UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA TEHNOLOGIA INFORMAȚIEI

Proiect de diplomă

AI în jocul Snake: algoritmi genetici și reinforcement learning

Coordonator științific
Conf. Dr. Alexe Bogdan

Absolvent
Anca Alexandru-Iulian

București, iulie 2025

Rezumat

Această lucrare propune o analiză comparativă între două metode AI aplicate jocului Snake: algoritmi genetici și reinforcement learning. Scopul urmărit este dezvoltarea unor agenți capabili să învețe să joace eficient, cu obiectivul final de a umple complet tabla de joc. Ambele metode utilizează rețele neuronale pentru luarea deciziilor, dar diferă fundamental în modul de optimizare a acestora: algoritmul genetic evoluează o populație de indivizi pe baza unui scor de fitness, în timp ce reinforcement learning actualizează rețeaua în urma interacțiunii agentului cu mediul. Lucrarea descrie arhitectura software, strategia de antrenare, mecanismul de evaluare și o comparație între performanțele obținute.

Abstract

This paper presents a comparative analysis of two AI methods applied to the game of Snake: genetic algorithms and reinforcement learning. The main objective is to develop agents capable of learning to play efficiently, with the ultimate goal of completely filling the game board. Both methods rely on neural networks for decision-making, but they differ fundamentally in their optimization approach: the genetic algorithm evolves a population of individuals based on a fitness score, while reinforcement learning updates the network through the agent's interaction with the environment. The paper includes a description of the software architecture, training strategy, evaluation mechanism, and a performance comparison of the two approaches.

Cuprins

Capitolul 1

Introducere și descrierea jocului

Jocul Snake, cunoscut pentru mecanica sa simplă și intuitivă, oferă un cadru ideal pentru dezvoltarea și evaluarea algoritmilor de inteligență artificială. Complexitatea sa nu se regăsește în regulile jocului, ci în numărul mare de decizii posibile care trebuie luate într-un mediu dinamic. Acest lucru îl transformă într-un mediu excelent pentru antrenarea agenților care trebuie să învețe comportamente eficiente prin explorare și adaptare.

1.1 Obiectivul lucrării

Lucrarea de față propune o comparație între două metode de optimizare a comportamentului unui agent în jocul Snake: algoritmi genetici și reinforcement learning. Ambele metode folosesc rețele neuronale ca mecanism de decizie, dar abordările de antrenare sunt fundamental diferite. Algoritmii genetici operează asupra unei populații de indivizi, care evoluează iterativ pe baza unui scor de fitness. În schimb, reinforcement learning presupune un agent care învață direct din interacțiunea cu mediul prin optimizarea unei funcții de recompensă.

În cadrul acestei lucrări sunt prezentate, pas cu pas, regulile jocului, modul în care au fost implementate cele două metode de antrenament, arhitectura aplicației și rezultatele obținute în urma testării. Toate componentele sistemului au fost dezvoltate în întregime ca parte a proiectului, iar experimentele descrise au fost concepute și realizate exclusiv în această lucrare.

1.2 Prezentarea jocului Snake

Snake este un joc clasic, apărut inițial pe dispozitive mobile și calculatoare, în care jucătorul controlează un șarpe ce se deplasează pe o hartă bidimensională. Scopul principal este colectarea merelor fără a se ciocni cu pereții sau cu propriul corp. De fiecare dată când șarpele mănâncă un măr, lungimea acestuia crește, ceea ce face jocul progresiv mai dificil.

Figura 1.1: Interfață clasică a jocului Snake

1.2.1 Reguli de bază

- Șarpele se deplasează într-una dintre cele patru direcții: sus, jos, stânga, dreapta.
- Coliziunea cu pereții sau cu propriul corp duce la încheierea jocului.
- Fiecare măr consumat adaugă un segment la coada șarpelui și crește scorul cu 1.

1.2.2 Reprezentarea internă

Implementarea jocului folosește drept hartă un grid rectangular. Fiecare poziție ocupată de corpul șarpelui sau de un măr este stocată sub formă de coordonate (x, y). Capul șarpelui este primul element din listă, urmat de segmentele corpului în ordinea apariției. Direcția de deplasare este exprimată printr-un vector (dx, dy), iar poziția corpului șarpelui se modifică la fiecare pas cu o singură celulă în direcția curentă.

1.2.3 Metrici de performanță

Pentru evaluarea agenților, sunt urmărite următoarele metrici:

- **Scorul total** — numărul de mere colectate într-o sesiune de joc.
- **Numărul de pași** — totalul pașilor efectuați de șarpe până la finalul jocului. Un număr mai mic de pași indică o strategie mai eficientă.
- **Finalizarea jocului** — agentul este considerat de succes dacă reușește să ocupe complet harta.

Aceste metrici sunt utilizate atât în faza de antrenament, cât și în procesul de comparare a celor două metode implementate.

Capitolul 2

Descrierea metodelor propuse

Pentru antrenarea agentului au fost propuse și implementate două abordări distincte: una bazată pe algoritmi genetici și cealaltă pe reinforcement learning, folosind metoda Proximal Policy Optimization (PPO). Ambele metode utilizează rețele neuronale cu structuri asemănătoare, dar se bazează pe concepte de optimizare diferite. În această secțiune sunt prezentate componentele comune și specifice fiecărei metode, precum și observații asupra comportamentului agenților antrenați.

2.1 Rețeaua neuronală utilizată

2.1.1 Arhitectura rețelei

Rețeaua neuronală utilizată în cadrul ambelor metode este un *Multi-Layer Perceptron* de tip *feedforward*. Aceasta este compusă dintr-un strat de intrare, două straturi ascunse și un strat de ieșire.

Detalii despre fiecare strat sunt enumerate mai jos, iar o reprezentare vizuală este oferită în Figura 2.1:

- **Strat de intrare**: 32 de neuroni, corespunzând vectorului de stare al șarpelui;
- **Strat ascuns 1**: 20 de neuroni
- **Strat ascuns 2**: 12 de neuroni

- **Strat de ieșire**: 3 neuroni, corespunzători celor trei acțiuni posibile: viraj la stânga, mers înainte și viraj la dreapta.

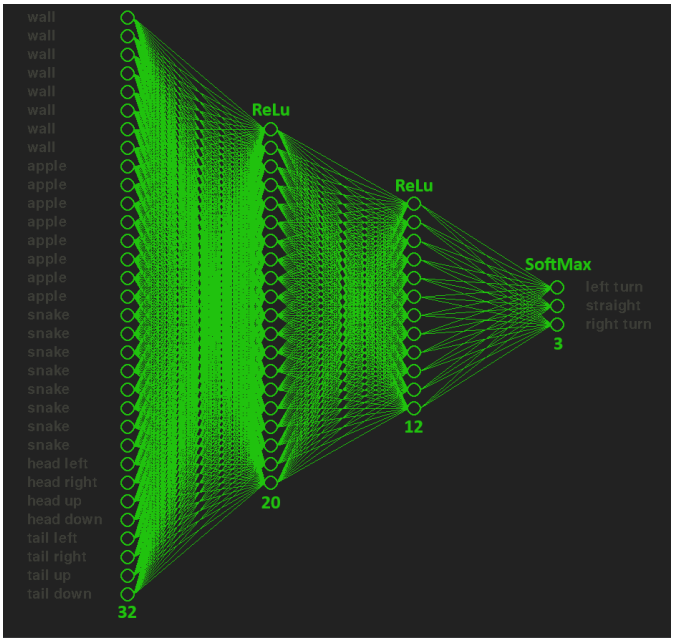

Figura 2.1: Structura rețelei neuronale utilizate de ambii agenți

Fiecare strat este complet conectat cu următorul, iar funcția de activare utilizată după fiecare strat ascuns este ReLU, aleasă pentru simplitatea și eficiența sa computațională. Ultimul strat utilizează funcția de activare Softmax, transformând ieșirea într-o distribuție de probabilitate, din care acțiunea finală este selectată în funcție de valoarea maximă.

Rețeaua a fost implementată folosind biblioteca `PyTorch`, care oferă un cadru flexibil pentru definirea și antrenarea modelelor neuronale. Straturile au fost definite utilizând `torch.nn.Linear` și organizate într-un obiect `torch.nn.Sequential`, o abordare preferată în locul definirii explicite a fiecărui strat, deoarece din testele efectuate a rezultat o performanță mai bună. Rețeaua este implementată ca o subclasă a clasei `torch.nn.Module`, permițând gestionarea automată a parametrilor și integrarea cu optimizatori standard din cadrul `PyTorch`.

2.1.2 Datele de intrare

Datele de intrare ale rețelei neuronale sunt extrase din mediul jocului și oferă o reprezentare compactă a stării curente a șarpelui. Vectorul de intrare are dimensiunea 32 și este alcătuit din următoarele componente:

- **8 valori** care indică distanța până la perete în 8 direcții radiale;
- **8 valori** care indică distanța până la corpul șarpelui în cele 8 direcții;
- **8 valori** care indică prezența unui măr în cele 8 direcții;
- **4 valori** care codifică direcția curentă a capului șarpelui (`[sus, jos, stânga, dreapta]`);
- **4 valori** care codifică direcția cozii șarpelui, în același format.

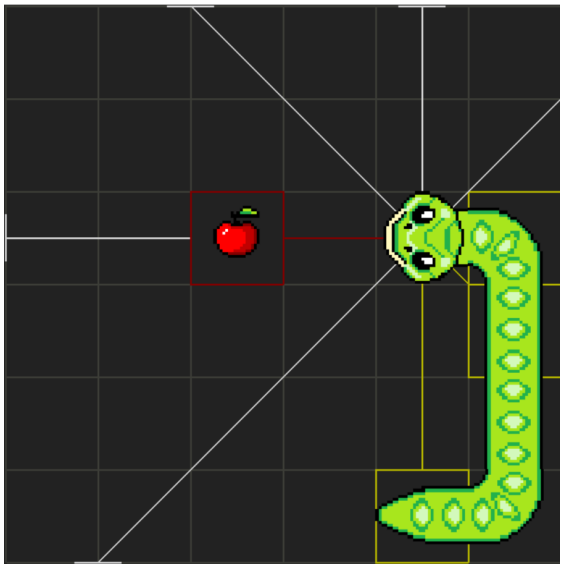

Figura 2.2: Reprezentare vizuala a viziunii șarpelui

În Figura 2.2 este ilustrată vizual această „viziune” a șarpelui: din poziția capului, se emit raze în cele opt direcții pentru a detecta prezența unui măr sau distanța până la un perete sau segment de corp. Informațiile rezultate sunt ulterior normalizate și utilizate ca intrare în rețea.

2.1.3 Datele de ieșire

Ieșirea rețelei neuronale constă într-un vector de dimensiune 3, în care fiecare componentă reprezintă probabilitatea asociată uneia dintre cele trei acțiuni posibile:

- viraj la dreapta față de direcția curentă;
- menținerea direcției curente;
- viraj la stânga față de direcția curentă.

Selectarea acțiunii se realizează pe baza valorii maxime din acest vector, corespunzător celei mai probabile acțiuni conform predicției rețelei. Astfel, dacă prima valoare este maximă, agentul va executa un viraj la stânga; dacă valoarea maximă se află pe poziția a doua, se menține direcția curentă; iar dacă a treia valoare este dominantă, se efectuează un viraj la dreapta.

2.1.4 Rolul rețelei în algoritm

Rețeaua neuronală acționează ca o funcție de decizie. La fiecare pas, starea curentă a jocului este transmisă ca intrare, iar rețeaua returnează un vector cu trei valori. Acțiunea aleasă este cea corespunzătoare celei mai mari valori, determinând astfel direcția de deplasare a șarpelui.

Această logică este aplicată identic în ambele metode, diferența constând în modul de antrenare al rețelei, detaliat în secțiunile următoare.

2.2 Metoda evolutivă: Algoritmul Genetic

Algoritmii genetici reprezintă o metodă de optimizare inspirată din procesul de selecție naturală, în care populații de indivizi evoluează treptat pentru a rezolva o anumită problemă. În contextul aplicației, aceștia sunt utilizați pentru optimizarea comportamentului șarpelui, prin adaptarea parametrilor rețelei neuronale.

Fiecare individ din populație corespunde unei instanțe a rețelei definite anterior, caracterizată printr-un set unic de ponderi și biasuri. Acești parametri

constituie genomul agentului și sunt supuși unui proces evolutiv iterativ, bazat pe selecție, recombinare și mutație. Antrenarea se realizează exclusiv prin modificarea acestor valori, arhitectura rețelei rămânând neschimbată pe tot parcursul evoluției.

2.2.1 Procesul evolutiv

Procesul de evoluție implică trei componente fundamentale: selecția, recombinarea și mutația parametrilor rețelelor neuronale.

Selecția se bazează pe scorul de fitness: un număr fix de indivizi cu cele mai bune performanțe sunt păstrați neschimbați în generația următoare. Această strategie elitistă conservă caracteristicile eficiente și reduce riscul de pierdere a indivizilor performanți în urma mutațiilor.

Recombinarea are ca scop generarea de descendenți prin combinarea informației genetice (parametrii rețelelor) de la doi părinți selectați. În cadrul acestei lucrări, recombinarea se aplică direct asupra matricilor de ponderi și vectorilor de bias, care sunt reprezentați ca array-uri de numere reale.

Se utilizează două metode:

- *Încrucișare binară cu un punct*: se alege aleator o poziție în vectorul de parametri (sau într-o matrice), iar componentele sunt schimbate între părinți de la acel punct înainte. Procesul este ilustrat în Figura 2.3.

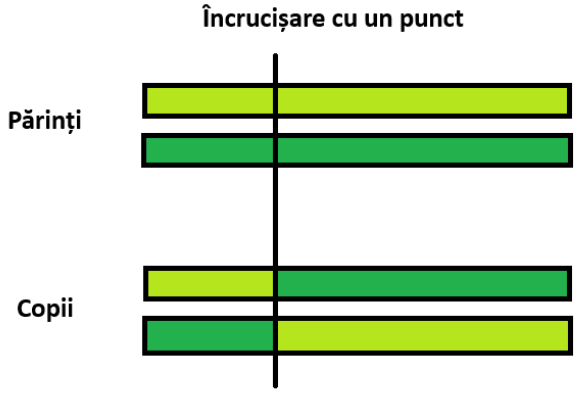

Figura 2.3: Reprezentare grafică a încrucișării cu un punct

- *Încrucișare uniformă*: fiecare parametru este selectat aleatoriu de la unul dintre cei doi părinți. O reprezentare grafică a acestei metode este prezentată în Figura 2.4.

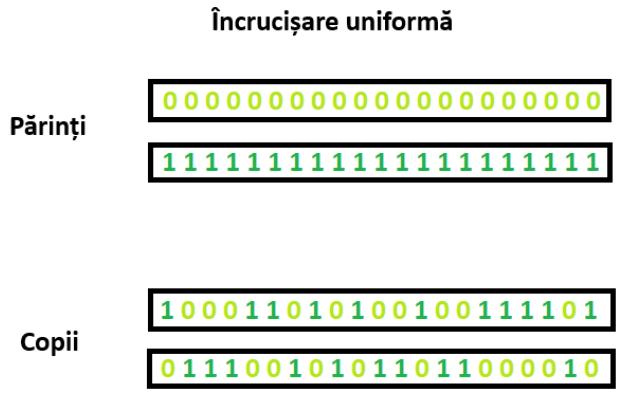

Figura 2.4: Reprezentare grafică a încrucișării uniforme

Mutația introduce diversitate în populație prin modificarea aleatoare a valorilor parametrilor descendenților. În această lucrare, s-a utilizat o *mutație gaussiană*, prin care fiecare element numeric al ponderilor și biaseurilor poate fi ajustat prin adăugarea unui zgomot generat dintr-o distribuție normală.

2.2.2 Evaluarea performanței

Pentru a determina calitatea unui individ, acesta este evaluat la finalul unei sesiuni complete de joc. Se calculează un scor de fitness care ia în considerare numărul total de pași supraviețuiți și numărul de mere consumate. Formula utilizată este preluată și adaptată dintr-un proiect open-source disponibil pe GitHub [2]:

$$F = S + \left(2^A + A^{2.1} \cdot 500\right) - \left((0.25 \cdot S)^{1.3} \cdot A^{1.2}\right) \tag{2.1}$$

unde:

- F este scorul de fitness final,

- S reprezintă numărul total de pași,
- A reprezintă numărul de mere colectate.

Scorul de fitness penalizează stagnarea sau mișcările repetitive care nu contribuie la colectarea de mere și recompensează comportamentele care implică explorare eficientă și maximizarea lungimii.

2.2.3 Procesul de antrenament

Inițial, se generează o populație de indivizi cu proprietăți aleatorii. Fiecare individ este evaluat, iar cei cu cele mai mari scoruri de fitness sunt selecționați. Prin recombinare și mutație, alături de indivizii cu cele mai bune scoruri, se generează o nouă generație care înlocuiește populația curentă. Acest proces se repetă timp de un număr fix de generații sau până la atingerea unei performanțe satisfăcătoare.

De-a lungul procesului, evoluția populației poate fi monitorizată prin urmărirea atât a fitness-ului, cât și a scorului mediu și maxim. La final, individul cu cel mai bun scor este păstrat și utilizat pentru testare comparativă cu cealaltă metodă.

În prima etapă, antrenarea s-a realizat folosind o populație de 100 de indivizi, dintre care cei mai buni 20 erau selectați pentru a forma baza generației următoare. În faza finală a optimizării, configurația a fost extinsă la 500 de indivizi, cu 100 de indivizi selectați în fiecare generație, pentru a crește diversitatea și calitatea selecției. Procesul a fost rulat pe aproximativ 1000 de generații. Evoluția scorului celui mai bun individ de-a lungul generațiilor este ilustrată în Figura 2.5.

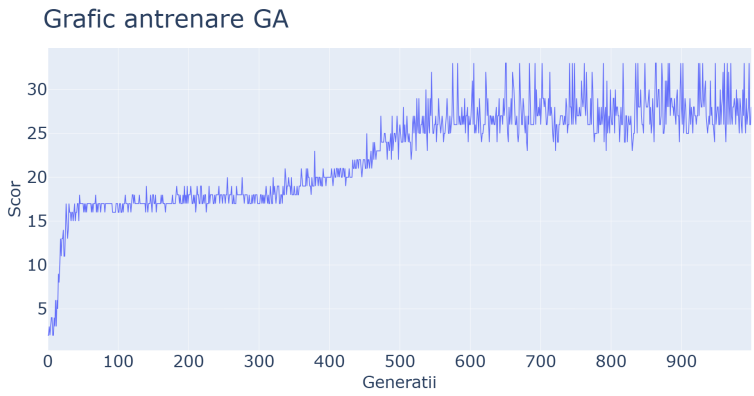

Figura 2.5: Evoluția scorului în timpul antrenării agentului folosind algoritmi genetici

2.2.4 Avantaje și limitări

Metoda s-a dovedit eficientă în formarea unor comportamente competitive, chiar și în absența unor recompense clare sau a unui ghidaj precis. Cu toate acestea, rămân câteva provocări importante: costul computațional ridicat, ritmul lent de convergență și dependența puternică de alegerea funcției de fitness. De asemenea, menținerea diversității în populație este esențială pentru a evita blocarea într-o soluție suboptimală prea devreme.

2.3 Reinforcement Learning: PPO

Algoritmul *Proximal Policy Optimization* este o metodă de reinforcement learning care optimizează direct o politică stocastică, utilizând o funcție de tip *policy gradient* cu restricții explicite asupra actualizărilor. Spre deosebire de metodele bazate pe funcții de valoare, PPO ajustează parametrii rețelei care generează distribuția acțiunilor, maximizând în același timp recompensa acumulată de agent.

În implementarea de față, agentul este compus din două rețele neuronale separate: una pentru politica de acțiune (*actor*) și una pentru estimarea valorii stărilor (*critic*). Actorul returnează o distribuție categorială, din care se extrage acțiunea selectată, iar criticul oferă o estimare a valorii asociate stării curente. Această separare permite actualizări independente ale celor două componente, cu obiective

distincte.

Pentru a preveni modificările instabile, a fost introdusă funcția *clipped*, care limitează raportul dintre probabilitățile noii politici și celei anterioare. În plus, actualizările sunt realizate în loturi mici și repetate de mai multe ori la fiecare iterație de învățare.

Metoda este *on-policy*, ceea ce înseamnă că fiecare actualizare se bazează pe date noi, colectate în urma interacțiunilor directe ale agentului cu mediul. În contextul jocului *Snake*, agentul învață exclusiv din propriile experiențe, fără a beneficia de reguli programate sau exemple predefinite. Comportamentul este ajustat progresiv pe baza recompenselor primite în timpul jocului.

2.3.1 Structura rețelelor actor și critic

În cadrul metodei PPO implementate în această lucrare, politica și funcția de valoare sunt modelate folosind rețele neuronale complet conectate, antrenate simultan, dar cu scopuri diferite.

Actorul este responsabil de alegerea acțiunii și are o arhitectură simplă, compusă din două straturi ascunse cu 20, respectiv 12 neuroni, activate prin funcții ReLU. Stratul final aplică o funcție *softmax* pentru a genera o distribuție de probabilitate asupra celor trei acțiuni posibile.

Criticul este o rețea separată, mai profundă, formată din două straturi ascunse a câte 64 de neuroni fiecare. Aceasta are ca scop estimarea valorii stării curente și este antrenată prin regresie, folosind eroarea față de recompensa estimată.

Separarea arhitecturală între actor și critic permite o specializare clară: actorul este optimizat pentru maximizarea avantajului, iar criticul pentru estimarea precisă a valorii. Antrenamentul ambelor rețele se realizează cu ajutorul algoritmului Adam.

2.3.2 Antrenarea agentului

Antrenamentul agentului PPO se bazează pe acumularea de experiență prin interacțiune directă cu mediul, urmată de actualizarea rețelelor neuronale pe baza acestor date. La fiecare episod, agentul observă starea curentă, alege o acțiune conform politicii sale, primește o recompensă și ajunge într-o stare nouă. Fiecare

astfel de tranziție este memorată, împreună cu probabilitatea acțiunii și valoarea estimată a stării.

După un număr suficient de pași, aceste tranziții sunt organizate în loturi și utilizate pentru optimizarea parametrilor rețelelor. Pentru estimarea avantajului asociat fiecărei acțiuni, se folosește o variantă de calcul bazată pe estimarea generalizată a avantajului (GAE), care ia în considerare diferențele dintre recompense și valorile criticului pe termen scurt și mediu.

Actualizarea actor-ului se face cu ajutorul unui obiectiv PPO specific, care penalizează deviațiile prea mari față de politica anterioară. Acest lucru este realizat printr-un raport între probabilitățile noii politici și celei vechi, care este apoi limitat într-un interval prestabilit, astfel se menține stabilitatea procesului de învățare.

Pe de altă parte, criticul este antrenat pentru a reduce eroarea dintre valorile estimate și cele reale, calculate pe baza sumelor de recompense și avantajului observat. Cele două rețele sunt optimizate simultan, iar procesul este repetat pe parcursul mai multor epoci, pentru fiecare set de date colectat.

Actualizările au loc periodic, după un număr fix de pași, ceea ce permite agentului să învețe treptat, dar constant, din experiențele acumulate. Rețelele sunt salvate ori de câte ori se obține un scor mai bun decât cel anterior.

2.3.3 Observații asupra comportamentului

În primele episoade, acțiunile agentului sunt în mare parte aleatorii și conduc adesea la coliziuni rapide. Pe măsură ce antrenamentul progresează, apar comportamente tot mai coerente: agentul începe să evite pereții, să stea departe de propriul corp și să se apropie cu mai multă siguranță de mere.

Comparativ cu agentul bazat pe algoritm genetic, cel antrenat prin PPO manifestă un comportament mai stabil și mai adaptiv. Deși nu maximizează întotdeauna scorul, deciziile sunt mai echilibrate, iar riscurile inutile sunt evitate mai frecvent. Totuși, în unele episoade, politica învățată poate genera secvențe repetitive, mai ales când diferențele dintre acțiuni nu sunt clar distinse de rețea.

Agentul bazat pe algoritmul Proximal Policy Optimization (PPO) a fost antrenat pe parcursul a 10 000 de jocuri consecutive, în cadrul aceluiași mediu stan-

dardizat. Pentru a evidenția tendința de învățare, în Figura 2.6 sunt prezentate mediile scorurilor obținute la fiecare 10 episoade.

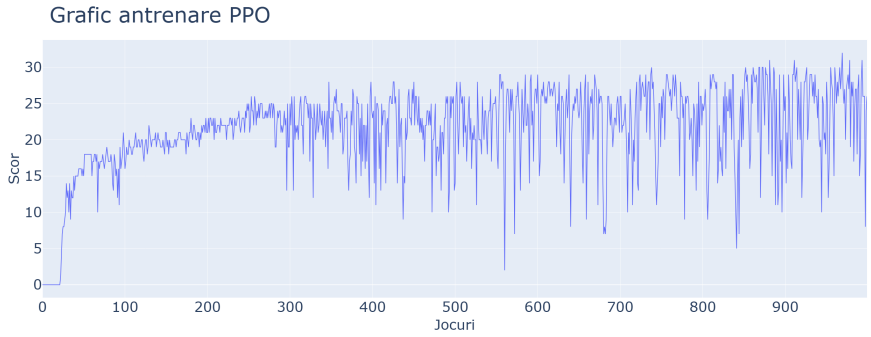

Figura 2.6: Evoluția scorului în timpul antrenării agentului folosind PPO

2.3.4 Avantaje și limitări

Utilizarea algoritmului PPO în contextul jocului *Snake* oferă mai multe avantaje. În primul rând, agentul poate învăța strategii eficiente fără a avea acces la reguli explicite sau trasee predefinite, întreaga politică fiind construită prin experiență directă, ceea ce duce la o adaptabilitate bună în situații variate. Mai mult, separarea dintre actor și critic permite o optimizare clară a fiecărei componente, iar mecanismul de actualizare cu restricții (*clipping*) contribuie la stabilitatea învățării.

Pe de altă parte, PPO este sensibil la alegerea hiperparametrilor, precum rata de învățare, coeficientul de clip sau dimensiunea loturilor. În lipsa unor recompense bine stabilite, agentul poate învăța comportamente suboptime sau repetitive. De asemenea, fiind un algoritm *on-policy*, nu poate reutiliza eficient datele vechi, ceea ce duce la un cost de antrenament mai ridicat în comparație cu metodele *off-policy*.

Capitolul 3

Evaluare și comparație

3.1 Scopul evaluării

Scopul acestui capitol este de a compara performanța celor două metode propuse. Evaluarea urmărește să evidențieze diferențele dintre cele două abordări în termeni de eficiență, stabilitate și capacitate de adaptare la mediu.

Pentru a avea un punct de referință clar, comparația include și o soluție naivă, în care șarpele traversează toată harta urmând o traiectorie ciclică prestabilită. Deși această strategie reușește mereu să termine jocul, este ineficientă din punct de vedere al pașilor făcuți. Prezența acestei soluții în evaluare ajută la înțelegerea diferenței dintre un comportament programat și unul învățat.

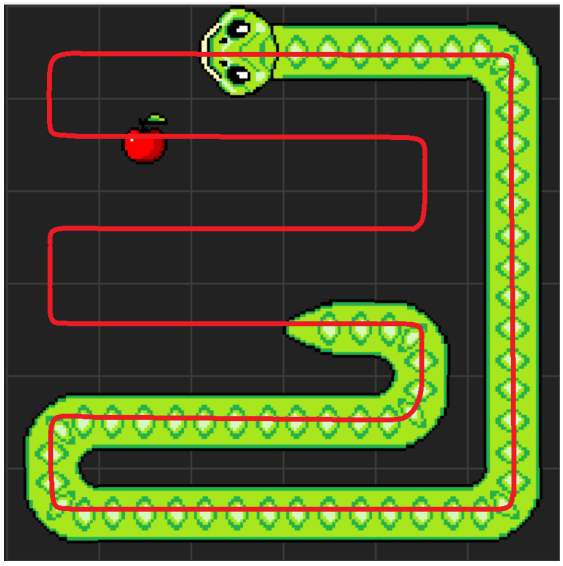

Figura 3.1: Vizualizare traiectorie soluție naivă

3.2 Metodologie

Pentru a compara performanța celor două s-au realizat o serie de teste controlate, desfășurate în condiții identice pentru fiecare agent. A fost folosit același mediu de joc, cu o hartă de dimensiune $(6, 6)$ și un singur măr activ în orice moment. Fiecare agent a fost testat pe un număr de 1000 de episoade, pornind de la aceeași stare inițială.

În faza de evaluare, comportamentul fiecărui agent a fost determinist: acțiunea aleasă în fiecare stare a fost cea cu probabilitatea maximă. Astfel, s-au eliminat efectele aleatorii din politica stocastică a PPO, iar scorurile obținute reflectă strict calitatea politicii învățate.

Pentru fiecare episod s-au înregistrat următorii indicatori:

- scorul final (numărul de mere colectate),
- numărul total de pași supraviețuiți,
- numărul de episoade terminate cu succes,
- media și abaterea standard a scorului.

În cadrul acestui test, un episod este considerat finalizat cu succes dacă agentul reușește să atingă un scor egal cu numărul total de pătrățele de pe hartă minus lungimea inițială a șarpelui(3 segmente), ceea ce, în cadrul testării pe o hartă de dimensiune 6×6, corespunde unui scor de 33.

În plus, a fost inclusă și soluția naivă. Deși nu este capabilă să reacționeze la mediu, această strategie atinge mereu scorul maxim,însă într-un număr mai mare de pași. Aceasta oferă un reper clar pentru performanța *neinteligentă*.

Pentru a evalua eficiența fiecărui agent din perspectiva numărului de pași necesari pentru finalizarea completă a jocului, a fost realizat un test separat. În acest scop, au fost extrase câte 1000 de episoade în care fiecare agent a reușit să atingă scorul maxim de 33.

Această metodologie a permis o analiză obiectivă între cele două metode de învățare, dar și o contextualizare a rezultatelor în raport cu o abordare statică.

3.3 Analiza rezultatelor

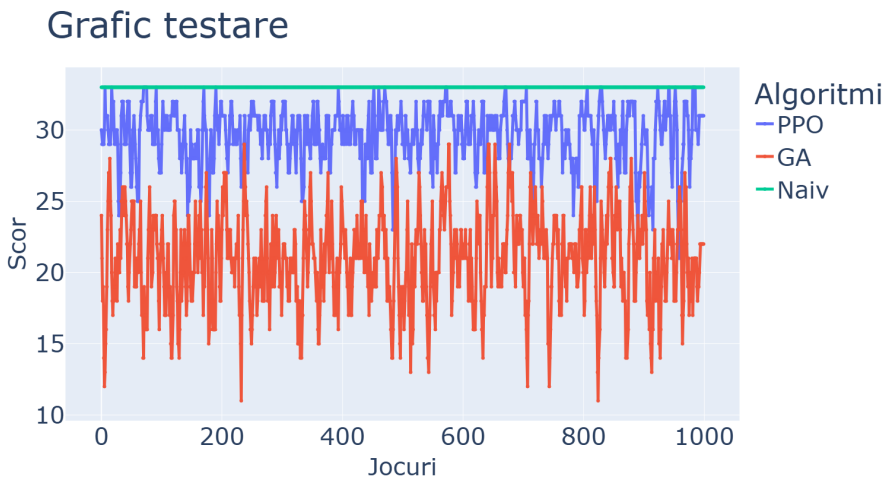

Figura 3.2: Compararea scorurilor obținute de fiecare agent în 1000 de episoade de testare.

Din Figura 3.2 se poate observa că agentul antrenat prin *Reinforcement Learning* a demonstrat un comportament constant și bine calibrat. Acesta reușește să ajungă frecvent la scorul maxim de 33, iar media scorurilor sale este foarte apropiată de acest prag. În plus, deciziile sale se adaptează rapid la poziția mărului și la forma actuală a corpului său, evitând coliziuni în situații complexe.

Spre deosebire de PPO, agentul bazat pe algoritm genetic a obținut o performanță mai modestă. Deși în anumite episoade a reușit să atingă scoruri ridicate, în general, media scorurilor este mai redusă. Comportamentul său tinde să fie mai rigid, iar în situațiile complexe, unde traiectoria necesită ajustări frecvente, agentul manifestă dificultăți în găsirea unor soluții eficiente. Acest aspect indică o lipsă de adaptabilitate în strategia învățată și o sensibilitate crescută la configurații spațiale dificile.

În timpul testării, indiferent de metoda de antrenare utilizată, s-au observat situații recurente în care șarpele ajunge să se blocheze în propriul corp, în special în fazele avansate ale jocului, când lungimea acestuia devine semnificativă. De cele mai multe ori, astfel de blocaje apar din cauza unei succesiuni de decizii suboptime care determină agentul să formeze un traseu închis, fără cale de ieșire, chiar dacă mărul se află în apropiere. Câteva exemple tipice de astfel de situații sunt ilustrate în Figura 3.3, unde șarpele nu reușește să își reorganizeze traiectoria și rămâne blocat într-un traseu închis.

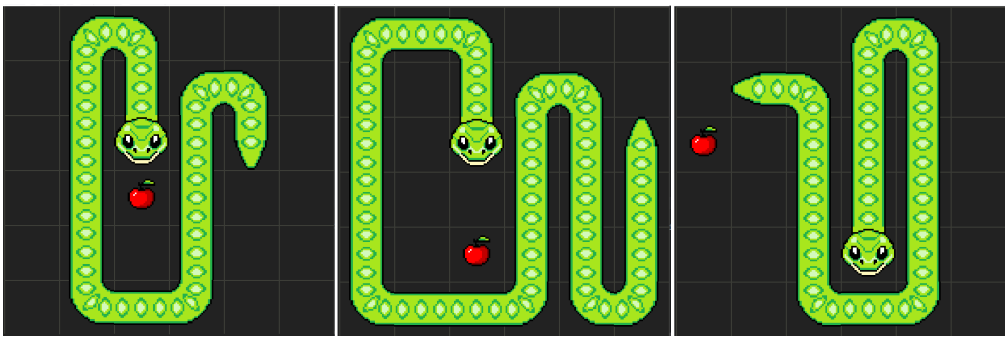

Figura 3.3: Situații în care șarpele ajunge blocat în propriul corp

Soluția naivă, deși nu învață nimic din mediu, urmează o traiectorie fixă care garantează evitarea oricărei coliziuni. Astfel, reușește să atingă scorul maxim în toate episoadele. Totuși, acest comportament vine cu un cost semnificativ: numărul de pași necesari pentru finalizarea jocului este vizibil mai mare decât în cazul agenților inteligenți. Această ineficiență evidențiază diferența dintre o strategie sigură, dar lentă, și una învățată, care optimizează atât scorul, cât și timpul de execuție.

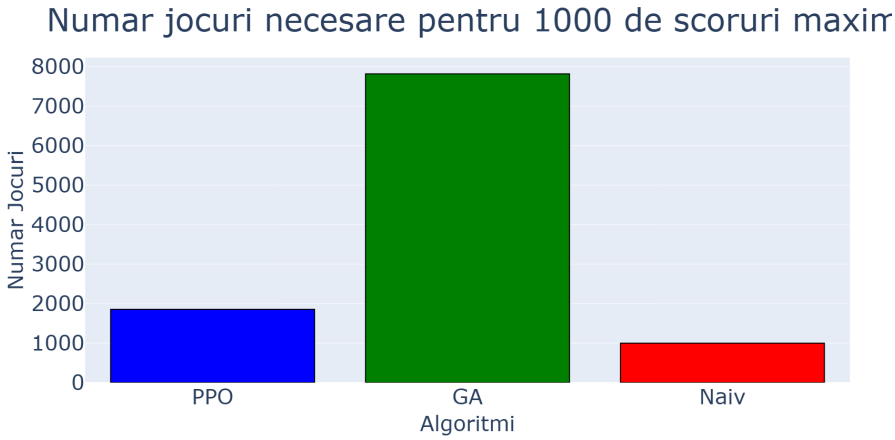

Figura 3.4: Numărul de jocuri necesare pentru a obține 1000 de episoade încheiate cu scorul maxim.

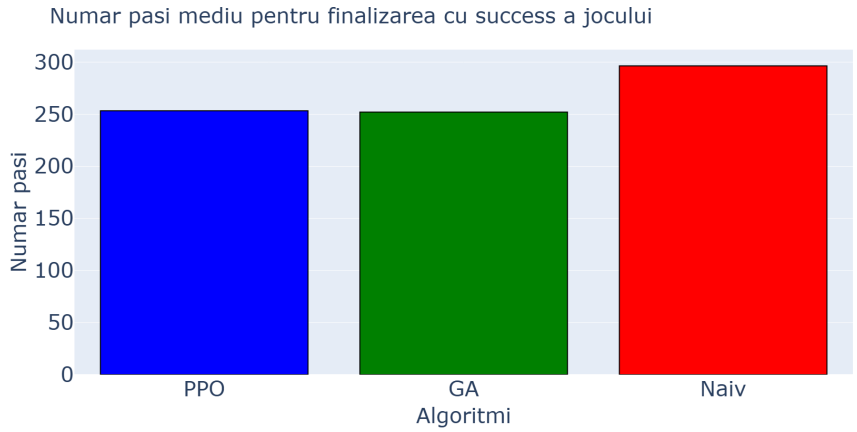

Figura 3.5: Numărul mediu de pași necesari pentru finalizarea cu succes a jocului.

În Figura 3.4 și Figura 3.5 este evidențiată diferența între cele trei metode analizate în ceea ce privește consistența și eficiența finalizării jocurilor cu succes. Primul grafic arată numărul de jocuri necesare pentru a obține 1000 de episoade încheiate cu scor maxim. Se observă că agentul PPO are o rată de succes ridicată, agentul bazat pe algoritm genetic are o rată de reușită mult mai scăzută, iar soluția naivă reușește, în mod previzibil, să finalizeze fiecare joc. Al doilea grafic arată că atât agentul PPO, cât și agentul genetic finalizează jocul, în medie, cu un număr mai mic de pași comparativ cu soluția naivă, demonstrând o eficiență mai bună în optimizarea traiectoriei.

Această analiză completă arată că, dintre cele trei abordări testate, agentul PPO este cel mai echilibrat, reușind să combine adaptabilitatea, eficiența și consistența decizională. Agentul genetic rămâne competitiv în anumite situații, dar nu atinge nivelul de generalizare al PPO, iar soluția naivă, deși garantează finalizarea jocului, este lipsită de inteligență reală.

Capitolul 4

Concluzii

Rezumatul contribuțiilor

Lucrarea a urmărit compararea a două metode de antrenare a agenților pentru jocul *Snake*: algoritmi genetici și reinforcement learning prin PPO. A fost implementat un mediu complet de testare, iar pentru fiecare metodă s-a antrenat un agent capabil să învețe din interacțiunea cu jocul.

Evaluarea în condiții identice a evidențiat comportamente distincte și a permis o analiză obiectivă a performanței fiecărei metode. Compararea cu o soluție naivă a oferit un punct de referință util pentru a înțelege valoarea comportamentelor învățate.

Observații finale

Rezultatele obținute arată că ambele metode sunt capabile să învețe comportamente funcționale în jocul Snake. Agentul PPO s-a remarcat prin consistență și eficiență, reușind să finalizeze frecvent jocul cu scor maxim și un număr redus de pași. De cealaltă parte, agentul bazat pe algoritmul genetic a atins scoruri mari în unele episoade, dar cu o rată de succes mai mică și o variabilitate crescută, în special în situații complexe.

Soluția naivă, deși garantează atingerea scorului maxim, evidențiază limitele unei strategii rigide: finalizarea cu succes este obținută în mod sigur, dar într-un

număr mult mai mare de pași și fără capacitatea de adaptare la situații neprevăzute.

Diferențele dintre cele trei abordări nu s-au reflectat doar în scor, ci și în stilul de joc adoptat. Astfel, alegerea metodei potrivite depinde în mare măsură de obiectivele urmărite: dacă se dorește maximizarea scorului cu un comportament sigur și robust, PPO este cea mai echilibrată soluție. În schimb, pentru explorare și diversitate comportamentală, algoritmul genetic oferă un cadru flexibil, iar soluția naivă rămâne un reper de siguranță absolută, dar ineficientă.

Direcții viitoare

O posibilă continuare a acestei lucrări ar fi testarea agenților în medii mai complexe, cum ar fi hărți mai mari sau cu mai multe mere simultan. De asemenea, se poate introduce o competiție între mai mulți șerpi controlați de agenți diferiți, fiecare urmărind să colecteze cât mai multe resurse. Un astfel de cadru ar permite evaluarea comportamentului în scenarii competitive și ar evidenția diferențele de strategie între metode. Alte direcții includ îmbunătățirea funcției de recompensă, analizarea capacității de generalizare pe structuri de joc variate sau înlocuirea arhitecturii dense actuale cu rețele neuronale convoluționale, care ar putea extrage automat trăsături relevante dintr-o reprezentare vizuală a hărții și ar reduce nevoia de preprocesare manuală a stării.

Bibliografie

[1] ezra anderson, *A.I Learns Snake And Wins - Part 1*, 2022, URL: https://www.youtube.com/watch?v=G8NEj5InvVA&t=19s.

[2] Chrispresso, *AI Learns to play Snake!*, 2020, URL: https://github.com/Chrispresso/SnakeAI.

[3] Google DeepMind, *DeepMind x UCL | Deep Learning Course 2018*, 2022, URL: https://youtube.com/playlist?list=PLqYmG7hTraZCkftCvihsG2eCTH2OyGScc&si=QcuN87cDT6nFGIpV.

[4] GeeksforGeeks, *Crossover in Genetic Algorithm*, GeeksforGeeks, 2020, URL: https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/.

[5] Cal Jeffrey, *Coder shrinks classic Snake game down to just 56 bytes*, TechSpot, 2024, URL: https://www.techspot.com/news/106339-coder-shrinks-classic-snake-game-down-56-byte.html.

[6] Patrick Loeber, *Snake AI with PyTorch*, 2020, URL: https://github.com/patrickloeber/snake-ai-pytorch.

[7] Machine Learning with Phil, *Proximal Policy Optimization (PPO) is Easy With PyTorch | Full PPO Tutorial*, 2020, URL: https://www.youtube.com/watch?v=hlv79rcHws0&t=3152s.

[8] Richard S. Sutton și Andrew G. Barto, *Reinforcement Learning: An Introduction*, a 2-a ed., MIT Press, 2018, URL: http://incompleteideas.net/book/the-book-2nd.html.

[9] Phil Tabor, *Proximal Policy Optimization (PPO) Implementation in PyTorch*, 2021, URL: https://github.com/philtabor/Youtube-Code-Repository/tree/master/ReinforcementLearning/PolicyGradient/PPO/torch.