UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

TEHNOLOGIA INFORMAȚIEI

Proiect de dimplomă

AGENT INTELIGENT PENTRU JOCUL ȚINTAR

Absolvent
Preda Alexandru-Florin

Coordonatori științifici:
Conf. dr. Bogdan Alexe
Asist. drd. Alexandra Diaconu

București, iunie 2023

Rezumat

Țintarul, cunoscut și sub numele de "Nine men's morris", este un joc de strategie în care doi jucători își plasează piesele pe tablă cu scopul de a forma linii a câte trei piese și de a captura piesele adversarului.

Obiectivul acestei lucrări este de a oferi pasionaților de țintar o variantă digitală prin care să învețe și să avanseze în strategiile jocului. Pentru a atinge acest obiectiv am implementat o interfață grafică unde utilizatorul poate selecta tipul de joc (împotriva unui alt utilizator, împotriva unui agent inteligent sau să vizioneze un joc între doi agenți inteligenți) și poate schimba parametrii agentului inteligent (algoritmul folosit, adâncimea acestuia și euristica folosită). De asemenea, pentru utilizatorii ce vor să joace acest joc pe o tablă reală, există functionalitatea de sincronizare cu ajutorul imaginii redate de o cameră video. Pentru îndeplinirea obiectivului, lucrarea este alcătuită din patru componente realizate în Python:

- Interfața grafică realizată cu ajutorul PySimpleGUI pentru selectarea opțiunilor agentului inteligent și pentru selectarea tipului de joc
- Interfața grafică realizată cu ajutorul PyGame pentru simularea jocului
- Realizarea agentului inteligent ce constă în algoritmul bazat pe paradigma învățării prin recompense, min-max sau varianta accelerată cu alpha-beta, împreună cu opt tipuri de euristici
- Realizarea sincronizării tablei reale cu cea din PyGame:
  - Alinierea imaginii redate de camera video folosind descriptori SIFT
  - Extragerea de patch-uri ce conțin pozițiile pieselor
  - Realizarea unui clasificator ce prezice ce fel de piesă există sau nu într-un patch folosind descriptori ResNet-18 și un model LinearSVC
  - Recunoașterea configurației curente pe tabla de joc

Lucrarea se concentrează pe prezentarea celor patru componente și îmbinarea lor în aplicația finală, precum și pe prezentarea rezultatelor experimental obținute.

Abstract

Nine men's morris is a strategy game in which two players place their pieces on the board in order to form lines of three pieces and capture their opponent's pieces.

The goal of this paper is to provide a digital version for the board game enthusiast to learn and advance their strategies. To achieve this goal we have implemented a graphical interface where the user can select the type of game (against another user, against a smart agent or to watch a game between two smart agents) and change the parameters of the smart agent (the algorithm used, its depth and the the heuristics used). Also, for users who want to play this game on a real board, there is the functionality of synchronization using the image captured by a video camera. To achieve the goal, the paper consists of four components made in Python:

- Graphical user interface realized using PySimpleGUI for selecting the options of the intelligent agent and for selecting the type of game
- Graphical user interface made with PyGame for game simulation
- Implementation of the intelligent agent consisting of the Reinforcement Learning, min-max or alpha-beta algorithm, along with eight types of heuristics
- Syncronization of the real board with the PyGame board:
  - Alignment of the camera image using SIFT descriptors
  - Patch extraction containing piece positions
  - Creating a classifier that predicts what kind of exists or not in a patch using restnet18 descriptors and LinearSVC model
  - Extract table configuration

The paper focuses on the presentation of the four components and their connection in the final application, as well as the presentation of the experimental results obtained.

Cuprins

Listă de figuri

Listă de tabele

Listă de ecuații

Capitolul 1

Introducere

Inteligența artificială reprezintă un domeniu vast și de actualitate, cu o relevanță semnificativă în societatea contemporană, regăsindu-se în numeroase aspecte din viața cotidiană. Prezența constantă a inteligenței artificiale subliniază importanța sa primordială în modelarea vieții noastre moderne, pe măsură ce apare în industrii, în mediul academic, în domeniul sănătății, în economie și nu numai. Raza sa de acțiune se extinde în diverse ramuri, facilitând progresele, sporind procesele decizionale, optimizând alocarea resurselor, permițând automatizarea și revoluționând modul în care interacționăm cu tehnologia. Având în vedere implicațiile sale ample și potențialul de care dă dovadă, studiul și înțelegerea inteligenței artificiale capătă o mare importanță, pe măsură ce continuă să remodeleze lumea în care trăim.

Inteligența Artificială în jocuri a reprezentat o preocupare majoră și o inovație încă de la începuturile sale, iar în ultimii zece ani a cunoscut o creștere și un progres semnificativ. Acest lucru se datorează în mare parte includerii sale în jocurile video, domeniu care reprezintă o parte importantă din toate cercetările actuale [7]. Inteligența artificială implementată în majoritatea jocurilor contemporane, îndeplinește trei cerințe fundamentale: permite mișcarea personajului, ia decizii cu privire la deplasare și se ocupă de gândirea tactică sau strategică [5].

Jocul "Țintar" este un joc strategic cu radăcini istorice puternice, datând încă din 1400 î.Hr., având diferite denumiri, configurații ale tablei și reguli care au evoluat de-a lungul timpului. Acest joc își are originile în antichitate și se presupune că a fost dezvoltat în Egipt.

1.1 Scopul și obiectivele lucrării

Scopul acestei lucrări de licență este de a oferi utilizatorilor o soluție pentru a juca "Țintar" și a-și dezvolta aptitudinile cu ajutorul unui agent inteligent, integrat într-o interfață grafică, care să ofere o experiență interactivă.

Obiectivele principale au fost implementarea a doi algoritmi de inteligență artificială

pentru a juca împotriva utilizatorului: min-max(simplu și accelerat cu alpha-beta) și un algoritm care utilizează paradigma învățării prin recompense. O componentă unică a acestei aplicații este regăsirea configurației de joc, folosind componenta de vedere artificială, prin alinerea tablei reale de joc cu o imagine de referință, astfel utilizatorul poate interacționa cu cei doi algoritmi printr-o metodă mai realistă.

1.2 Structura proiectului de diplomă

În capitolele ce urmează sunt prezentate noțiunile teoretice ce stau la baza aplicației și cum au fost folosite în implementarea lor.

În Capitolul 2 sunt prezentate regulile și modalitatea de desfășurare al jocului. De asemenea, sunt enumerate cateva strategii simple pe care orice începator ar trebui să le cunoască pentru a avansa.

În Capitolul 3 este explicat teoretic cum sunt calculați descriptorii SIFT și cum au fost folositi pentru alinierea imaginii tablei, după care este descris modelul de clasificare care a fost folosit pentru extragerea configurației tablei.

În Capitolul 4 este prezentat modul în care algoritmul min-max extrage cea mai bună stare viitoare, îmbunatățirile algoritmului alpha-beta și euristicile folosite de aceștia. În acest capitol se regăseste și explicarea algoritmului bazat pe paradigma învățării prin recompense, arhitectura, setul de antrenare și evaluare, și cum a fost integrat în aplicație.

În utimul capitol sunt prezentate rezultatele experimentale pentru toate componentele aplicației. Experimentele regăsite sunt efectuate asupra modelului de clasificare, diferențele de performanta ale euristicilor, cât și cea mai bună euristică și performanța rețelei in comparație cu min-max, alpha-beta și un jucător incepător de ”Țintar”.

Capitolul 2

Jocul Țintar

Țintar este unul dintre cele mai vechi jocuri încă practicate în prezent. Acesta are numeroase variații precum: numărul diferențiat de piese al jucătorilor, colțurile pătratelor sunt legate cu linii diagonale, ș.a. În cazul meu, tabla de joc după care au fost definite regulile este cea modernă, adică formată din trei pătrate concentrice între care centrul laturilor acestora sunt conectate printr-o linie.

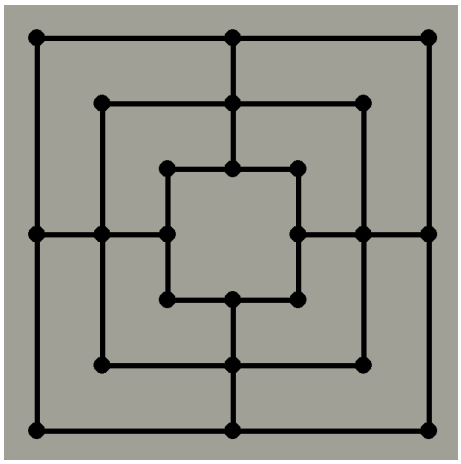

Figura 2.1: *Tabla de joc.*

2.1 Reguli și desfășurare

Pentru a începe un joc avem nevoie de o tablă de joc precum cea din Figura 2.1, 2 jucători, 9 piese de culoare portocalie și 9 piese de culoare verde, iar la început tabla de joc trebuie să fie goală.

Scopul principal al jocului este de a captura piesele adversarului astfel încât acesta să ramană cu mai putin de trei piese sau blocarea pieselor acestuia în așa fel încât să nu mai poată avea nici o mutare validă. Dacă jucătorul reușește să își ducă adversarul în una dintre cele două stări finale, acesta este învingător.

Pentru a îndepărta o piesa a adversarului, jucatorul trebuie să formeze o moară. O moară este formată dacă după mutarea jucătorului se realizează o configurație de 3 piese pe linie sau coloană. În Figura 2.2 se pot observa două configurații valide de mori încercuite cu culoarea verde și o configurație invalidă de moară încercuită cu culoarea roșu.

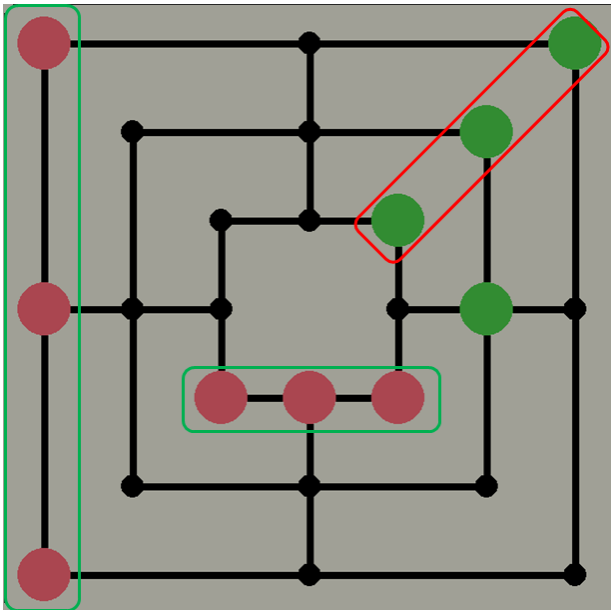

Figura 2.2: *Exemplu moară validă și invalidă*

Jocul începe prin împărțirea pieselor jucătorilor, astfel încât fiecare dintre aceștia să aibă 9 piese de aceeasi culoare. După împărțirea pieselor, jocul se desfășoară în două faze:

2.1.1 Faza 1: Punerea pieselor

În această fază, jucătorii își poziționează pe rând piesele pe tablă. Aceștia au voie să își pună piesele în oricare poziție validă și neocupată de pe tablă.

O poziție este validă dacă piesa jucătorului acoperă unul dintre cercurile negre din intersectiile liniilor de pe tablă.

În această etapă a jocului se pot forma mori. În cazul în care, după mutarea unuia dintre jucători, acesta formează o moară, el trebuie să îndepărteze una din oricare piesă de pe tablă a celuilalt jucător.

În Figura 2.3, se observa un exemplu de mutare în care jucătorul cu piese portocalii a format o moară. Configurația este încercuită cu verde, iar piesa jucătorului verde marcată cu un X roșu, este piesa pe care jucătorul ce a format moara decide să o îndepărteze. O dată cu îndepărtarea piesei, jucătorul își termină mutarea și este rezultată următoarea stare a jocului.

De asemenea, dacă un jucător formează 2 mori într-o singură mutare, acesta va îndepărta o singură piesă a adversarului.

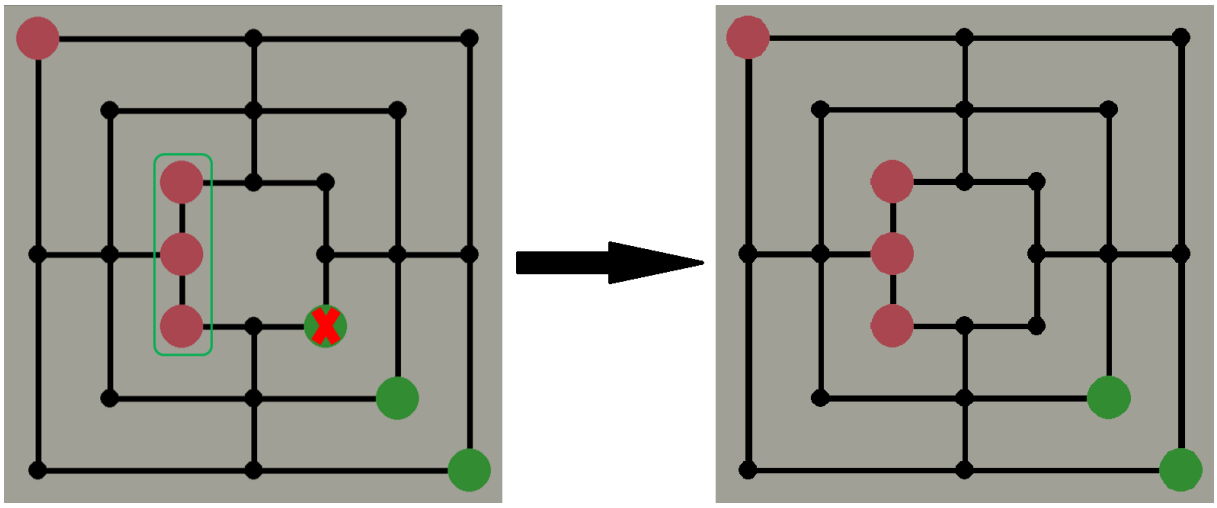

Figura 2.3: *Exemplu moară validă și îndepărtare piesă.*

2.1.2 Faza 2: Mutarea pieselor

După plasarea celor 9 piese pe tablă de către ambii jucători, jocul intră în faza a doua. În această etapă, jucătorii nu mai poziționează piesele, ci le mută pe cele deja plasate pe tablă, până când jocul atinge o stare finală.

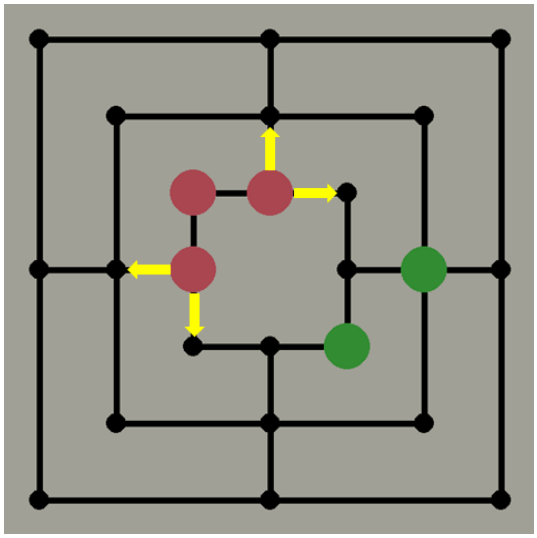

Figura 2.4: *Exemplu mutări valide din faza 2.*

Figura 2.4 ilustrează că o mutare este considerată validă atunci când piesa este mutată într-o poziție imediat adiacentă. Similar fazei precedente, jucătorii trebuie să formeze mori pentru a elimina piesele adversarilor, până când jocul ajunge în una dintre cele două stări finale posibile.

În Figura 2.5 se observă o mutare efectuată de jucătorul cu piesele portocalii în care piesa mutată de acesta formează moara încercuită cu verde. După formarea morii îndepărtează piesa marcată cu X-ul roșu. După îndepărtarea piesei, adversarul trebuie să mute la rândul lui, una dintre piesele verzi rămase pe tablă.

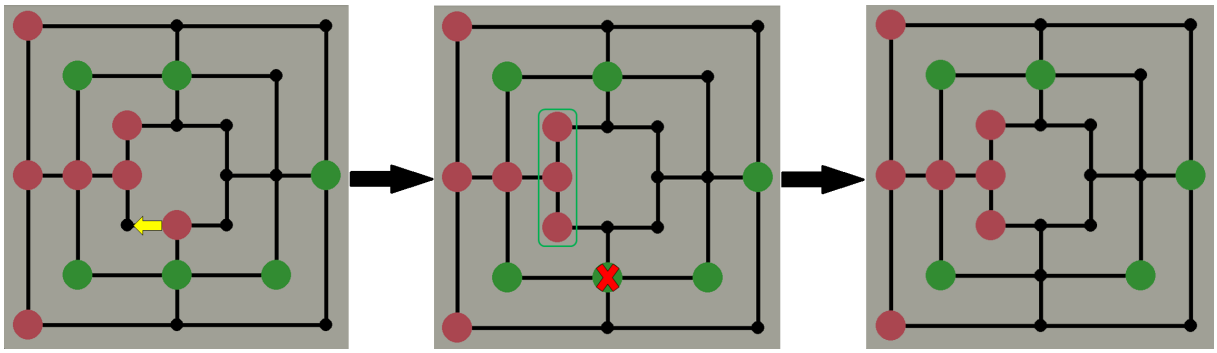

Figura 2.5: *Exemplu formare moară prin mutare și îndepărtarea piesă adversar*

2.1.3 Stări finale

Una dintre stările finale posibile este atinsă atunci când unul dintre jucători rămâne doar cu două piese, deoarece nu mai poate forma mori este declarat învins. A doua stare finală posibilă este atinsă atunci când unul dintre jucători nu mai are nicio mutare validă, ceea ce înseamnă că toate piesele sale sunt blocate, iar adversarul câștigă jocul.

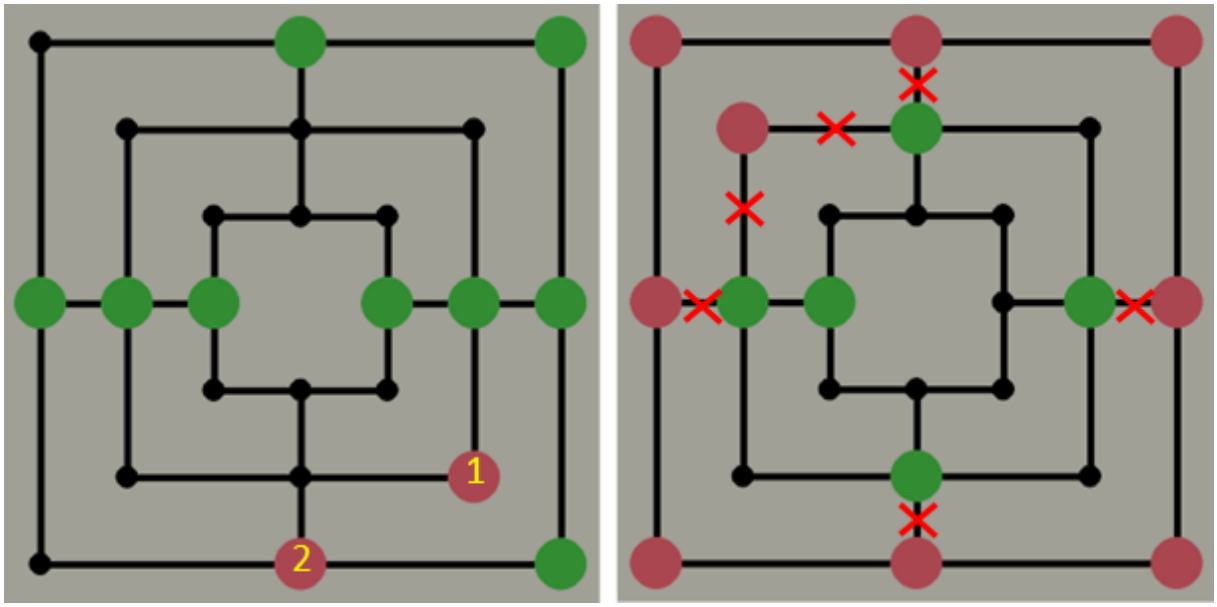

Figura 2.6: *Exemplu stari finale.*

În Figura 2.6 se pot observa cele doua tipuri de stări finale. În prima imagine se observă cum unul dintre jucători (cel cu piese portocalii) mai are doar 2 piese pe tablă, iar în a doua imagine se poate observa cum jucătorul cu piese portocalii nu are nici o mutare validă. Ambele stări de joc ilustrate duc la victoria jucatorului cu piese verzi.

2.2 Strategii

În prima fază a jocului, jucătorii experimentați nu se concentrează doar pe formarea morilor sau blocarea adversarului de a forma mori, ci pun accent pe ocuparea strategică a pozițiilor. De exemplu, ei caută să ocupe cele patru intersectii de pe pătratul mijlociu, deoarece acestea oferă avantaje strategice semnificative. În același timp, jucătorii evită să ocupe pozițiile din colțurile tablei de joc, deoarece acestea oferă mai puțină mobilitate și sunt mai ușor de blocat de către adversar. Astfel, jucătorii își stabilesc strategii în prima fază a jocului, luând în considerare pozițiile cheie și evaluând posibilitățile strategice pe măsură ce jocul avansează.

Una dintre cele mai simple strategii aplicate în joc se bazează pe faza a doua a jocului și se numeste ”Single Morris”. Prin utilizarea unei mori formate în prima fază a jocului, putem scoate o piesă din acea moară și o putem readuce în configurația inițială, formând astfel o nouă moară la fiecare două mutări. Deși această strategie poate fi ușor neutralizată prin îndepărtarea unei piese din această configurație, ea este eficientă împotriva jucătorilor începători. Această strategie simplă poate aduce avantajul de a avea mereu o moară în formare și poate pune presiune asupra adversarului, obligându-l să ia măsuri pentru a neutraliza această strategie.

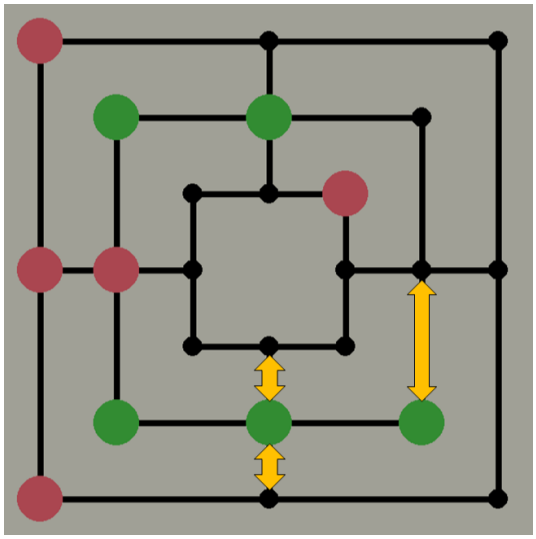

Figura 2.7: *Exemplu ”Single Morris”*

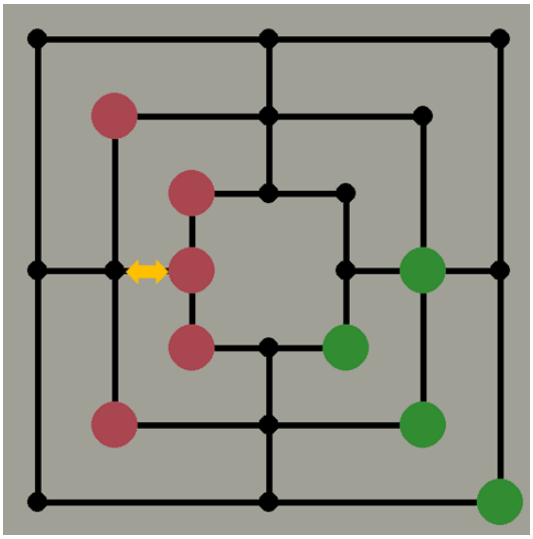

Figura 2.8: *Exemplu ”Double Morris”.*

Un exemplu de strategie eficientă în joc este cunoscută sub numele de ”Double Morris”. Această strategie constă în formarea unei configurații de piese în care, la fiecare mutare, se formează o moară. Un exemplu de astfel de configurație poate fi observat în Figura 2.8. În general, atunci când unul dintre jucători reușește să atingă o astfel de configurație, acesta are mari șanse de a câștiga jocul. Strategia ”Double Morris” reprezintă o abordare puternică în Nine Men’s Morris, permitând jucătorului să formeze rapid și eficient mori pentru a elimina piesele adversarului și a-și asigura victoria.

Capitolul 3

Recunoașterea configurației curente pe tabla de joc

Dacă utilizatorul selectează opțiunea de a juca împotriva agentului inteligent utilizând funcționalitatea de sincronizare, trebuie să urmeze următorii pași:

- Poziționează/mută piesa pe tabla de joc.
- Apasă tasta ”spațiu” pentru a semnala încheierea mutării.
- În cazul în care mutarea efectuată formează o moară, utilizatorul trebuie să îndepărteze o piesă a adversarului și apoi să apese din nou tasta ”spațiu” pentru a finaliza mutarea.
- Urmărește mutarea efectuată de agentul inteligent și poziționează/mută/îndepărtează piesele de pe tabla de joc conform indicațiilor acestuia.

Acești pași vor fi urmăriți atât în faza inițială a jocului, când utilizatorul trebuie să poziționeze piesele, cât și în faza a doua a jocului când piesele trebuie mutate.

Pentru a sincroniza tabla din interfața PyGame cu tabla reală capturată de camera video, am împărțit problema în trei componente: alinierea tablei, extragerea patch-urilor și clasificarea lor.

3.1 Alinierea tablei folosind descriptori SIFT

Pentru a alinia imaginea captată de cameră cu un șablon de referință (Figura 3.1) am folosit descriptori SIFT(Scale-Invariant Feature Transform). După cum sugerează și numele, algoritmul SIFT este o tehnică din vederea artificială utilizată pentru detectarea și definirea caracteristicilor. Acesta detectează puncte cheie sau caracteristici distinctive într-o imagine, care sunt robuste la modificări de scară, rotație și transformări. SIFT funcționează prin identificarea punctelor cheie pe baza extremelor lor de intensitate locală și

prin calcularea descriptorilor care captează informațiile locale ale imaginii din jurul acestor puncte cheie. Acești descriptori pot fi apoi utilizați pentru aplicații precum alinierea imaginilor, recunoașterea obiectelor și extragerea imaginilor [6].

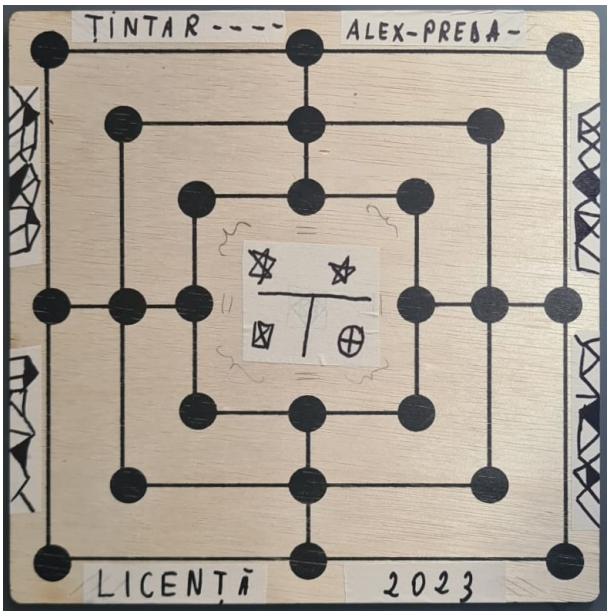

Figura 3.1: *Imagine referință.*

Principalele etape de calcul utilizate pentru a genera setul de caracteristici ale imaginii sunt:

1. Construirea spațiului scalar: Prima etapă de calcul se asigură că descriptorii vor fi independenți de scală, folosind diferențe de Gaussiene și redimensionări.

2. Localizarea punctelor de interes: Căutarea celor mai importante puncte de interes găsind maximele și minimele locale și îndepărtarea punctelor de interes scăzut.

3. Stabilirea orientării: Atribuirea orientării punctului de interes stabilit pe baza direcțiilor locale ale gradientului imaginii

4. Descriptor de puncte de interes: Calcularea descriptorului de puncte de interes [4]

3.1.1 Construirea spațiului scalar

Spațiul scalar(scalar space) este definit ca o colecție de imagini de diferite dimensiuni, care sunt generate dintr-o singură imagine de bază la care sunt adăugate imagini obțiune prin calcularea diferențelor Gaussiene intre imaginile generate folosind un filtru Gaussian [6].

Filtrul Gaussian este folosit pentru a îndepărta zgomotul unei imagini. Acesta parcurge fiecare pixel din imagine și calculează media pixelilor adiacenți, făcând astfel imaginea mai blurată. În cazul nostru, imaginea originală este trecută consecutiv prin acest filtru,

formând astfel un set de imagini din ce în ce mai blurate. Pentru a rezolva și problema de invarianța la scală, imaginea originală este înjumătățită de mai multe ori, generând astfel un spațiu scalar mai mare. Parametrii standard ai funcției din OpenCV definesc 3 injumătățiri al imaginii originale și trecerea acestora prin 5 filtre Gaussiene.

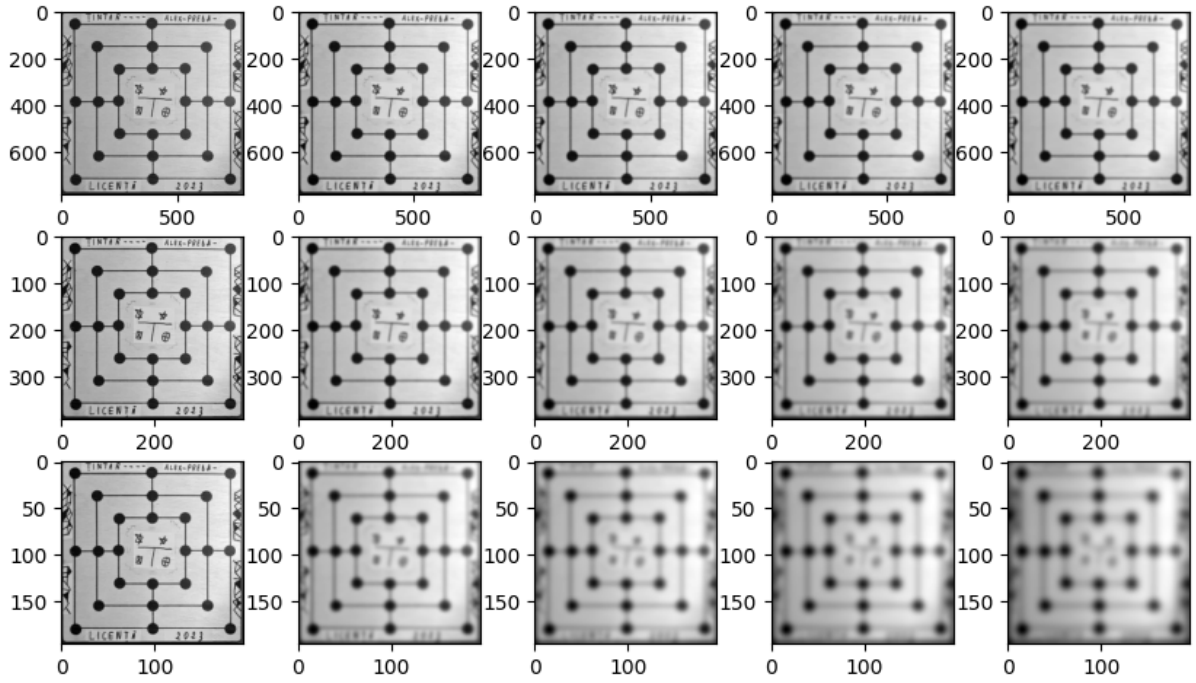

Figura 3.2: *Exemplu generare spațiu scalar blurat.*

În Figura 3.2 este ilustrată o astfel de generare de spațiu scalar în care imaginea originală a fost înjumătățită ca dimensiuni de 2 ori, din care au fost generate 4 imagini blurate.

Mai departe, algoritmul se folosește de spațiul scalar generat anterior, la care mai adaugă diferența Gaussiană între fiecare două imagini blurate de pe același rând. Diferența Gaussiana este o imagine generată din scăderea, pixel cu pixel, între cele două imagini.

În Figura 3.3 se observă eficiența diferenței între gausiene. Cu cât se avansează în blurarea imaginii se poate observă că detaliile ce nu sunt de interes din imagine, precum striațiile lemului de pe tabla de joc, sunt eliminate. Acest lucru a fost considerat revoluționar, fiind folosit chiar și 20 de ani mai târziu pentru aplicații mai simple, chiar dacă convolutiile rețelelor neuronale sunt mult mai puternice.

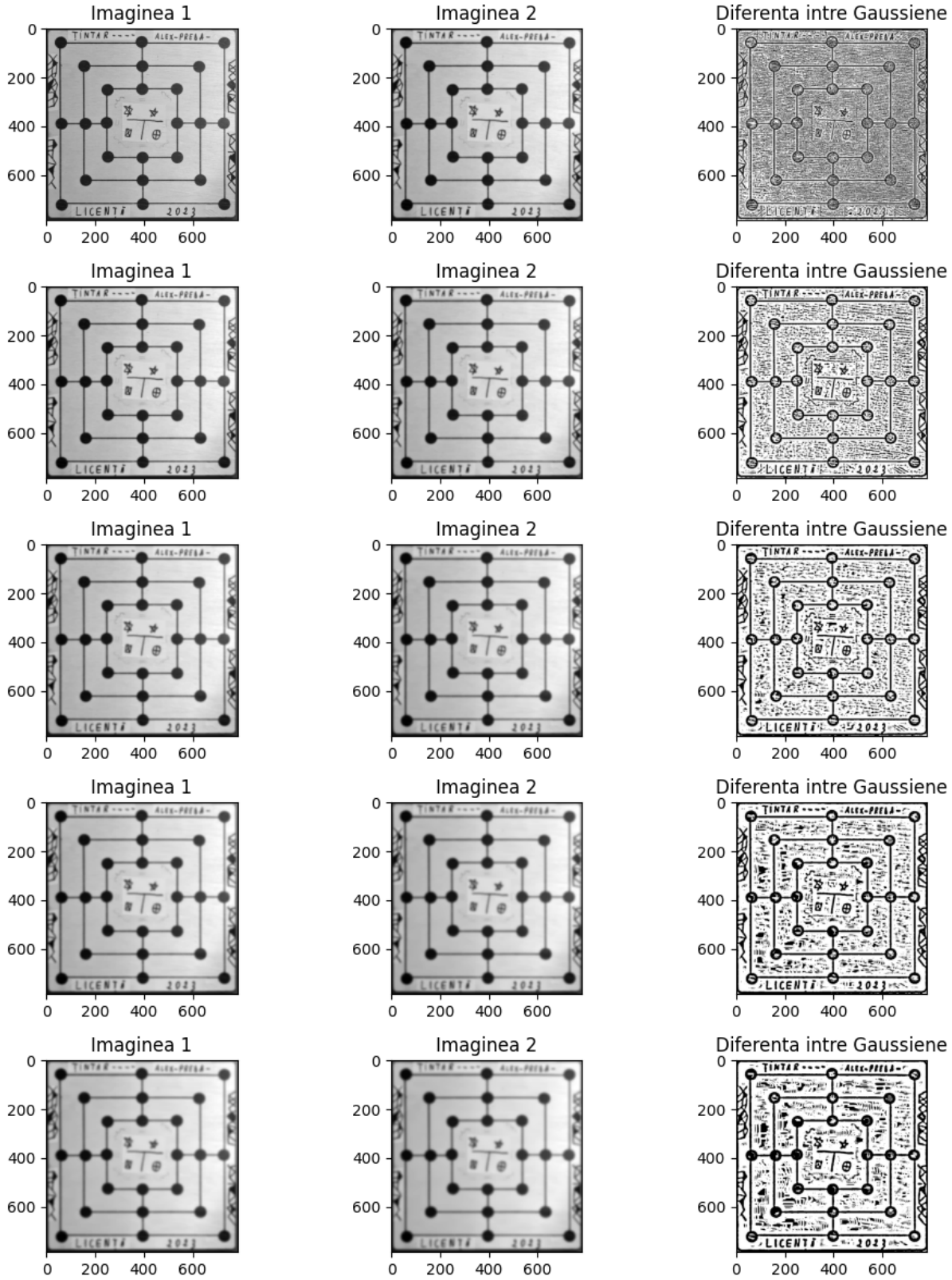

Figura 3.3: *Exemplu generare spațiu scalar Gaussian.*

3.1.2 Localizarea punctelor de interes

Având imaginile spațiului scalar, putem localiza punctele de interes. Pentru a le localiza, sunt efectuate 2 procese: găsirea punctelor de maxim și minim local și îndepărtarea punctelor ce nu sunt de interes. Pentru a găsi punctele de interes, sunt luați toți pixelii din imaginile originile (cele de pe prima coloană din Figura 3.2) și sunt comparați cu toți cei 8 vecini alăturați. Mai mult de atât, pixelii sunt comparați și cu ceilalți 9 pixeli suprapuși din imaginile generate după cea originală sau scalată, astfel se găsesc punctele de interes. Problema este că pot există puncte de interes foarte apropiate de marginea imaginii sau cu un contrast scăzut. Pentru a filtra aceste puncte de interes este aplicată extinderea Taylor de gradul doi, iar punctele de interes ce au magnitudinea mai mică de 0.03 sunt eliminate și o matrice hessiană de ordinul doi pentru a îndepărta punctele ce nu sunt foarte robuste la modificări mici de zgomot.

3.1.3 Stabilirea orientării

Pentru stabilirea orientării punctelor de interes, este construită o histograma de gradienti orientați. Histograma este împărțită în 36 de colecții, fiecare colecție conținând magnitudinea pixelilor cu gradul orientării corespunzător. Colecția din histograma ce va cuprinde cei mai mulți pixeli, va reda direcția punctului de interes. Dacă există mai multe colecții cu o înălțime mai mare de 80% decât cea mai înaltă, vor fi create două puncte de interes cu direcții diferite.

$$Magnitude = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.1}$$

$$\Phi = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{3.2}$$

Unde x_2 este valoarea pixelului din dreapta, x_1 cel din stânga, y_2 cel de sus, iar y_1 cel de jos

3.1.4 Descriptor de puncte de interes

După calcularea punctelor de interes, următorul pas este construirea descriptorului. Pentru a realiza asta, sunt luați toți vecinii punctului de interes formând 4 pătrate de 16×16 pixeli. Fiecărui pixel din fiecare pătrat îi este calculat magnitudinea și orientarea, apoi fiecare pătrat este împărțit la rândul lui în pătrate de 4×4 pixeli formându-se astfel 16 pătrate. La rândul lor, fiecărui pătrat îi este calculat o histogramă de gradienți orientați, doar că va avea 8 colecții, nu 36 că înainte. Drept urmare un descriptor va fi format

dintr-o histogramă de gradienți cu 128 de colecții.

3.2 Extragerea patch-urilor

Mai departe, având imaginea aliniată după imaginea de referință și cunoscându-i mărimea imaginii, putem deduce toate pozițiile pieselor de pe tablă. Toate aceste poziții au fost decupate în pătrate de marimea 50×50 pixeli, formandu-se astfel o lista de forma [24, 50, 50, 3].

3.3 Clasificarea patch-urilor folosind LinearSVC

Pentru a obține configurația tablei dintr-o imagine capturată cu ajutorul camerei, am preluat lista conținând cele 24 de patch-uri reprezentative și le-am clasificat în cele 3 clase: piesă portocalie, piesă verde și poziție liberă. Procesul de realizare a acestui clasificator a implicat următoarele etape: generarea datelor de antrenare și testare, antrenarea modelului și testarea acestuia.

LinearSVC este un algoritm de clasificare bazat pe mașini cu vector suport (SVM), iar SVC (Support Vector Classifier) reprezintă o implementare a SVM pentru probleme de clasificare binară sau pe mai multe clase. Scopul algoritmului LinearSVC este să găsească un hiperplan liniar optim care separă clasele de obiecte în spațiul caracteristicilor. Acest hiperplan este determinat astfel încât să maximizeze marginea dintre clase, adică distanța minimă între hiperplan și exemplele de antrenament din fiecare clasă.

LinearSVC utilizează o formulă de pierdere numită ”hinge loss” în procesul de antrenare pentru a gasi cel mai bun hiperplan de separare a claselor. Algoritmul LinearSVC are o complexitate liniară și poate trata seturi mai de date și spații de caracteristici cu un număr mare de dimensiuni.

3.3.1 Generarea datelor de antrenare și testare

Pentru generarea setului de antrenare, am utilizat algoritmul de aliniere și extragere a patch-urilor, patch-uri pe care le-am împarțit pe categorii. Pentru o obține un algoritm robust la schimbările de iluminare a tablei de joc am creat un set de date în care imaginile incluse au o variabilitate crescută în luminozitate. Setul de date de antrenare conține aproximativ 250 de imagini din fiecare categorie, iar setul de testare conține aproximativ 60 de imagini.

3.3.2 Antrenarea modelului și testarea lui

Pentru a antrena modelul LinearSVC menționat anterior, imagini din setul de antrenare au fost procesate în mai multe etape. În primul rând, imaginile au fost normalizate,

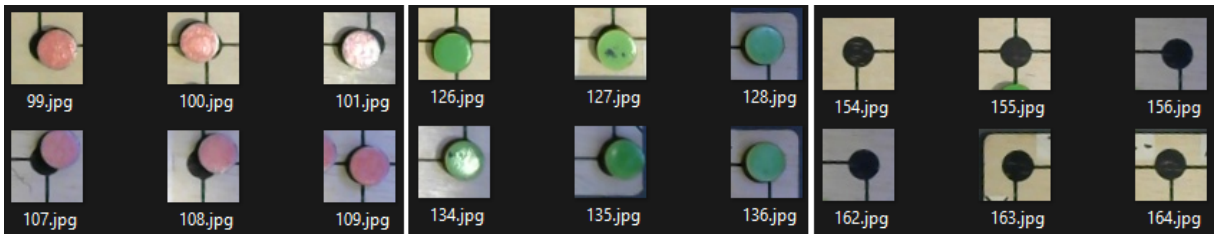

Figura 3.4: *Exemplu imagini de antrenare.*

adică valorile pixelilor au fost ajustate pentru a avea o medie cât mai apropiata de zero și o deviație standard cât mai aproape unu. Normalizarea are loc prin scăderea valorii medii și împarțirea rezultatului la deviația standard pentru fiecare canal de culoare al imaginii. Acest proces ajută algoritmul să se antreneze mai eficient și să obțină rezultate mai bune.

Pentru a extrage caracteristici semnificative din imagini, s-a utilizat rețeaua neuronală convoluțională ResNet-18. Această rețea este formată dintr-o succesiune de straturi de convoluție, urmate de straturi de normalizare batch și activare ReLU. Aceste componente formează blocuri reziduale, care permit propagarea informației prin rețea într-un mod eficient și scot în evidență caracteristici importante.

După ce imaginea originală a fost prelucrată de aceste blocuri reziduale, există două straturi finale. Primul strat convertește ieșirea ultimului bloc într-un vector unidimensional cu 512 elemente. Al doilea strat, numit strat de clasificare, este conectat la acest vector și este format din 1000 de neuroni care produc rezultatele finale utilizând funcția de activare softmax.

Descriptorii de tip ResNet-18 pe care îi folosim, sunt reprezentați de vectorul unidimensional de 512 elemente, obținut înainte de a trece prin ultimul strat al rețelei. Acest vector este utilizat pentru antrenarea modelului LinearSVC, împreună cu eticheta corespunzătoare.

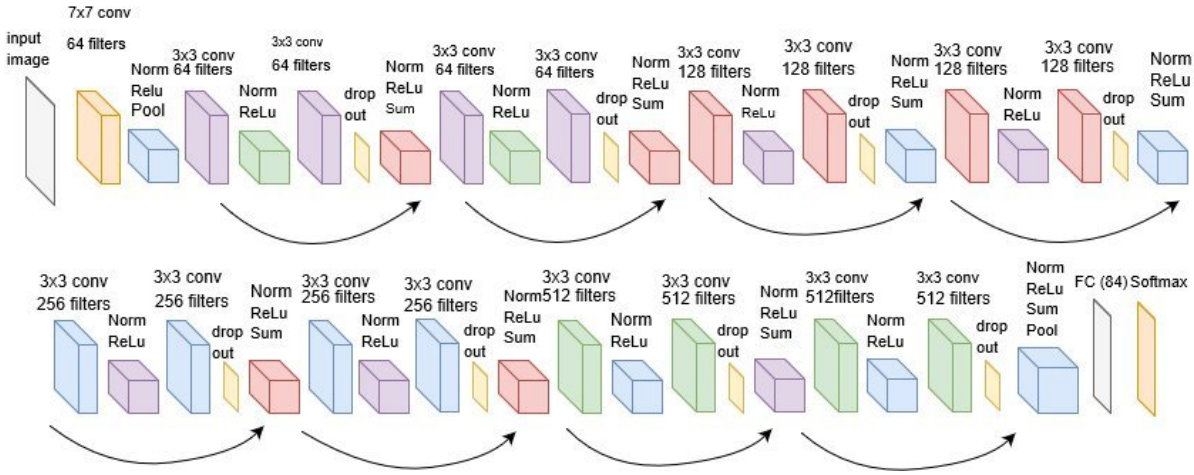

Figura 3.5: *Arhitectura rețelei ResNet-18 [2].*

3.4 Rezultatul final

În Figura 3.6 sunt ilustrați pașii transformării imaginii de intrare în lista reprezentativă configurării tablei. Imaginea de intrare este aliniată cu imaginea de referință folosind descriptori SIFT, iar din imaginea aliniată sunt extrase patch-urile ce conțin pozițiile valide ale pieselor, salvate într-o listă. Ulterior, fiecare patch este înlocuit, unul câte unul, cu predicția modelului de clasificare, 0 reprezentând o poziție liberă, 1 piesele verzi și -1 piesele portocalii.

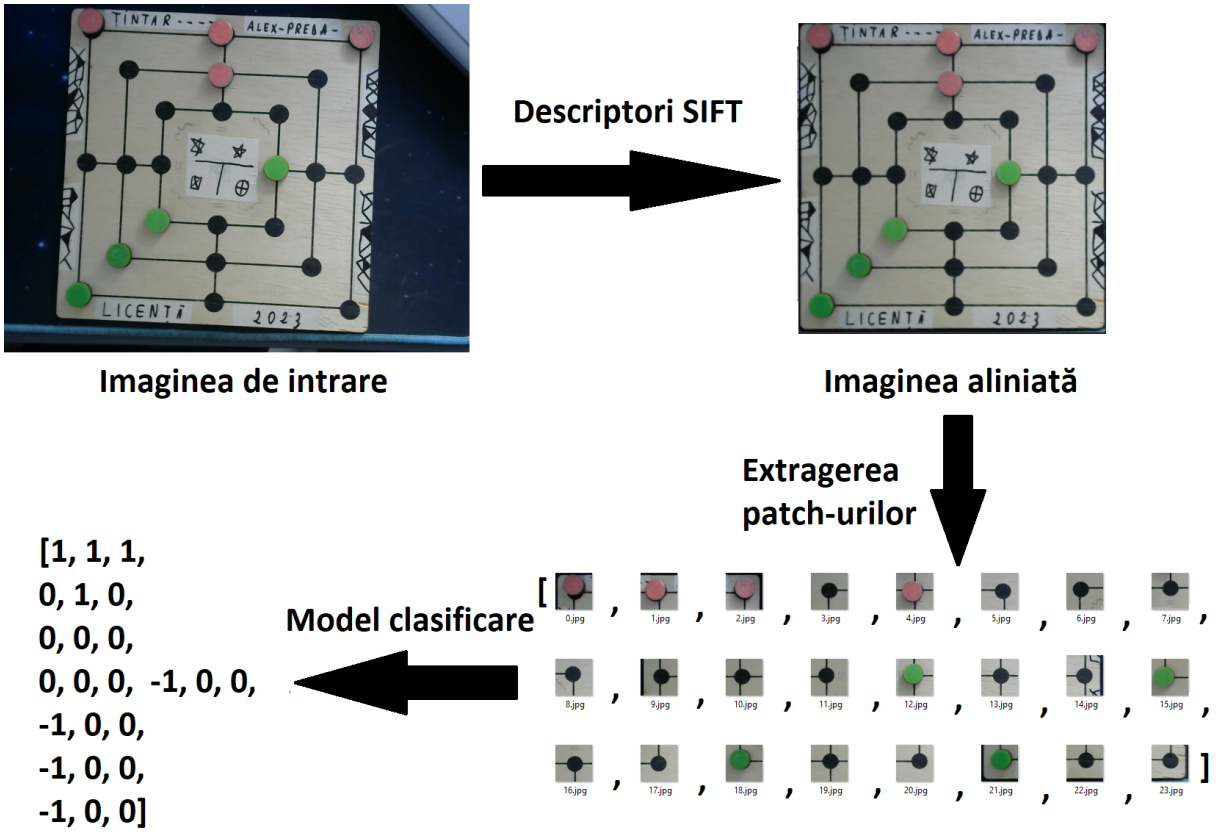

Figura 3.6: *Extragere configurație tablă*

Capitolul 4

Găsirea celei mai bune mutări

Găsirea celei mai bune mutări face referință la partea din agentul inteligent care returnează cea mai bună mutare calculată de acesta. În interfața grafică a jocului, în secțiunea de opțiuni, există doi algoritmi implementați: Învătarea pe bază de recomponse și min-max(simplu sau accelerat cu alpha-beta). În funcție de selecția facută, urmatorul joc inițializat în care este implicată prezența agentului inteligent, acesta va folosi algoritmul selectat. Algoritmul implicit este Învătarea pe bază de recomponse având cea mai bună performanță și cele mai bune mutări.

4.1 Algoritmul min-max

Min-max este un algoritm de decizie utilizat în teoria jocurilor pentru a determina cea mai bună mutare posibilă, care să maximizeze câștigul jucătorului curent. Pentru a determina această mutare optimă, algoritmul se bazează pe o funcție de aproximare a stării, numită euristică. Această funcție atribuie o valoare numerică unei stări, aproximând cât de avantajoasă este acea poziție în comparație cu celelalte.

Min-max este un algoritm similar cu backtracking pentru că acesta generează toate stările posibile până la o anumită adâncime. Adâncimea reprezintă numărul de mutări viitoare generate înainte de evaluarea stării. Pașii urmați de algoritmul min-max sunt următorii:

1. Construirea arborelui mutărilor posibile. Fiecare nod al arborelui reprezintă o stare a jocului, iar ramurile sale reprezintă stările posibile viitoare din acea stare.

2. Pentru fiecare nod frunză al arborelui, se evaluează starea cu ajutorul unei euristici, evaluarea fiind efectuată pentru jucătorul initial.

3. Cu ajutorul algoritmului, de la frunză la radacină, se propagă valorile nodurilor, decizie facută în funcție de rolul jucătorului. În cazul nodurilor MAX, valoarea aleasă este cea maximă, reprezentând mutarea optimă pentru jucătorul curent. În

cazul nodurilor MIN, valoarea aleasă este cea minimă, reprezentând mutarea optimă pentru adversar. Cu această metodă de selecție, ambii jucători vor alege mutarile optime.

4. Valoarea ajunsă la radacină reprezintă următoarea stare a jocului.

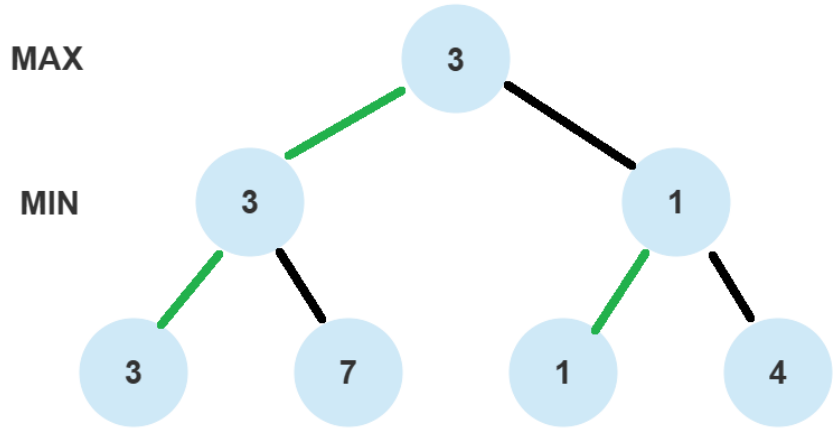

Figura 4.1: *Arbore generate de algoritmul min-max*

4.2 Algoritmul alpha-beta

La fel ca și algoritmul min-max, alpha-beta este un algoritm folosit în rezolvarea jocurilor, generând stările viitoare și folosind o euristică pentru a stabili dacă o stare este mai avantajoasă decât celelalte. Diferența principală între cei doi algoritmi constă în modul în care sunt explorate nodurile arborelui generat.

Algoritmul min-max explorează toate mutările posibile până la adâncimea maximă a arborelui și evaluează stările frunzelor, în timp ce algoritmul alpha-beta optimizează această căutare prin tăieri ale ramurilor arborelui. Alpha-beta elimină explorarea ramurilor irelevante, ceea ce duce la o reducere semnificativă a timpului de calcul.

Algoritmul alpha-beta utilizează doi parametri, alpha și beta, care reprezintă limitele inferioare și superioare ale valorilor posibile ale unui nod. Acești parametri au valori inițiale și sunt actualizați pe parcursul explorării arborelui. Implementarea acestui algoritm are rolul de a îmbunătăți timpul necesar pentru determinarea stării următoare, permițând explorarea la o adâncime mai mare și luând în considerare mai multe mutări.

In Figura 4.2 este ilustrat un arbore generat cu adancimea 3. De asemenea este ilustrat si modul in care decide cum se propaga valorile frunzelor la radacina si cum sunt taiate ramurile inutile.

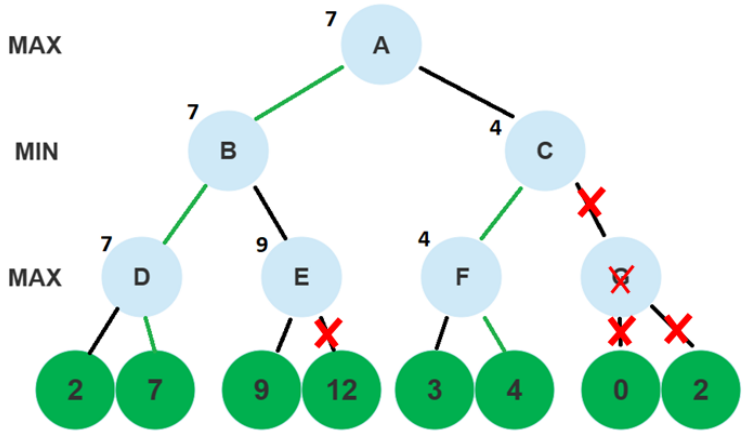

Figura 4.2: *Arbore generat de algoritmul alpha-beta*

Pașii parcurși pentru căutarea celei mai bune mutari în Figura 4.2 sunt:

1. Algoritmul este recursiv și începe din nodul A, unde alpha este $+\infty$, iar beta este $-\infty$. Nodul A ia valoarea maximă între B și C, însă deoarece căutarea începe cu nodurile din stânga, B este generat primul.

2. Nodul B ia valoarea minimă între D și E, și este generat mai întâi D.

3. Nodul D ia valoarea maximă între frunzele sale, iar frunzele sunt generate în ordine, astfel că nodul D trebuie să decidă dacă trebuie să continue căutarea sau nu, verificând condiția beta <= alpha. Deoarece condiția este falsă, alpha devine 2, iar nodul D devine 2.

4. Nodul D se uită și la frunza din dreapta. Deoarece 2 este mai mic decât 7, D și alpha devin 7.

5. Nodul D returnează valoarea 7 către B. B verifică minimul dintre $+\infty$și 7, care este 7, deci nodul B este asigurat că va avea o valoare minimă de 7 și continuă căutarea în nodul E.

6. În nodul E, alpha și beta sunt $-\infty$, respectiv 7. Nodul E se uită la prima frunză și găsește valoarea 9. Alpha devine 9, însă deoarece condiția alpha <= beta devine adevărată (deoarece alpha = 9 și beta = 7), nu mai are sens ca E să continue căutarea printre frunze și returnează valoarea 9 lui B.

7. Deoarece nodul B a primit valoarea 7 de la D și valoarea 9 de la nodul E, acesta alege minimul dintre cele două, astfel încât nodul A va avea garantată o valoare de cel puțin 7.

8. Nodul A începe căutarea cu nodul din dreapta. În nodul C, alpha este 7, iar beta este $+\infty$, și caută în F.

9. F se uită la frunza din stânga, care are valoarea 3. Maximul dintre 7 și 3 este 7, deci căutarea continuă.

10. F se uită la frunza din dreapta, care are valoarea 4. Maximul dintre 5 și 4 este 5, astfel că alpha rămâne 7.

11. Nodul F returnează valoarea 4 către C, iar beta alege minimul dintre $+\infty$ și 4. Deoarece condiția beta <= alpha devine adevărată (deoarece beta = 4 și alpha = 7), căutarea este încheiată, iar celelalte ramuri ale lui C sunt tăiate.

12. Acest lucru are perfectă logică deoarece nodul A dorește o valoare mai mare decât valoarea returnată de nodul B. Având în vedere că C are deja valoarea minimă 4, chiar dacă continuă căutarea și primește o valoare mai mare de la nodurile din subarborele său, acesta va alege în continuare 4, luând o decizie de minimizare.

13. Drept urmare, nodul A rămâne cu valoarea 7, iar căutarea este terminată.

4.3 Funcții euristici

Funcția euristică este o tehnică de a determina valoarea unei stări într-un joc. În funcție de valoarea determinată, putem decide dacă o stare este mai avantajoasă decât alta.

Pentru a avea o evaluare cât mai bună a unei stări, am implementat 8 funcții euristice:

1. "Last move is morris": returnează 1 dacă jucătorul a format o moară în ultima mutare, -1 dacă adversarul a format o moră și 0 în cazul în care nicio moară nu a fost formată.

2. "Number of closed morrises": returnează diferența dintre numărul de mori formate de jucător și cel al adversarului.

3. "Number of blocked opponent pieces": returnează diferența dintre numărul de piese blocate ale adversarului și ale jucătorului.

4. "Difference between the number of yours and yours opponent's morrises": returnează diferența dintre numărul de piese ce formează o moară și ale adversarului

5. "Number of 2 piece configurations": returnează diferența dintre numărul de configurații cu două piese ale jucătorului și ale adversarului.

6. "Number of Double morris": returnează diferența dintre numărul de configurații cu mori duble ale jucătorului și ale adversarului (o moară dublă este formata din două mori cu o piesă comună).

7. ”Winning configuration”: returnează 1 dacă jucătorul câștigă, -1 dacă adversarul câștigă și 0 dacă nu este o stare finală.

8. ”Best heuristic”: este o combinație a euristicilor prezentate, înmulțite cu un coeficient în funcție de importanța fiecărei stări [3].

4.4 Algoritmul bazat pe paradigma învătării prin recompense

Pentru a integra algoritmul bazat pe paradigma învătării prin recompense, m-am bazat pe rețeaua implementată și analizată de F. Chesani, A. Galassi, M. Lippi și P. Mello, denumită NNMM (Neural Nine Men’s Morris) [1]. Aceștia au dezvoltat o rețea neuronală capabilă să înțeleagă regulile jocului și să obțină performanțe remarcabile în comparație cu alți agenți inteligenți, care au fost învățați doar să câștige.

4.4.1 Arhitectura modelului

Modelul primește un șir de caractere cu o lungime de 28 de caractere. Primele 24 de caractere reprezintă configurația tablei de joc, în care piesele jucătorului sunt reprezentate de litera ”M”, piesele adversarului sunt reprezentate de litera ”E”, iar pozițiile libere sunt reprezentate de litera ”O”. Ultimele 4 caractere din acest șir reprezintă: numărul de piese ale jucătorului care trebuie plasate pe tabla de joc, numărul de piese ale adversarului care trebuie plasate pe tabla de joc, numărul de piese ale jucătorului deja aflate pe tabla de joc și numărul de piese ale adversarului deja aflate pe tabla de joc. Pentru a realiza o predicție referitoare la următoarea mutare, modelul are nevoie și de informații despre faza de joc în care se află, informații care pot fi deduse din ultimele patru caractere.

Predicția mutării constă în trei numere: TO, FROM și REMOVE.

- TO reprezintă locația unde piesa jucătorului trebuie plasată.
- FROM este opțional și reprezintă locația de unde piesa trebuie mutată. Această predicție este activă doar în faza a doua a jocului. În cazul în care jocul se află în prima fază, valoarea lui FROM este 0.
- REMOVE este opțional și reprezintă locația piesei jucătorului adversar care trebuie înlăturată în cazul în care mutarea formează o moară. Dacă mutarea nu formează o moară, valoarea lui REMOVE este 0.

Pentru a realiza predicția, modelul constă în trei rețele neuronale profunde, fiecare responsabilă de prezicerea unei valori: TO, FROM și REMOVE. Fiecare rețea este formată dintr-un set de straturi neuronale, care efectuează calcule complexe pentru a genera predicțiile.

Rețeaua responsabilă de predicția valorii TO are un strat terminal format din 24 de neuroni, fiecare neuron reprezentând cele 24 de stări posibile ale tablei de joc. Acest strat permite modelului să identifice poziția optimă pentru plasarea piesei jucătorului.

Rețeaua responsabilă de predicția valorilor FROM și REMOVE conține, în plus față de rețeaua pentru valoarea TO, un neuron suplimentar care reprezintă valoarea 0 pentru mutările care nu implică o acțiune de mutare sau înlăturare. Acest neuron asigură modelului să ofere predicții corespunzătoare în funcție de contextul jocului.

Prin utilizarea acestor trei rețele neuronale distincte, modelul estimează cu precizie cele trei acțiuni importante în cadrul jocului: plasarea piesei, mutarea piesei și înlăturarea piesei adversarului.

După fiecare predicție a fiecărei valori, rezultatul este verificat pentru a determina dacă mutarea este validă. În situația în care una dintre valorile prezise reprezintă o mutare invalidă, rețeaua generează o nouă predicție. Această etapă de verificare asigură că mutările generate de rețea sunt conforme cu regulile jocului.

Conform documentației rețelei [1], aceasta este compusă din trei rețele reziduale, fiecare formată din mai multe blocuri. Arhitectura acestor rețele poate fi observată în Figura 4.3. Fiecare rețea are un strat inițial cu 200 de neuroni, urmat de mai multe unități reziduale, introduse una după cealaltă. Rețelele TO și FROM sunt alcătuite din 10 unități reziduale, formând 22 de straturi, în timp ce rețeaua REMOVE este formată din 30 de astfel de unități și are 62 de straturi.

4.4.2 Seturile de antrenare și evaluare

Pentru antrenarea și evaluarea performanței modelului, au fost generate două seturi de date. Primul set de date, setul complet, conține toate stările posibile ale jocului, în total 1.626.673 de stări. Al doilea set de date, setul optimizat, conține doar stările unice ale jocului din care au fost filtrate miscările optime având doar 100.154 de stări. În setul complet, mai multe stări pot apărea de mai multe ori din cauza anumitor transformări, cum ar fi rotirea tablei de joc cu 90 de grade. Deși sunt 4 stări diferite ale jocului, în esență există o singură mutare optimă.

Pentru evaluarea acestei rețele, s-au realizat mai multe teste, printre care testul de legalitate al mutărilor și testul de acuratețe al mutărilor. Ambele teste sunt relevante, iar rezultatele obținute sunt următoarele: testul de legalitate are o acuratețe de 99.53%, ceea ce înseamnă că majoritatea mutărilor prezise de rețea sunt corecte din punct de vedere legal. Pe de altă parte, testul de acuratețe obține o valoare mai mică, de 37.20%, ceea ce indică faptul că mutările efectuate de rețea nu sunt neapărat cele mai bune mutări posibile.

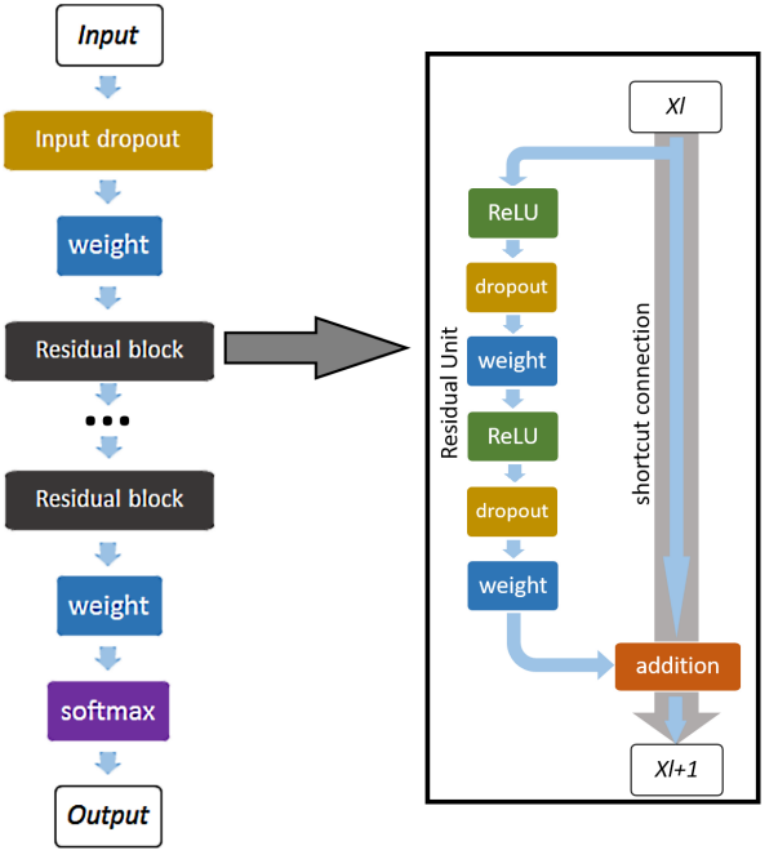

Figura 4.3: *Ilustrare rețea reziduală. Imagine preluată din [1]*

4.4.3 Integrare in aplicatie

Pentru a utiliza această rețea în aplicația mea, rulez două servere în fundal: un server care rulează pe adresa IP 127.0.0.1 și portul 5082, ce prezice mutările pentru jucătorii care utilizează piesele portocalii, și un server care rulează pe aceeași adresă IP, dar pe portul 5083, ce prezice mutările pentru jucătorii care utilizează piesele verzi. Primul pas în utilizarea rețelei a fost să preiau informațiile despre starea jocului și să le traduc în formatul necesar. Având formatul necesar și știind cine trebuie să efectueze o mutare, am creat un client care se conectează la serverul deschis anterior. Acest client trimite starea și primește înapoi cei trei parametri esențiali: TO, FROM, REMOVE. Având acești parametri, jucătorul poate efectua mutarea, iar jocul continuă.

Capitolul 5

Evaluări experimentale

În acest capitol descriem rezultatele obținute modelului de clasificare, testarea euristicilor și a retelei. Modelul de tip LinearSVC a avut cea mai bună performanță antrenat pe descriptori ResNet-18, dar am incercat mai multe variante precum: imaginile originale vectorizate, histograma de pixeli gri, histograma pentru toate canalele de culoare(RBG) si descriptori hog.

Euristicile sunt o bună aproximare a stării unui joc, însă nu se pot compara cu timpul de răspuns și acuratețea rețelei cu invațări prin recompense.

5.1 Modelul de clasificare

Pentru a evalua performanța modelului de clasificare în ceea ce privește imaginile, am calculat acuratețea acestuia. Setul de antrenare conține 772 de imagini, în timp ce setul de testare conține 179 de imagini. Acestea au fost împărțite în mod aproximativ egal între cele trei categorii.

1. În prima încercare, am aplicat o vectorizare a imaginilor, obținând o acuratețe de 69.27%. Chiar și cu o simplă vectorizare, rezultatele sunt promițătoare. Dacă setul de antrenare era mai voluminos, există posibilitatea ca acuratetea să crească chiar pană la 100%.

2. În a doua încercare, am convertit imaginile într-o histogramă de nuanțe de gri. Deoarece culoarea este un aspect semnificativ în clasificarea acestor imagini, acuratețea a scăzut, atingând 58.1%.

3. În a treia încercare, am îmbunătățit rezultatele celei de-a doua încercări prin generarea unei histograme de culori RGB. Această metodă a obținut rezultate promițătoare, cu o acuratețe de 92.73%.

4. În a patra încercare, am încercat să transform imaginile în descriptori HOG (Histogram of Oriented Gradients). Totuși, similar cu a doua încercare, descriptorii HOG

au fost calculați doar pe imagini în tonuri de gri, ceea ce a dus la o acuratețe scăzută de 69.27%.

5. În încercarea finală, am folosit descriptori ResNet-18 pentru a transforma imaginile. Această abordare a obținut cele mai bune rezultate, deoarece operațiile efectuate de rețeaua neurală pe toate straturile sale au evidențiat multiple caracteristici cheie. Acuratețea obținută în acest caz a fost de 100%, model pe care l-am folosit pentru clasificarea patch-urilor extrase.

	Algoritm	Acuratețe
1	Vectorizarea imaginilor	0.692
2	Conversia în histogramă de nuanțe gri	0.581
3	Conversia în histogramă RGB	0.927
4	Descriptori HOG	0.692
5	Descriptori ResNet-18	1

Tabela 5.1: *Evaluările experimentale ale modelului de clasificare.*

5.2 Cea mai bună euristică

Pentru a determina cea mai bună euristică, am organizat un campionat între cele 8 euristici. Campionatul a fost structurat în formatul ”bracket” cu 8 echipe, în care fiecare euristică a fost reprezentată de o echipă. Pentru a determina euristica câștigătoare, am organizat un total de 40 de meciuri: 20 de meciuri în care o euristică a avut prima mutare și alte 20 de meciuri în care cealaltă euristică a avut prima mutare. La finalul celor 40 de meciuri, euristica care a obținut cele mai multe victorii a avansat în etapa următoare.

Pentru a asigura o competiție echitabilă, adâncimile algoritmilor folosiți în joc au fost ajustate. Inițial, adâncimea era setată la 3 pentru algoritmul de plasare a pieselor și 3 pentru algoritmul de mutare a pieselor. Apoi, adâncimea a fost modificată la 3 cu 5, 5 cu 5 și 5 cu 7 pentru o serie de 5 meciuri din fiecare categorie.

În Figura 5.1 este ilustrat progresul campionatului. După simularea celor 280 de partide de joc, euristica cu numele ”Best heuristic” a ieșit ca fiind cea mai bună euristică.

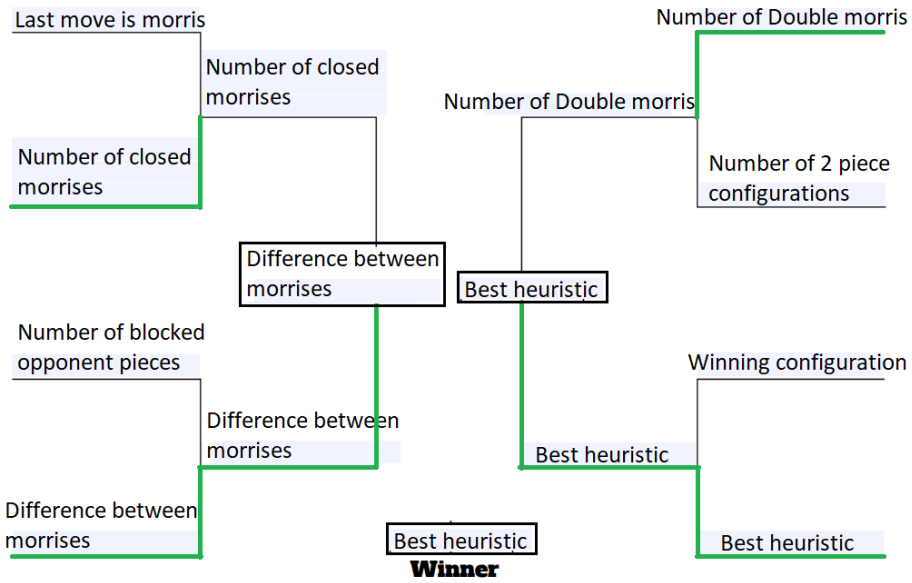

Figura 5.1: *Campionat euristici*

5.3 Performanța rețelei

Pentru a testa performanța rețelei, am simulat 100 de jocuri împotriva celor mai bune euristici care au ieșit în topul campionatului prezentat anterior. Am considerat că este suficientă simularea a doar 100 de jocuri pentru că rețeaua a ieșit învingătoare în toate acestea. Având în vedere faptul că algoritmul de min-max sau alpha-beta nu a reușit să realizeze nici măcar o victorie împotriva rețelei, se poate spune că este mult superioară. În ciuda faptului că obiectivul rețelei a fost să învețe jocul țintar fără a-i cunoaște regulile, aceasta a învățat și strategii eficiente.

De asemenea, pentru a evalua acuratețea rețelei, m-am pus și eu în locul adversarului încercând să obțin o victorie. După simularea aproximativ a 20 de partide de joc, pot concluziona că rețeaua este mult mai eficientă și mai avansată decât un jucător obișnuit de țintar.

Capitolul 6

Concluzii

Această lucrare de licență a reușit să combine componente din diferite domenii într-o singură aplicație funcțională. Aplicația permite utilizatorului să joace un joc de țintar împotriva unui agent inteligent ce operează pe mai multe nivele de dificultate sau împotriva altui utilizator. De asemenea, o altă realizare importantă este posibilitatea de a juca pe o tablă reală care este sincronizată cu cea din aplicație, astfel încât să se poată verifica corectitudinea mutărilor sau să se poată realiza antrenamente profesionale.

Toate componentele aplicației au avut particularități dificile de implementat, pre1:

1. Definirea regulilor jocului în interfața grafică, având în vedere că acesta funcționează în mai multe faze, precum și sincronizarea mutărilor utilizatorului cu cele ale agentului inteligent.

2. Găsirea soluției pentru generarea adaptivă în funcție de faza de joc, deși algoritmii tradiționali nu au prezentat dificultăți majore.

3. Sincronizarea tablei a ridicat o serie de probleme, de la alegerea descriptorului adecvat pentru alinierea tablei, până la implementarea unui clasificator pentru configurarea tablei.

4. Importarea algoritmului de învățare prin recompensă a fost un proces anevoios din cauza utilizării unor tehnologii mai vechi și a lipsei de documentație.

În concluzie, consider că tehnicile și aplicația prezentate pot reprezenta un punct de plecare pentru aprofundarea jocurilor de tablă utilizând atât algoritmi tradiționali, cât și învățare profundă. În același timp, aplicația dezvoltată își îndeplinește scopul de a implementa algoritmii propuși, demonstrându-și utilitatea.

Bibliografie

[1] M. Lippi F. Chesani A. Galassi și P. Mello, în 10.4 (Dec. 2018), pp. 344–353, URL: doi:%2010.1109/TG.2018.2804039.

[2] DOKUMACI KEREM, *Brain tumor prediction w/ ResNet in PyTorch*, Available at: https://www.kaggle.com/code/keremdokumaci/brain-tumor-prediction-w-resnet-in-pytorch, Accessed: June 2, 2023.

[3] Kartik Kukreja, *Heuristic/Evaluation Function for Nine Men's Morris*, Available at: https://kartikkukreja.wordpress.com/2014/03/17/heuristicevaluation-function-for-nine-mens-morris/, Accessed: June 4, 2023.

[4] David G. Lowe, „Distinctive Image Features from Scale-Invariant Keypoints", în *International Journal of Computer Vision* (2004), URL: https://people.eecs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf.

[5] Ian Millington și John Funge, „Artificial Intelligence for Games", în (2009), URL: http://www.matt-versaggi.com/mit_open_courseware/GameAI/ArtificialIntelligencefor pdf.

[6] Aishwarya Singh, *SIFT Algorithm | How to Use SIFT for Image Matching in Python (Updated 2023)*, Available at: https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python, Accessed: May 31, 2023.

[7] Georgios Yannakakis și Julian Togelius, „Artificial Intelligence and Games", în (2018), URL: http://gameaibook.org/book.pdf.