

Documentatie Tema 1 CAVA

Capatina Razvan Nicolae, grupa 352

Task 1

Decuparea Careului de Joc

Pentru a identifica pozitionarea piesei noi aparute trebuie sa avem o imagine restransa doar asupra tablei de joc. Acest lucru implica sa decupam din imaginea originala acest careu.

Urmatoarele transformari au fost aplicate in aceasta ordine unei imagini initiale:

- Conversia imaginii initiale din formatul BGR in grayscale.
- Aplicarea unui filtru median folosind un kernel de dimensiune 7×7 . Filtrul median contribuie la inlaturarea noise-ului existent in imagine.
- Aplicarea unui filtru Gaussian folosind un kernel de dimensiune 5×5 si o deviatie standard calculata din dimensiunea acestuia. Acest filtru ajuta la o mai buna identificare a muchiilor din imagine.
- Aplicarea operatorului morfologic de eroziune cu un kernel de dimensiune 5×5 si cu 2 iteratii. Aceasta transformare este utila pentru a elimina contururile inguste, cum ar fi cele ale informatiilor din dreapta tablei de joc legate despre numarul total de piese.
- Identificarea muchiilor folosind metoda Canny cu 2 praguri: 200 si 400.
- Aplicarea operatorului morfologic de dilatare cu 2 iteratii si un kernel de dimensiune 5×5 . Aceasta transformare evidentiaza contururile ramase dupa transformarile anterioare.

Exemplu al unei imagini rezultate in urma aplicarii transformarilor:

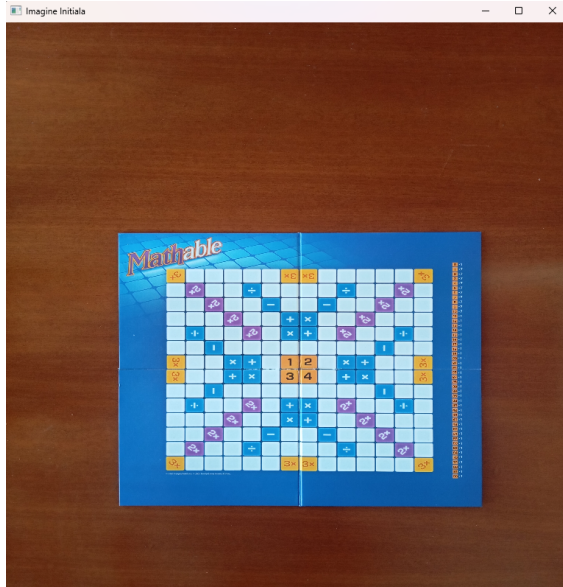


Figure 1: Imaginea Initiala

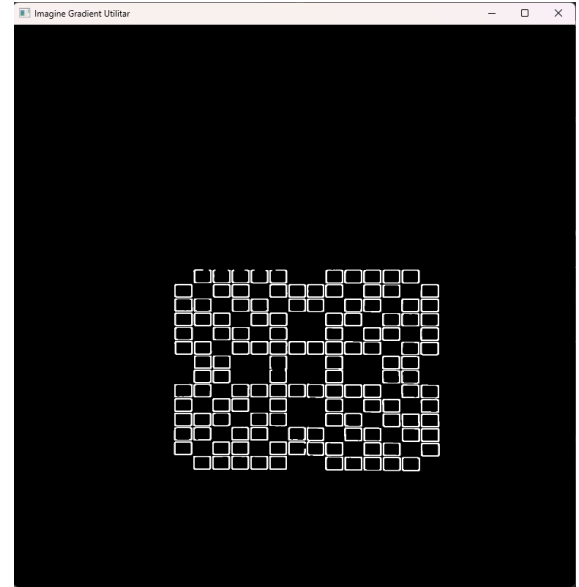


Figure 2: Imaginea Transformata

Acum putem identifica minimal bounding-box-ul care cuprinde toate contururile existente in imaginea transformata si astfel obtinem o decupare a tablei de joc din imaginea originala.

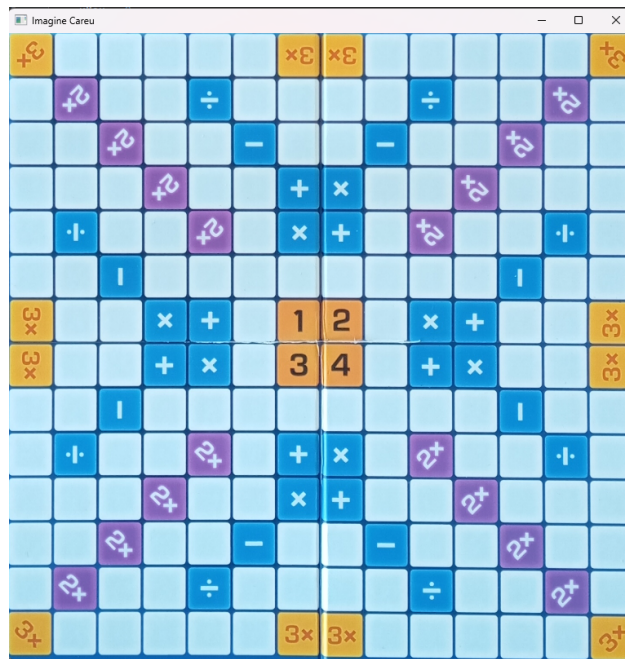


Figure 3: Imagine Careu Joc

Identificarea Celulei

Daca dimensiunea imaginii ce contine doar tabla de joc este (w, h) , atunci dimensiunea unei celule de joc ar fi $(\lceil w/14 \rceil, \lceil h/14 \rceil)$. Acest lucru ne permite sa traversam fiecare celula din careu.

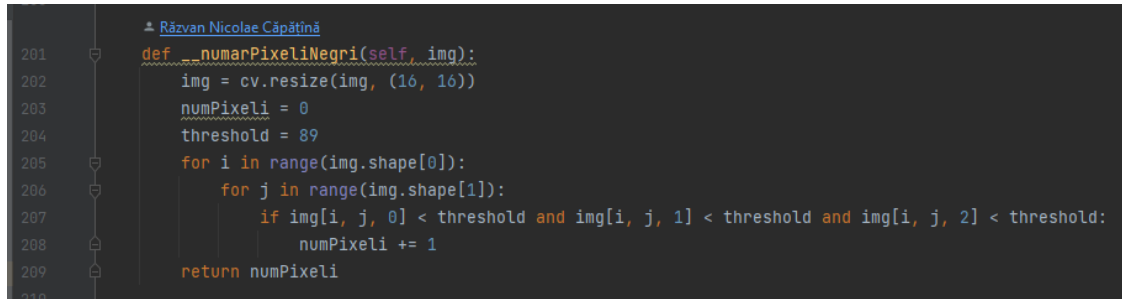
Pentru a identifica pozitia unde s-a plasat o piesa trebuie sa comparam rand pe rand fiecare celula din imaginea curenta cu celula de la aceeasi pozitie din imaginea anterioara si sa observam diferentele intre

acestea.

Nu putem sa ne uitam la contururile din celulele respective, deoarece o celula libera poate avea si contururi (cum sunt, de exemplu, celulele ce ofera bonus x2 si x3).

In schimb se poate observa faptul ca piesele sunt singurele care au nuante de negru (fiecare celula libera de pe tabla de joc nu are o nuanta de negru notabila). Acest lucru ne permite sa identificam daca o celula este in momentul de fata ocupata si daca in imaginea anterioara a fost libera.

Fragment de cod ce numara pentru o imagine cati pixeli cu o nuanta de negru exista:



```
201 def __numarPixeliNegri(self, img):
202     img = cv.resize(img, (16, 16))
203     numPixeli = 0
204     threshold = 89
205     for i in range(img.shape[0]):
206         for j in range(img.shape[1]):
207             if img[i, j, 0] < threshold and img[i, j, 1] < threshold and img[i, j, 2] < threshold:
208                 numPixeli += 1
209     return numPixeli
```

Figure 4: Cod Pixeli Nuante Negru

Are loc o redimensionare a imaginii la 16x16, apoi orice pixel care are toate canalele RGB sub un prag setat vor fi considerate ca fiind nuante de negru.

Aici au fost aplicate cateva optimizari ce au sporit acuratetea identificarii:

- Trecem rand pe rand doar prin celulele care inca erau libere in imaginea anterioara (tinem aceasta evidenta cu o structura de date in cod, de exemplu o matrice booleana). Si pentru fiecare celula nou ocupata marcam acest lucru in matrice, pentru ca in viitor sa nu o mai testam.
- Trecem doar prin celulele libere unde jucatorii pot plasa o piesa (stim faptul ca o piesa plasata trebuie sa completeze macar o ecuatie). Acest lucru duce, de asemenea, la limitarea spatiului de cautare.

Dupa ce am filtrat folosind optimizarile de mai sus celulele candidat unde s-a plasat ultima piesa o vom alege pe cea unde diferenta este maximala intre ea si corespondenta ei in imaginea anterioara (folosim norma L2).

Fragment de cod:

```

287     xPragProcent = 0.082
288     yPragProcent = 0.082
289
290     difMaxima = -1.0
291     iMaxim = -1
292     jMaxim = -1
293     imgCelulaMaximaFaraContur = None
294
295     yImgAnt = 0.0
296     yImgCrt = 0.0
297     for i in range(14):
298         xImgAnt = 0.0
299         xImgCrt = 0.0
300
301         for j in range(14):
302             celulaImgAnt = imgAnt[int(yImgAnt):int(yImgAnt + inaltimeCelulaImgAnt), int(xImgAnt):int(xImgAnt + latimeCelulaImgAnt)].copy()
303             celulaImgCrt = imgCrt[int(yImgCrt):int(yImgCrt + inaltimeCelulaImgCrt), int(xImgCrt):int(xImgCrt + latimeCelulaImgCrt)].copy()
304
305             celulaImgAnt = cv.resize(celulaImgAnt, (int(latimeCelulaImgCrt), int(inaltimeCelulaImgCrt)))
306             celulaImgAntFaraContur = celulaImgAnt[int(yPragProcent * inaltimeCelulaImgCrt):int((1.0 - yPragProcent)
307                 * inaltimeCelulaImgCrt), int(xPragProcent * latimeCelulaImgCrt):int((1.0 - xPragProcent) * latimeCelulaImgCrt)].copy()
308             celulaImgCrtFaraContur = celulaImgCrt[int(yPragProcent * inaltimeCelulaImgCrt):int((1.0 - yPragProcent)
309                 * inaltimeCelulaImgCrt), int(xPragProcent * latimeCelulaImgCrt):int((1.0 - xPragProcent) * latimeCelulaImgCrt)].copy()
310
311             if (self.__candidatCorectCelulaLibera(i, j) == False) or (filtrareStrictaCelule and
312                 (self.__numarPixeliNegri(celulaImgAntFaraContur) > 12 or self.__numarPixeliNegri(celulaImgCrtFaraContur) < 4)):
313                 xImgAnt += latimeCelulaImgAnt
314                 xImgCrt += latimeCelulaImgCrt
315                 continue
316
317             # difCurenta = np.mean((celulaImgCrtFaraContur - celulaImgAntFaraContur) ** 2)
318             # difCurenta = np.mean(np.abs(celulaImgCrtFaraContur - celulaImgAntFaraContur))
319             difCurenta = cv.norm(celulaImgAntFaraContur, celulaImgCrtFaraContur, cv.NORM_L2)
320
321             if difCurenta > difMaxima:
322                 difMaxima = difCurenta
323                 iMaxim = i
324                 jMaxim = j
325                 imgCelulaMaximaFaraContur = celulaImgCrtFaraContur.copy()
326
327             # Debug
328             xStangaSusSol = xImgCrt
329             yStangaSusSol = yImgCrt
330             latimeSol = latimeCelulaImgCrt
331             inaltimeSol = inaltimeCelulaImgCrt
332
333             xImgAnt += latimeCelulaImgAnt
334             xImgCrt += latimeCelulaImgCrt

```

Figure 5: Cod Identificare Celula

Se poate observa si faptul ca atunci cand decupam celula curenta la care ne aflam o facem eliminand o fractie din exteriorul acesteia (folosind variabilele $xPragProcent$ si $yPragProcent$). Acest lucru este util pentru rezolvarea task-urilor viitoare, neavand astfel in decupare si portiuni de nuante de negru care apartin marginilor celulei.

Task 2

Generarea unui Dataset de Sabloane

Dupa identificarea celulei unde s-a plasat ultima piesa trebuie sa identificam valoarea de pe aceasta.

Putem refolosi implementarea de la Task 1, alaturi de fisierele de antrenare, pentru a construi in mod automat un set de date cu sabloane, care apoi vor putea fi folosite in template matching.

- Luam pe rand fiecare imagine din fisierul de antrenare. Pentru fiecare dintre acestea stim solutia ei (stim celula care s-a modificat, dar si numarul de pe piesa).
- Decupam celula unde s-a plasat ultima piesa folosind implementarea de la Task-ul 1.

- Aplicam inca o data transformarile de la Task-ul 1, dar doar pe imaginea celulei decupate. Obtinem astfel un minimal bounding box care cuprinde doar numarul de pe piesa. Aceasta imagine rezultata are o oarecare invarianta la cum este plasata piesa pe tabla in timpul jocului. Aici se observa si importanta extragerii piesei folosind acele variabile $xPragProcent$ si $yPragProcent$. Extragerea interiorului piesei garanteaza ca nu vom obtine un minimal bounding box gresit (prea mare), datorat aparitiei unor nuante de negru pe marginile imaginii.
- Stiind eticheta acestei piese din fisierul text al antrenarii putem construi astfel un set de date pe care sa il folosim in viitor pentru template matching.
- Putem augmenta acest set de date, adaugand aceleasi piese, dar rotite cu -2 grade, -1 grad, 1 grad si respectiv 2 grade.

Exemplu de set de date obtinut pentru valoarea 2:

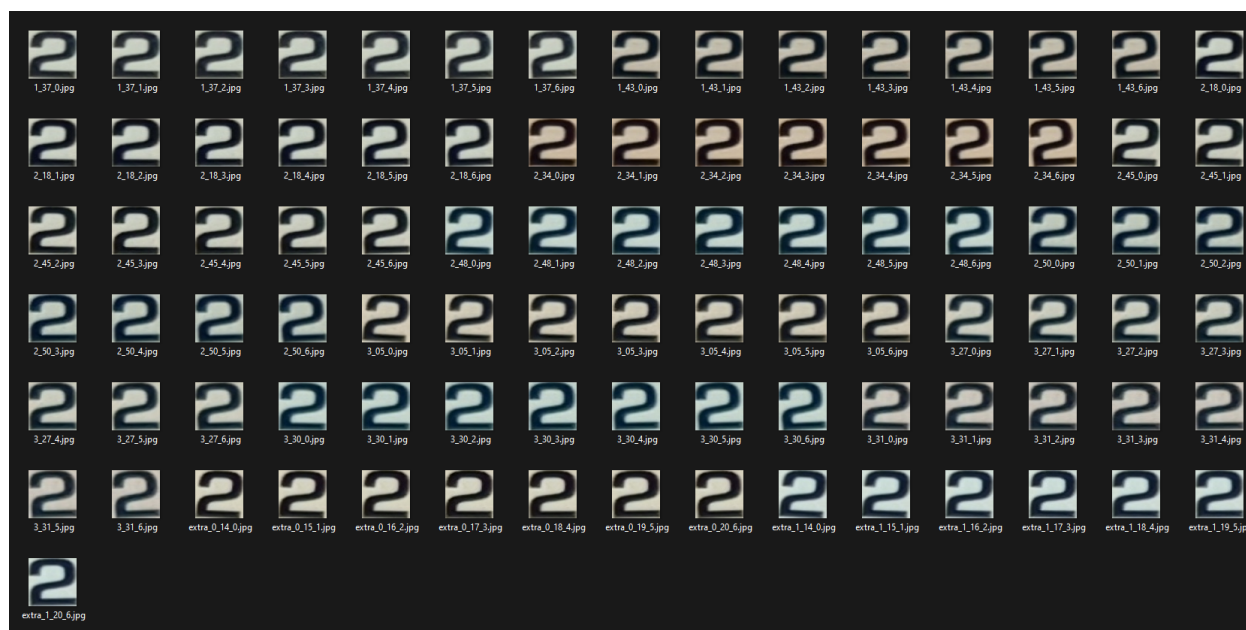


Figure 6: Set de Date pentru Valoarea 2

Potrivre de Sabloane

Utilizam setul de date generat anterior pentru a identifica valoarea de pe o piesa:

- Aplicam asupra piesei aceleasi transformari care au fost rulate pe piesele din fisierul de antrenare atunci cand am construit setul de date.
- Consideram eticheta solutie ca fiind cea pentru care avem media cat mai mica pentru distantele tuturor normelor L2 intre imaginile din setul de date cu eticheta respectiva si imaginea de test.

Exemplu implementare:

```

284
    Răzvan Nicolae Căpărină
285 def __distantaImagini(self, img, sablon):
286     #return np.mean((img - sablon) ** 2)
287     return cv.norm(img, sablon, cv.NORM_L2)
288     #return np.mean(np.abs(img - sablon))
289     #, valoareMaxima, _ = cv.minMaxLoc(cv.matchTemplate(img, sablon, cv.TM_CCOEFF_NORMED))
290     #return -valoareMaxima
291
292
    Răzvan Nicolae Căpărină
293 def evalueazaImagine(self, img, sabloaneAcceptabile: set):
294     imgPrelucrat = Utilitar.prelucreazaSablon(img)
295
296     etichetaSolutie = -1
297     distantaMinima = float('inf')
298
299     for eticheta, sabloane in self.colectiiSabloane.items():
300
301         if (eticheta not in self.pieseIncaFolosibile) or (eticheta not in sabloaneAcceptabile):
302             continue
303
304         distanceSablon = []
305         for sablon in sabloane:
306             distanceSablon.append(self.__distantaImagini(imgPrelucrat, sablon))
307
308         distantaCurenta = np.mean(distanceSablon)
309
310         if distantaCurenta < distantaMinima:
311             distantaMinima = distantaCurenta
312             etichetaSolutie = eticheta
313
314
315     self.pieseIncaFolosibile[etichetaSolutie] -= 1
316     if self.pieseIncaFolosibile[etichetaSolutie] == 0:
317         del self.pieseIncaFolosibile[etichetaSolutie]
318     return etichetaSolutie
319
320

```

Figure 7: Implementare Potrivire de Sabloane

Optimizari ce au imbunatatit acuratetea potrivirii sabloanelor:

- Nu testam o eticheta pentru care nu mai avem piese posibile pe care sa le plasam pe tabla de joc. Exista riscul sa obtinem media minimala a normelor L2 pentru acest label. Folosim structuri de date, precum un dictionar, pentru a tine evidenta etichetelor care mai pot fi asociate pieselor noi venite.
- Testam doar pentru etichetele care completeaza o ecuatie daca se afla pe pozitia respectiva. Micsoram astfel spatiul de cautare si imbunatam acuratetea. Aceasta optimizare se poate implementa folosind structuri de date precum matrici, alaturi de o implementare similara cu cea de la Task-ul 3.

Fragment de cod pentru implementarea ultimei optimizari mentionate:

```

227
228 def _identificareSingureleSabloaneAcceptabile(self, i : int, j : int):
229     sabloaneAcceptabile = set()
230
231     if i > 1 and self.numarDePeCelula[i - 1][j] != -1 and self.numarDePeCelula[i - 2][j] != -1:
232         sabloaneAcceptabile.add(self.numarDePeCelula[i - 1][j] + self.numarDePeCelula[i - 2][j])
233         sabloaneAcceptabile.add(self.numarDePeCelula[i - 1][j] * self.numarDePeCelula[i - 2][j])
234         sabloaneAcceptabile.add(abs(self.numarDePeCelula[i - 1][j] - self.numarDePeCelula[i - 2][j]))
235         if self.numarDePeCelula[i - 1][j] != 0 and self.numarDePeCelula[i - 2][j] % self.numarDePeCelula[i - 1][j] == 0:
236             sabloaneAcceptabile.add(self.numarDePeCelula[i - 2][j] // self.numarDePeCelula[i - 1][j])
237         if self.numarDePeCelula[i - 2][j] != 0 and self.numarDePeCelula[i - 1][j] % self.numarDePeCelula[i - 2][j] == 0:
238             sabloaneAcceptabile.add(self.numarDePeCelula[i - 1][j] // self.numarDePeCelula[i - 2][j])
239
240     if i + 2 < 14 and self.numarDePeCelula[i + 1][j] != -1 and self.numarDePeCelula[i + 2][j] != -1:
241         sabloaneAcceptabile.add(self.numarDePeCelula[i + 1][j] + self.numarDePeCelula[i + 2][j])
242         sabloaneAcceptabile.add(self.numarDePeCelula[i + 1][j] * self.numarDePeCelula[i + 2][j])
243         sabloaneAcceptabile.add(abs(self.numarDePeCelula[i + 1][j] - self.numarDePeCelula[i + 2][j]))
244         if self.numarDePeCelula[i + 1][j] != 0 and self.numarDePeCelula[i + 2][j] % self.numarDePeCelula[i + 1][j] == 0:
245             sabloaneAcceptabile.add(self.numarDePeCelula[i + 2][j] // self.numarDePeCelula[i + 1][j])
246         if self.numarDePeCelula[i + 2][j] != 0 and self.numarDePeCelula[i + 1][j] % self.numarDePeCelula[i + 2][j] == 0:
247             sabloaneAcceptabile.add(self.numarDePeCelula[i + 1][j] // self.numarDePeCelula[i + 2][j])
248
249     if j > 1 and self.numarDePeCelula[i][j - 1] != -1 and self.numarDePeCelula[i][j - 2] != -1:
250         sabloaneAcceptabile.add(self.numarDePeCelula[i][j - 1] + self.numarDePeCelula[i][j - 2])
251         sabloaneAcceptabile.add(self.numarDePeCelula[i][j - 1] * self.numarDePeCelula[i][j - 2])
252         sabloaneAcceptabile.add(abs(self.numarDePeCelula[i][j - 1] - self.numarDePeCelula[i][j - 2]))
253         if self.numarDePeCelula[i][j - 1] != 0 and self.numarDePeCelula[i][j - 2] % self.numarDePeCelula[i][j - 1] == 0:
254             sabloaneAcceptabile.add(self.numarDePeCelula[i][j - 2] // self.numarDePeCelula[i][j - 1])
255         if self.numarDePeCelula[i][j - 2] != 0 and self.numarDePeCelula[i][j - 1] % self.numarDePeCelula[i][j - 2] == 0:
256             sabloaneAcceptabile.add(self.numarDePeCelula[i][j - 1] // self.numarDePeCelula[i][j - 2])
257
258     if j + 2 < 14 and self.numarDePeCelula[i][j + 1] != -1 and self.numarDePeCelula[i][j + 2] != -1:
259         sabloaneAcceptabile.add(self.numarDePeCelula[i][j + 1] + self.numarDePeCelula[i][j + 2])
260         sabloaneAcceptabile.add(self.numarDePeCelula[i][j + 1] * self.numarDePeCelula[i][j + 2])
261         sabloaneAcceptabile.add(abs(self.numarDePeCelula[i][j + 1] - self.numarDePeCelula[i][j + 2]))
262         if self.numarDePeCelula[i][j + 1] != 0 and self.numarDePeCelula[i][j + 2] % self.numarDePeCelula[i][j + 1] == 0:
263             sabloaneAcceptabile.add(self.numarDePeCelula[i][j + 2] // self.numarDePeCelula[i][j + 1])
264         if self.numarDePeCelula[i][j + 2] != 0 and self.numarDePeCelula[i][j + 1] % self.numarDePeCelula[i][j + 2] == 0:
265             sabloaneAcceptabile.add(self.numarDePeCelula[i][j + 1] // self.numarDePeCelula[i][j + 2])
266
267     return sabloaneAcceptabile

```

Figure 8: Implementare Identificare Sabloane Acceptabile

Task 3

Implementare

Implementarea Task-ului 3 este una directa si se bazeaza pe rezultatele obtinute la Task-urile 1 si 2. Acuratetea task-urilor anterioare este vitala pentru a obtine rezultate bune la aceasta cerinta.

Am folosit 2 matrici de dimensiune 14x14 pentru a memora bonusul primit pentru fiecare celula in parte, dar si daca pe o anumita celula exista vreo constrangere de operatie.


```

34
35 self.PONDERI_SCOR_CELULA = [
36     [3, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 3],
37     [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1],
38     [1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1],
39     [1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1],
40     [1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1],
41     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
42     [3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3],
43     [3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3],
44     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
45     [1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1],
46     [1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1],
47     [1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1],
48     [1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1],
49     [3, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 3]
50 ]
51
52 self.CONSTRANGERI_CELULE = [
53     ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.'],
54     ['.', '.', '.', '.', '.', '/', '.', '.', '.', '/', '.', '.', '.', '.'],
55     ['.', '.', '.', '.', '.', '-', '.', '.', '-', '.', '.', '.', '.', '.'],
56     ['.', '.', '.', '.', '.', '+', '.', '.', '+', '.', '.', '.', '.', '.'],
57     ['.', '.', '/', '.', '.', '.', '.', '+', '+', '.', '.', '.', '/', '.', '.'],
58     ['.', '.', '-', '.', '.', '+', '+', '.', '.', '-', '.', '.', '.', '.', '.'],
59     ['.', '.', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+'],
60     ['.', '.', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+'],
61     ['.', '.', '-', '.', '.', '+', '+', '.', '.', '-', '.', '.', '.', '.', '.'],
62     ['.', '.', '/', '.', '.', '+', '+', '.', '.', '/', '.', '.', '.', '.', '.'],
63     ['.', '.', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+', '+'],
64     ['.', '.', '-', '.', '.', '-', '.', '.', '-', '.', '.', '.', '.', '.'],
65     ['.', '.', '/', '.', '.', '/', '.', '.', '/', '.', '.', '.', '.', '.'],
66     ['.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.'],
67 ]
68

```

Figure 9: Matrici

Fragment de cod implementare:


```

108
109  def calculeazaScorPiesaNova(self, iPiesa: int, jPiesa: int):
110      valoarePiesa = self.PONDERI_SCOR_CELULA[iPiesa][jPiesa] * self.numarDePeCelula[iPiesa][jPiesa]
111      scorTotalPiesa = 0
112
113      print('Pondere Piesa:', valoarePiesa)
114
115      # Linie Stanga
116      if jPiesa > 1 and self.numarDePeCelula[iPiesa][jPiesa - 1] != -1 and self.numarDePeCelula[iPiesa][jPiesa - 2] != -1:
117          if (self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == ' ' or self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == '+')\
118              and self.numarDePeCelula[iPiesa][jPiesa - 1] + self.numarDePeCelula[iPiesa][jPiesa - 2] == self.numarDePeCelula[iPiesa][jPiesa]:
119              scorTotalPiesa += valoarePiesa
120          elif (self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == ' ' or self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == '*')\
121              and self.numarDePeCelula[iPiesa][jPiesa - 1] * self.numarDePeCelula[iPiesa][jPiesa - 2] == self.numarDePeCelula[iPiesa][jPiesa]:
122              scorTotalPiesa += valoarePiesa
123          elif (self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == ' ' or self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == '-')\
124              and self.numarDePeCelula[iPiesa][jPiesa - 1] - self.numarDePeCelula[iPiesa][jPiesa - 2] == self.numarDePeCelula[iPiesa][jPiesa]:
125              scorTotalPiesa += valoarePiesa
126          elif (self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == ' ' or self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == '-')\
127              and self.numarDePeCelula[iPiesa][jPiesa - 2] - self.numarDePeCelula[iPiesa][jPiesa - 1] == self.numarDePeCelula[iPiesa][jPiesa]:
128              scorTotalPiesa += valoarePiesa
129
130          elif (self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == ' ' or self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == '/')\
131              and self.numarDePeCelula[iPiesa][jPiesa - 1] != 0 and self.numarDePeCelula[iPiesa][jPiesa - 2] % self.numarDePeCelula[iPiesa][jPiesa - 1] == 0\
132              and self.numarDePeCelula[iPiesa][jPiesa - 2] // self.numarDePeCelula[iPiesa][jPiesa - 1] == self.numarDePeCelula[iPiesa][jPiesa]:
133              scorTotalPiesa += valoarePiesa
134          elif (self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == ' ' or self.CONSTRANGERI_CELULE[iPiesa][jPiesa] == '/')\
135              and self.numarDePeCelula[iPiesa][jPiesa - 2] != 0 and self.numarDePeCelula[iPiesa][jPiesa - 1] % self.numarDePeCelula[iPiesa][jPiesa - 2] == 0\
136              and self.numarDePeCelula[iPiesa][jPiesa - 1] // self.numarDePeCelula[iPiesa][jPiesa - 2] == self.numarDePeCelula[iPiesa][jPiesa]:
137              scorTotalPiesa += valoarePiesa
138      print('Scor Piesa:', scorTotalPiesa)
139

```

Figure 10: Implementare Task 3

Implementarea de mai sus este doar pentru a verifica daca o ecuatia s-a format la stanga de pozitia curenta si pentru a calcula scorul obtinut in urma acesteia. In mod analog se pot implementa si verificarile pentru dreapta, sus si respectiv jos.

Un detaliu in cadrul implementarii ar fi asigurarea existentei unor verificari pentru a nu accesa celule din afara tablei de joc.