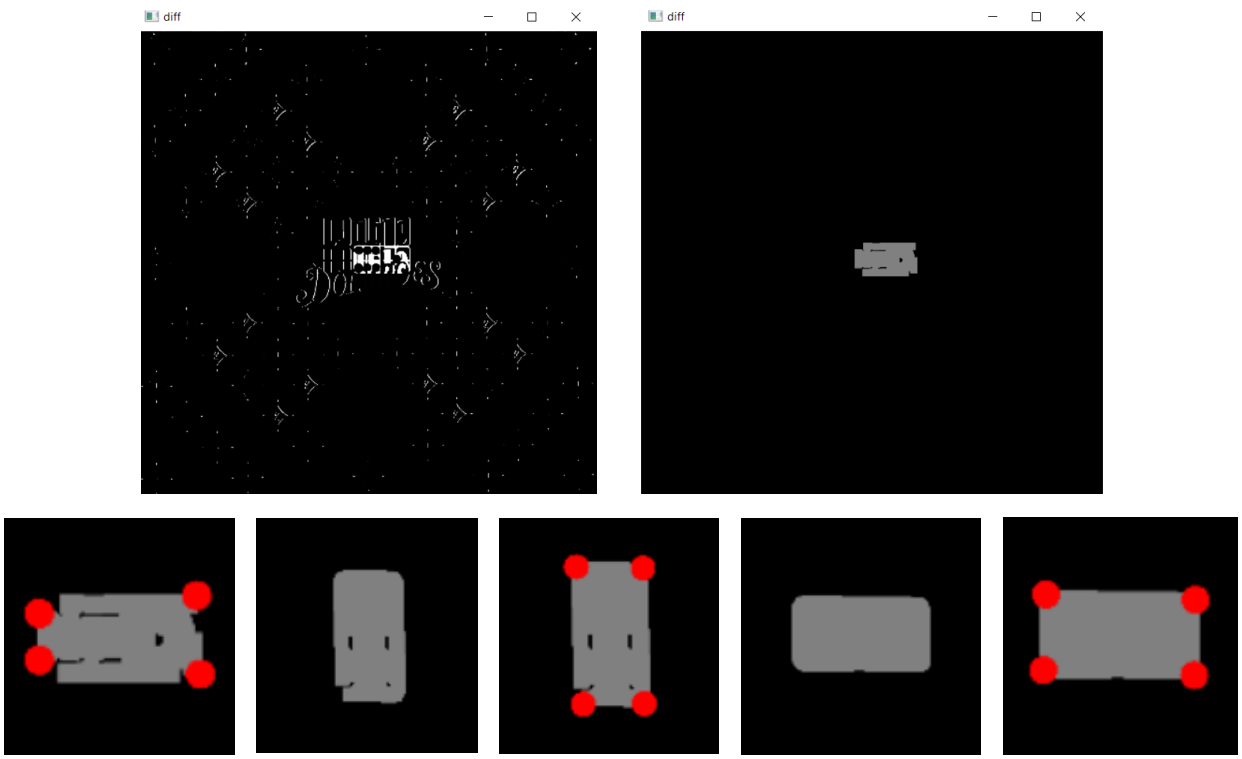
Acum că avem tabla din poza curentă și cea din poza anterioară, apelăm funcția *image_difference()* pentru a obține o imagine alb negru care evidențiază poziția piesei adăugate.

```
# image_g_blur = cv.GaussianBlur(image_m_blur, (5, 5), 0)
cv.addWeighted(img1, 0.8, img1, -0.3, 0, img1)
_, thresh = cv.threshold(img1, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
thresh = cv.erode(thresh, kernel)

# image_g_blur2 = cv.GaussianBlur(image_m_blur2, (5, 5), 0)
cv.addWeighted(img2, 0.8, img2, -0.3, 0, img2)
_, thresh2 = cv.threshold(img2, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)
thresh2 = cv.erode(thresh2, kernel)

difference = abs(thresh2-thresh)
# show_image('diff',difference)
difference = cv.erode(difference,kernel=np.ones((6,6), np.uint8), iterations=1)
cv.addWeighted(difference, 0.5, difference, 0, 0, difference)
difference = cv.dilate(difference,kernel=np.ones((5,5), np.uint8), iterations=6)
```

Iar rezultatele codului de mai sus sunt:



Urmează să apelăm funcția *extrage_piesa()* pentru a obține coordonatele colțurilor piesei. Cu aceste coordonate putem determina poziția piesei pe tablă.

Utilizăm funcția *piece_position()*. Fiindcă am aflat deja care este lungimea și lățimea unui pătrat din interiorul careului, utilizând colțul din stânga sus (Ne interesează să găsim poziția jumătății din stânga sau de sus) și adăugând o marjă de eroare de jumătate de pătrățel ca să obținem aproximativ mijlocul acelui pătrățel, împărțim coordonata noastră la măsurătorile unui pătrat.

Punctul din mijloc al unui pătrat are cea mai mare șansă de a rămâne pe poziția corectă în ciuda eventualelor mișcări ale piesei. De aceea îl folosim ca punct de reper (marja de eroare menționată anterior).

```
x = math.floor((top_left[1] + square_vertical/2) / square_vertical) + 1
y = math.floor((top_left[0] + square_horizontal/2) / square_horizontal) + 1

first_piece = [x,y,num_to_letter[y]]

if top_len > left_len or bottom_len > right_len:
    # piesa este orizontala
    # a doua piesa este pe aceeasi linie, pe urmatoarea coloana
    isHorizontal = True
    second_piece = [x, y+1, num_to_letter[y+1]]
else:
    # piesa este verticala
    # a doua piesa este pe aceeasi coloana, pe urmatoarea linie
    isHorizontal = False
    second_piece = [x+1, y, num_to_letter[y]]
```

Variabilele *first/second_piece* conțin pozițiile celor două jumătăți ale piesei, dar și litera care corespunde coordonatei y pentru afișarea finală.

Task 2. Numerele piesei

Din cauza modificărilor făcute la pasul anterior, calitatea pozelor alb negru ale pieselor nu pot fi folosite drept mască pentru task 2. Rezultatele variază în funcție de cât de întreagă era piesa.

Am considerat că printr-un simplu calcul matematic putem obține coordonatele generale ale piesei. Astfel, folosind din nou colțul din stânga sus, calculăm coordonatele și decupăm piesa din poza cu tabla. Pentru a determina numerele de pe piesă, am decis să folosesc template-uri pentru fiecare număr, așa că din poza cu piesa întreagă trebuie să decupez cele două jumătăți:



Acum că avem cele două poze cu jumătățile, folosim funcția *number_classification()* care verifică poza dacă cu fiecare dintre imaginile template și determină care dintre ele este mai similară. Deoarece numărul 0 avea o similaritate de 100% cu toate piesele, am dedus că o similaritate foarte mică cu toate celelalte template-uri reprezintă numărul 0. Dacă procentajul obținut este mai mic de 40%, o să fie considerat 0.

Luând în considerare faptul că piesele pot să fie răsucite pe tablă, există template-uri pentru varianta rotită a numerelor 2,3 și 6. Celelalte numere nu se schimbă indiferent de orientare.

```
def number_classification(patch):
        maxi = -np.inf
        poz = -1
        patch_shape = patch.shape
        files_template = os.listdir(folder_template)
        patch = cv.cvtColor(patch,cv.COLOR_BGR2GRAY)
        _, patch = cv.threshold(patch, 125, 255, cv.THRESH_BINARY)

        for template_img in files_template:
            img_template = cv.imread(folder_template + template_img)
            img_template = cv.cvtColor(img_template,cv.COLOR_BGR2GRAY)
            _, img_template = cv.threshold(img_template, 125, 255, cv.THRESH_BINARY)
            h, w = img_template.shape
            img_template = cv.resize(img_template,(h, w))
            # show_image('sample',img_template)
            corr = cv.matchTemplate(patch, img_template, cv.TM_CCOEFF_NORMED)
            corr = np.max(corr)
            if corr > maxi:
                maxi = corr
                poz = template_img[0]
            # print("Number ",template_img[0], " corr:", corr)
        if maxi < 0.4:
            poz = 0

        return poz
```

Task 3. Calcul scor

Pentru a calcula scorul ne folosim de tabla de joc (*main_game[][]* și *side_game[]*). Matricea *main_game* ne ajută să determinăm care este scorul pe care în primește jucătorul după mutarea făcută. Vectorul *side_game* este folosit împreună cu vectorul *score_tab* care păstrează scorul total al fiecărui player, acest scor total reprezintă și poziția pionului pe tablă și este folosit pentru a ști ce număr se află sub pion.

Folosim aceste date pentru a determina dacă numărul de sub pion *side_game[pawn]* este egal cu unul dintre numerele de pe piesa de domino *first/second_number*. Acest lucru se verifică pentru toți jucătorii.

```
score1 = main_game[piece1[0]][piece1[1]]
score2 = main_game[piece2[0]][piece2[1]]
current_score = 0

# Scorul din urma mutarii, daca piesa este dublă și punctajul se dublează
# Pentru prima jumatate a piesei
if score1 != 0:
    if first_number == second_number:
        score1 = score1 * 2
    # Daca un numar de pe piesa = pozitia unui jucator => +3 puncte
    for id, pawn in enumerate(score_tab):
        pawn = int(pawn)
        if side_game[pawn] == int(first_number) or side_game[pawn] == int(second_number):
            score_tab[id]+=3
            current_score+=3
    score_tab[current_player-1] += score1
    current_score += score1
```

Astfel, *current_score* reprezintă scorul obținut de jucătorul curent în runda respectivă.

Afișarea rezultatelor

Pentru a crea fișierele .txt necesare și a afișa rezultatele, folosim următorul cod:

```
# Scrierea în .txt a rezultatelor
line1 = str(piece1[0])+str(piece1[2])+" "+str(first_number)+"\n"
line2 = str(piece2[0])+str(piece2[2])+" "+str(second_number)+"\n"
line3 = str(current_score)
# Afisare in consola
print("Joc ",game_number," mutarea ",round_number,":\n",line1,line2,line3,"\n\n")

# Afisare in .txt
with open(folder_output + file[:-4]+'.txt', 'w') as f:
    f.write(line1)
    f.write(line2)
    f.write(line3)
```

Fiindcă am creat fiecare string înainte de scrierea în fișier ținând cont de formatul dorit, garantăm că .txt v-a arăta conform cerinței.