

# Documentație Proiect

## *Double Double Dominos*

*Disciplina Concepte și Aplicații în Vedere Artificială*

Universitatea din București  
Facultatea de Matematică și Informatică  
Iulia-Georgiana Talpalariu  
Grupa 334

Decembrie 2023

## 1 Introducere

Spre rezolvarea taskurilor de vedere artificială de la tema *Double Double Dominos*, propun o soluție documentată și detaliată în raportul de față. Soluția mea utilizează noțiunile dobândite la disciplina *Concepte și Aplicații în Vedere Artificială* și este implementată în limbajul Python cu ajutorul librăriilor `numpy` și `opencv`.

## 2 Prezentarea datelor

Datele de antrenare, cât și cele de test se prezintă sub forma a 100 de imagini (3072 x 4080 pixeli), format `jpg`, care reprezintă primele 20 de mutări din 5 jocuri diferite de *Double Double Dominos*, acompaniate de un fișier text cu 20 de linii, care face corespondența *titlu imagine - jucătorul care a făcut mutarea surprinsă în imagine*. De menționat că jocul poate fi jucat și în mai mulți jucători, varianta de față fiind una cu constrângerea că există doar 2 jucători, numiți mereu *player1* și *player2*.

### 2.1 Tabla de joc

Tabla de joc nu se schimbă de-a lungul jocului și este alcătuită dintr-o margine formată din domino-uri care determină traseul de scor (pe acest traseu se mișcă pionii) și careul de joc pe care se așeză piesele de domino.

### 2.2 Careul

Careul reprezintă centrul tablei de joc și este alcătuit din 15 linii, numerotate 1-15 și 15 coloane, adnotate cu literele A-O, fiecare poziție ocupată de o jumătate de domino fiind o combinație linie-coloană (exemplu: 7H, 8I, etc)

## 3 Prezentarea taskurilor

Proiectul cuprinde rezolvarea celor 3 taskuri:

**Task 1** Detectia pozitiei piesei pe careul de joc.

**Task 2** Detectia numerelor de pe piesele de domino.

**Task 3** Calculul corect al scorului primit de fiecare jucător din așezarea unei piese.

## 4 Metrii de evaluare

Performanța algoritmului va fi analizată cu un punctaj, la fiecare imagine primind punctajul dacă se determină corect poziția ambelor capete (0.05p - Task 1), numerele de pe ambele jumătăți de domino (0.02p - Task 2) și punctajul asociat mutării (0.02p - Task 3).

## 5 Metode abordate

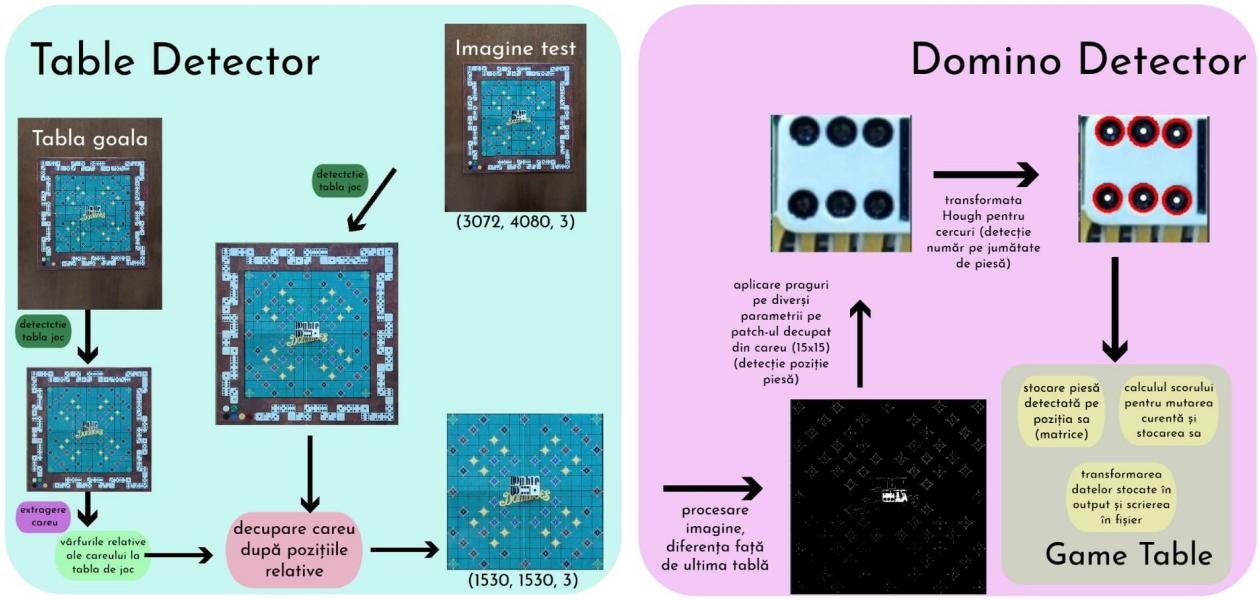


Figura 1: Viziune de ansamblu asupra soluției oferite de mine

## 5.1 Augmentarea datelor

Pentru a testa algoritmul meu pe imagini cu diverse luminoziăți, am folosit, din modulul `opencv`, funcția `cv.convertScaleAbs(image, alpha, beta)`, setând `alpha=1` și `beta` luând valori: -30, -20, -10, 10, 20. Pentru toate aceste cazuri, algoritmul meu greșește cel mult două piese. În schimb, algoritmul pare sensibil la schimbări de saturatie, cu un `alpha=0.9` și `beta=30`, greșește 40% din piese (probabil din cauza filtrării HSV, unde filtrez în dimensiunea saturatiei). Pentru saturatie mai mare (ex. `alpha 1.1` și `beta=-30`), deși întunecate imaginile sunt mai bine detectate și procesate cu ajutorul soluției propuse de mine.

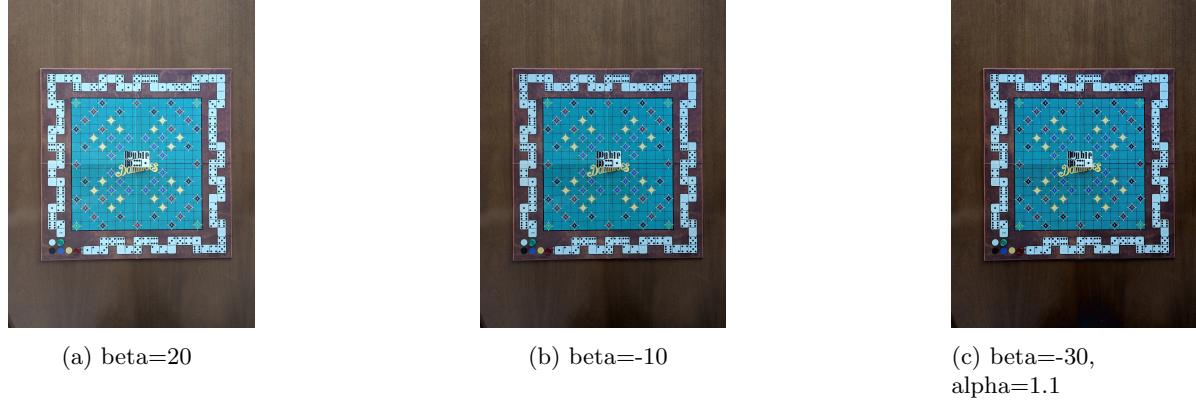


Figura 2: Augmentare prin schimbarea luminozității

## 5.2 Rezolvarea taskului 1

### 5.2.1 Detectia tablei de joc

Pentru detectia tablei de joc, mai intai aplic o preprocesare pe imagine, prin tarierea a 10 % superioare si 10 % inferioare din inaltimea imaginii, deoarece am observat ca astfel filtrele aplicate apoi fac distinctia mai bine tabla-masa mai ales pentru variajii de luminozitate. Ulterior, aplic o filtrare in spatiul HSV intre (0, 0, 0) si (45, 255, 255). Pasterez masca obtinuta si lucrez apoi cu ea. Pentru a scapa de zgomot aplic o combinatie (ponderi 1.2, -0.8) de filtru median si gaussian (ambele cu kernel de dimensiune 21), similara cu cea folosita la laborator. Aplic un prag (100) pentru binarizare, apoi operatia de deschidere (eroziune cu kernel 15 + dilatare cu kernel 5). Ulterior, pentru conturarea marginilor, aplic filtrul Canny cu pragurile 200 si, respectiv, 400.

Apoi, gasesc contururile si aplic metoda de gasire a celui mai mare dreptunghi implementata la laborator. Repet aceasta operatie la fiecare imagine de test ( fiecare mutare ).

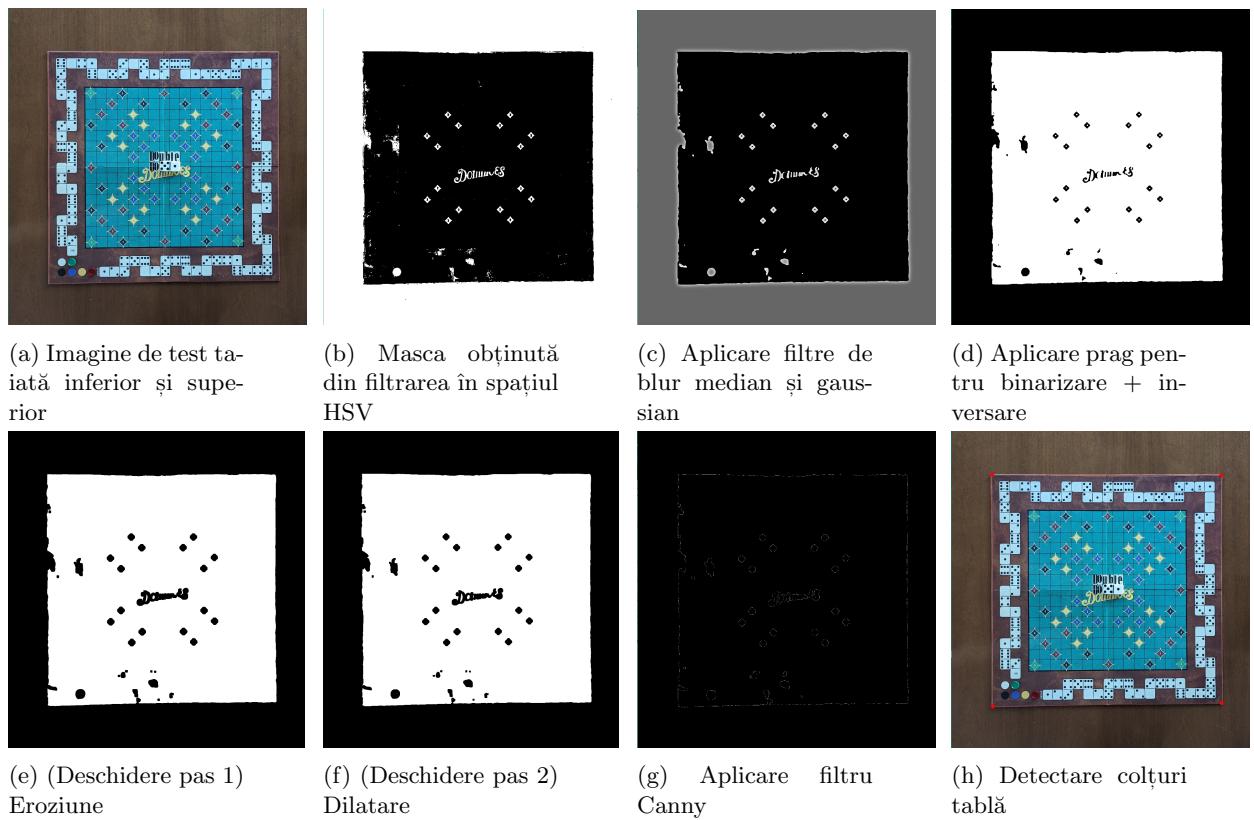


Figura 3: Detectare tablă de joc

### 5.2.2 Detectia careului

Pentru detectia careului, am păstrat într-o variabilă pozitia colturilor careului relativ la pozitia tablei, extrase cu ajutorul imaginii auxiliare 01.jpg (tabla goală).

Extragerea careului urmează etape similare celor de la pasul anterior. Inițial obțin o mască prin filtrarea în spațiul HSV în intervalul  $(60, 150, 40) - (255, 255, 255)$ , apoi aplic aceeași combinație de filtre de blur ca la detectia tablei. Pun un prag de 30 pentru binarizare, aplic eroziune cu kernel de dimensiune 3 și apoi aplic același filtru Canny ca mai devreme (5.2.1).

Pentru imaginile de test, fac doar detectia tablei de joc, iar careul îl extrag cu ajutorul acestor coordinate (vârfurile careului) detectate inițial pe tabla goală, deoarece imaginile nu sunt în perspectivă, pozitia careului relativ la tabla de joc nu se modifică, scena fotografiilor nemodificându-se. Pentru a mă asigura de corectitudinea pasului următor, la careul detectat automat de algoritm, adaug o margine de siguranță de 7 pixeli, observând că piesele de pe liniile și coloane de pe margine se pot afla chiar pe conturul negru al tablei.

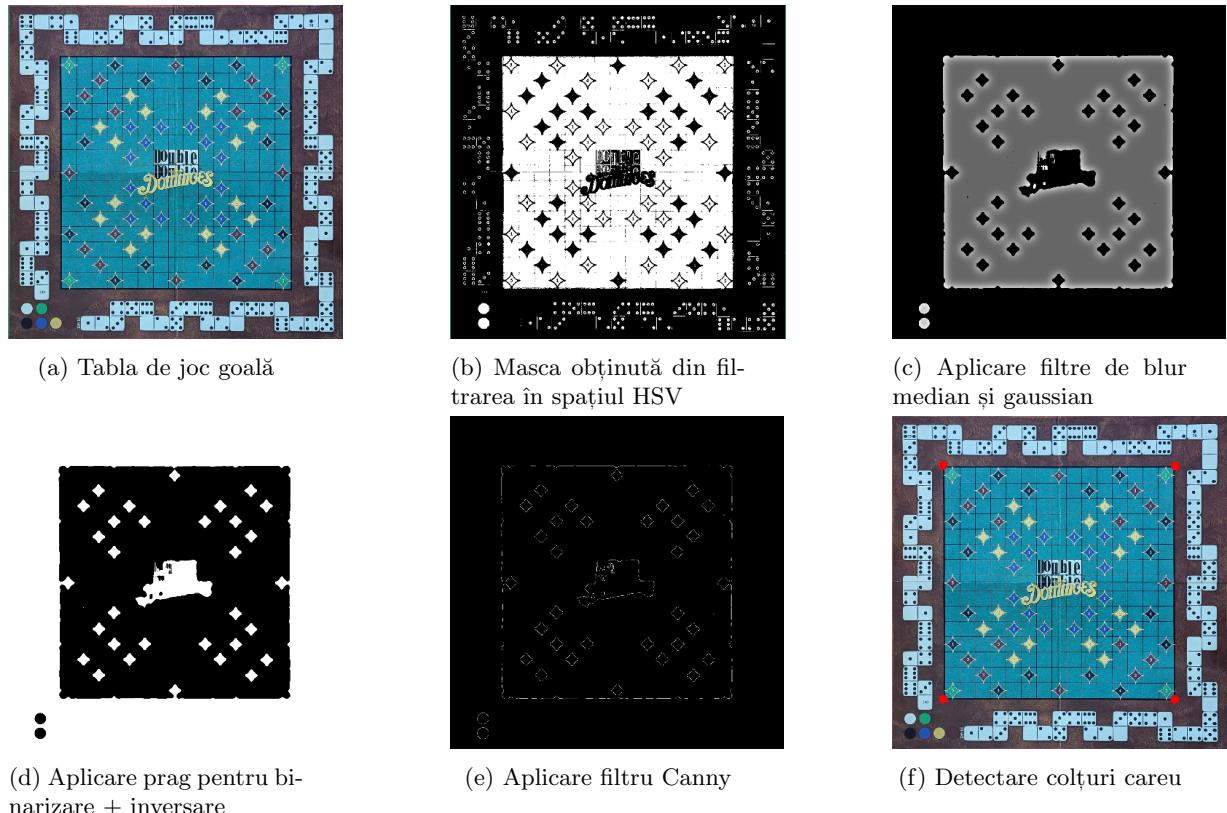


Figura 4: Detectare careu de joc

### 5.2.3 Detectia pozitiilor ocupate

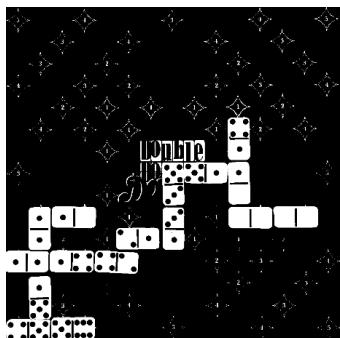
În acest pas, am deja careul tăiat și redimensionat la (1530, 1530, 3). Împărțirea acestuia în patch-uri care vor fi testate dacă sunt ocupate sau nu de jumătăți de piesă este simplă având în vedere că știm că tabla are 15 linii și 15 coloane. Cum avem dimensiunea tablei de 1530, fiecare patch va avea (102, 102, 3). Pentru a detecta corect piesele mișcate, adaug și o margine de siguranță de 1 pixel în toate cele 4 părți (sus, jos, stânga, dreapta).

Imaginea curentă și imaginea ultimei table (tabla goală în cazul primei imagini din fiecare joc) trec printr-o filtrare în spațiul HSV în intervalul (55, 0, 215)-(255, 112, 255), obținând măști care vor fi folosite la calculul indicatorilor de mai jos. Pentru detectia pozitiilor ocupate folosesc mai mulți indicatori:

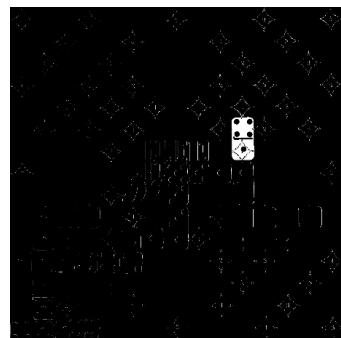
- 1 Media de albastru pe patch-ul extras. [ pondere 0.3 ]
- 2 Diferența pe canalul de roșu între tabla curentă și ultima tablă (tabla goală în cazul primei table din fiecare joc).[ pondere 0.3 ]
- 3 Media pe imaginea diferență între 2 măști obținute prin filtrarea în spațiul HSV a celor 2 table (curentă și ultima tablă). [ pondere 0.4 ]

$$\text{coeficient} = (<\text{indicator1}> - 170) / 170 * 0.3 + (<\text{indicator2}> - 25) / 25 * 0.3 + (<\text{indicator3}> - 45) / 45 * 0.4$$

Valorile din acest coeficient sunt găsite empiric prin observarea unui prag general între patch-urile *piese* și *non-piese*.



(a) Mască obținută prin filtrarea în spațiul HSV  
în intervalul (55, 0, 215) - (255, 112, 255)



(b) Diferența cu imaginea anterioară (diferență de măști)

Figura 5: Detectie pozitie piesă

Obțin astfel un coeficient care ar trebui să fie *negativ* pentru patch-urile care nu s-au schimbat de la ultima mutare și *pozitiv* pentru patch-urile pe care s-a adaugat piesă. Evident, coeficientul este obținut prin încercări și nu este exact, poate da fals pozitiv pentru scrisul de pe centrul tablei (cel cu alb - primul Double). De aceea, dacă obțin un coeficient pozitiv, dar totuși cu indicatorul 3 mai mic decât 100, (între pozițiile 7G- 7I), testez cu template matching dacă nu cumva patchu-ul respectiv este exact acel scris. În detectia pozitiilor ocupate, **nu** am utilizat informația că prima piesă este mereu așezată la poziția din centru (8H).

### 5.3 Rezolvarea taskului 2

#### 5.3.1 Determinarea numerelor de pe jumătăți de piesă

Pentru rezolvarea acestui task, am folosit transformata Hough pentru cercuri. Pentru a pregăti patch-ul cu jumătate de piesă detectat de taskul anterior, aplic un filtru bilateral, care blurează imaginea (pentru a scădea zgomotul), dar păstrează marginile destul de evidente pentru detecția cercurilor. Prin încercări succesive am găsit parametrii pentru transformata Hough. Am setat distanța minimă dintre 2 detecții de cercuri ca fiind 7, iar intervalul razelor între 9 și 14, majoritatea cercurilor detectate pe piese având raza 11-13, dar am lăsat o marjă pentru a capta și piesele de la marginea tablei care uneori au cercurile în perspectivă/ alte piese la care din cauza luminii cercurile se văd mai mici. Detecția este robustă la rotația ușoară a pieselor (Figura 5e). Nu am utilizat poziția cercurilor în a determina numărul lor (poziția specifică a unei configurații).

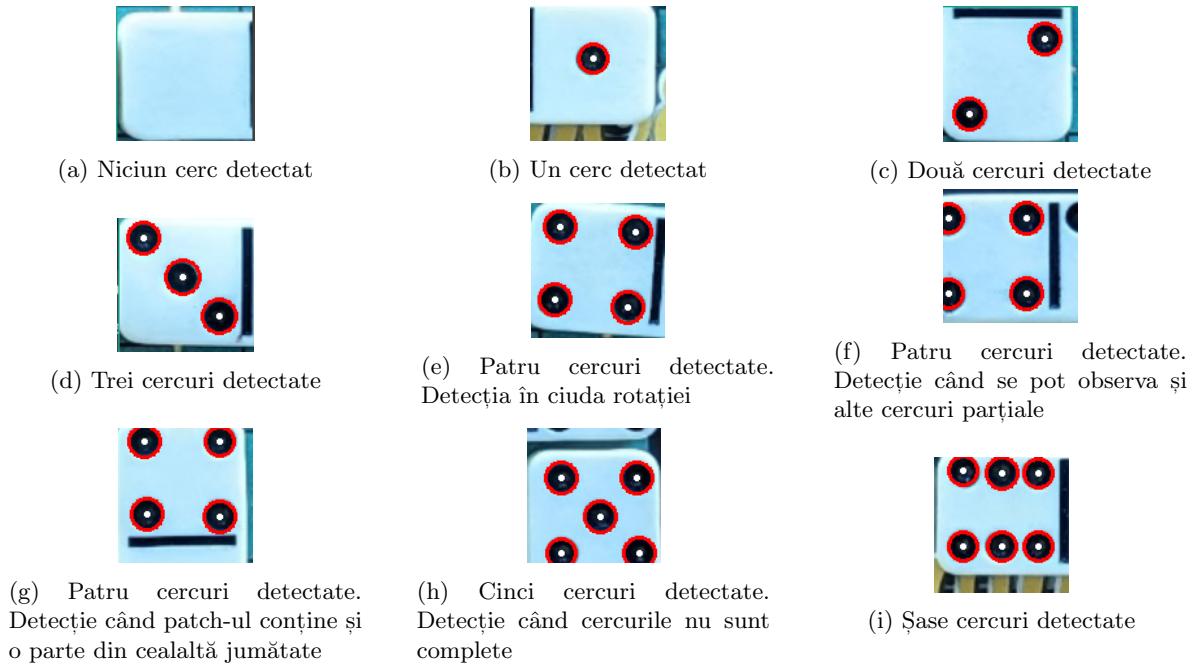


Figura 6: Diverse patch-uri cu cercurile detectate (conturul roșu și centrul alb)

## 5.4 Rezolvarea taskului 3

### 5.4.1 Calcularea scorului

Pentru calcularea scorului, am realizat clasa `GameTable`, care nu se ocupă de nimic legat de vede-rea artificială, ci doar are tabla de joc cu piesele stocate și calculează, la fiecare pas, scorul. De menționat că fiecare joc are un singur obiect de tip `GameTable`, pe când fiecarei mutări îi este alocat câte un obiect de tip `TableDector` și `DominoDetector`. O altă precizare este că stocarea sub formă de matrice a careului are în vedere poziții pentru linii și coloane 0-14, iar funcțiile care fac conversia între reprezentarea internă și reprezentarea externă (pentru scris în fișier) sunt `line_to_actual_line(line)` și `column_to_actual_column(column)`. Am hardcodat punctele de pe un sfert de careu de joc, iar pe celelalte le obțin la runtinme, prin luarea valorii de pe poziția simetrică față de x și sau y (funcția `position_to_points(position)`). Am hardcodat și traseul de scor și luarea numărului de pe poziția pionului curent se poate face cu ajutorul funcției `get_number_on_score_trail(player_current)`. Outputul poate fi obținut și totodată scris în fișier cu ajutorul funcției `get_output_current_move(self, added_positions, write, file_to_write)`.

## 6 Concluzii

Soluția oferită de mine recunoaște corect, atât poziția cât și numărul de pe piesele de domino pentru toate imaginile de antrenare și `fake_test`. Pentru variații mici de luminozitate, soluția mea oferă rezultate suficient de bune. Îmbunătățiri pot fi aduse la detecția pieselor după contur, detecția liniei de pe mijlocul domino-ului pentru tăierea jumătății de piesă, etc.