UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

REINFORCEMENT LEARNING ÎN JOCURI VIDEO

Absolvent
Totolici Alexandru-Gabriel

Coordonator științific
Conf. dr. Ciprian Ionuț Păduraru

București, iunie 2024

Rezumat

Progresele recente în inteligența artificială au transformat dezvoltarea jocurilor video și a aplicațiilor de simulare. Alegerea acestei teme este motivată de potențialul de a crea scenarii dinamice și competitive, unde agenții colaborează și concurează în echipe, demonstrând abilități avansate de luare a deciziilor și adaptabilitate.Scopul acestei lucrări este de a arăta cum aceste tehnologii pot îmbunătăți experiența de joc și realismul simulărilor. Utilizând Unity ML-Agents pentru antrenarea agenților care interacționează cu mediul de joc, aceștia învață să anticipeze mișcările adversarilor și să colaboreze eficient cu coechipierii lor. Agenții inteligenți pot adapta strategiile în funcție de stilul de joc al adversarilor, menținând un nivel optim de dificultate.

Abstract

Recent advances in artificial intelligence have transformed the development of video games and simulation applications. The choice of this theme is motivated by the potential to create dynamic and competitive scenarios, where agents collaborate and compete in teams, demonstrating advanced decision-making skills and adaptability. The purpose of this paper is to demonstrate how these technologies can improve the gaming experience and the realism of simulations. By using Unity ML-Agents to train agents that interact with the gaming environment, they learn to anticipate opponents' moves and collaborate efficiently with their teammates. Intelligent agents can adapt their strategies based on opponents gameplay styles, maintaining an optimal level of difficulty.

Cuprins

Capitolul 1

Introducere și Motivație

1.1 Introducere

În numeroase domenii ale societății, inteligența artificială a avut un impact semnificativ, oferind soluții inovatoare pentru probleme complexe și optimizând procese într-o manieră care nu a fost posibilă anterior. Acest progres remarcabil este susținut de o gamă largă de tehnici de învățare automată, care au fost adaptate pentru a satisface cerințele specifice ale diferitelor scenarii de utilizare.[19]

Reinforcement learning este o ramură a învățării automate care permite sistemelor informatice să învețe să ia decizii folosind recompense pe măsură ce interacționează cu mediul lor. În „RL", un agent autonom își ajustează comportamentul prin recompense, fără a avea nevoie de exemple de răspunsuri corecte pre-etichetate; în schimb, învățarea supervizată antrenează modelele folosind un set de date cu răspunsuri corecte deja cunoscute.[18]

Agenții de învățare prin întărire sunt folosiți într-o varietate de domenii practice, demonstrând capacitatea lor de a învăța și optimiza comportamente în medii complexe. În industria jocurilor, aceștia îmbunătățesc provocările și interacțiunile, oferind o experiență de joc dinamică. Vehiculele autonome utilizează aceste tehnici pentru a naviga în condiții de trafic variabile și a lua decizii de conducere sigure. În robotică, ajută la efectuarea sarcinilor precise și la adaptarea la medii noi. În sectorul telecomunicațiilor, se utilizează învățarea prin întărire pentru a optimiza gestionarea traficului și resurselor de rețea, în timp ce în finanțe, acești agenți dezvoltă strategii de tranzacționare automatizate care răspund rapid la schimbările de piață. În sănătate, este aplicat pentru personalizarea tratamentelor, ajustând terapiile în funcție de răspunsul individual al pacienților.[17]

Vom examina modul în care acest tip de agenți interacționează cu mediul înconjurător într-un ciclu continuu de feedback. Obiectivul acestei interacțiuni este de a optimiza comportamentul agentului prin maximizarea recompenselor pe care le primește. Pentru a descrie procesul, există patru pași esențiali, așa cum este ilustrat în diagrama prezentată:

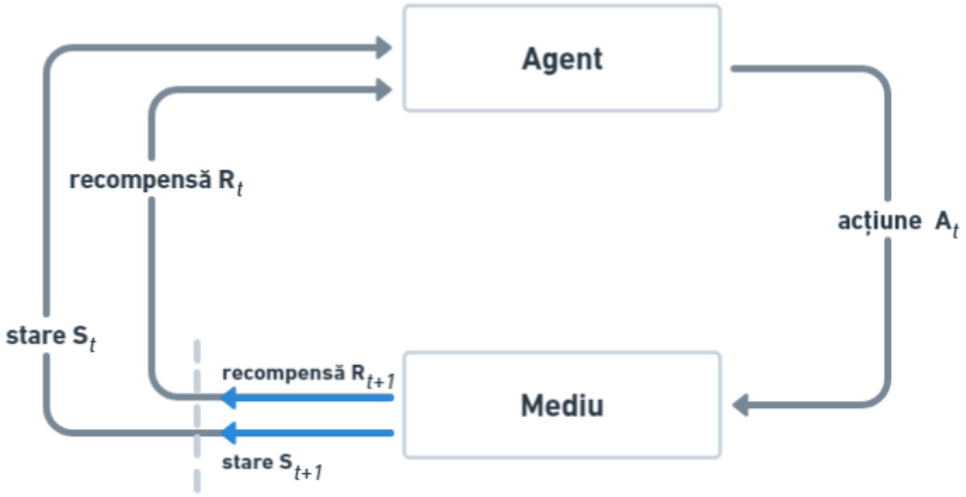

Figura 1.1: Procesul de învățare prin întărire

Unul dintre instrumentele tot mai folosite în cercetarea și implementarea algoritmilor de reinforcement learning, in jocuri video, este Unity, prin intermediul extensiei ML-Agents. Unity este un motor grafic avansat care oferă o platformă puternică pentru simularea și vizualizarea comportamentelor complexe în medii controlate. Un set de instrumente dezvoltat de Unity Technologies, ML-Agents permite cercetătorilor să testeze și să îmbunătățească modele de învățare în scenarii vizuale realiste. Acest lucru facilitează integrarea acestor algoritmi în medii virtuale. Unreal Engine este un alt motor grafic folosit pentru implementarea algoritmilor de învățare prin întărire în jocuri. Acesta permite integrarea și testarea modelelor în scenarii vizuale complexe. Utilizând diferite extensii, dezvoltatorii pot crea agenți inteligenți care operează în medii detaliate, extinzând aplicațiile posibile în simulări, VR și AR.[16]

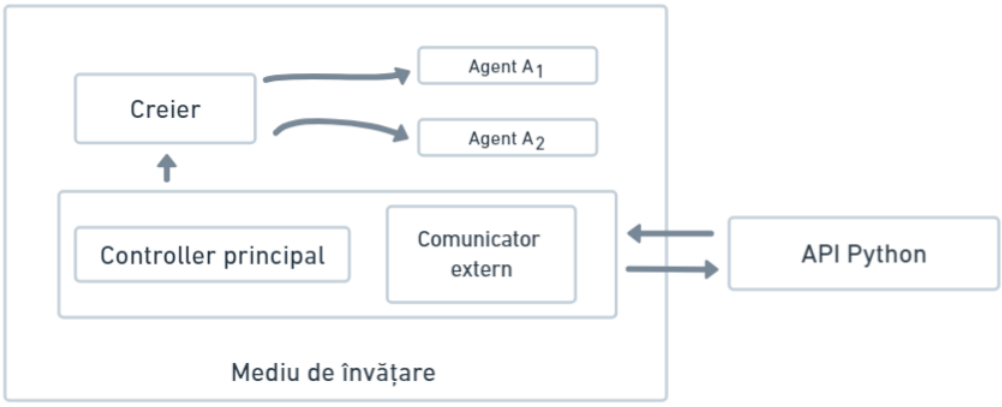

Figura 1.2: Mediu de învățare utilizat pentru antrenarea agenților de reinforcement learning

Vom folosi un exemplu ipotetic pe parcursul discuției pentru a ilustra materialul și terminologia prezentată în acest document. Vom discuta problema antrenării comportamentului unui personaj non-jucabil (NPC). Să presupunem, de exemplu, că dezvoltăm un joc, în care jucătorii controlează diferite personaje. În acest joc, avem un singur NPC care servește ca gardian, având responsabilitatea de a păzi anumite obiective și de a interveni în situații de conflict. Să presupunem că există două echipe, fiecare cu cinci jucători și un NPC gardian.[15]

Comportamentul unui gardian este destul de complex. În primul rând, trebuie să evite să fie rănit, ceea ce necesită detectarea situațiilor periculoase și deplasarea într-o locație sigură. În al doilea rând, trebuie să fie conștient de atacurile care se petrec în jurul său și să decidă când și cum să intervină. În cazul mai multor conflicte simultane, trebuie să evalueze gravitatea fiecărei situații și să decidă unde să intervină mai întâi. În cele din urmă, un gardian eficient va ocupa întotdeauna o poziție strategică, de unde poate interveni rapid pentru a-și proteja obiectivele. Luând în considerare toate aceste trăsături, gardianul trebuie să măsoare mai multe atribute ale mediului, de exemplu , poziția membrilor echipei, poziția inamicilor, locațiile conflictelor active și apoi să decidă asupra unei acțiuni: să se ascundă de focul inamic sau să se deplaseze pentru a interveni într-un conflict. Având în vedere numărul mare de setări ale mediului și numărul mare de acțiuni pe care gardianul le poate întreprinde, definirea și implementarea acestor comportamente complexe manual este o provocare și predispusă la erori.

Cu ML-Agents, este posibil să antrenăm comportamentele unor astfel de NPC-uri, numiți agenți, folosind o varietate de metode. Ideea de bază este destul de simplă. Trebuie să definim trei entități în fiecare moment al jocului:

Gardianul colectează observații numerice și vizuale pentru a înțelege mediul înconjurător. Observațiile numerice reflectă pozițiile și acțiunile obiectivelor vizibile, în timp ce observațiile vizuale sunt generate de camerele atașate, oferind imagini directe din mediul de joc. Gardianul poate efectua acțiuni discrete sau continue, variind de la mișcări simple în patru direcții până la navigare complexă cu ajustări de viteză și direcție, în funcție de complexitatea scenariului de joc. Gardianul primește recompense bazate pe performanța acțiunilor sale, cu valori pozitive pentru intervenții reușite și penalități pentru eșecuri sau incapacitatea de a proteja obiectivele.

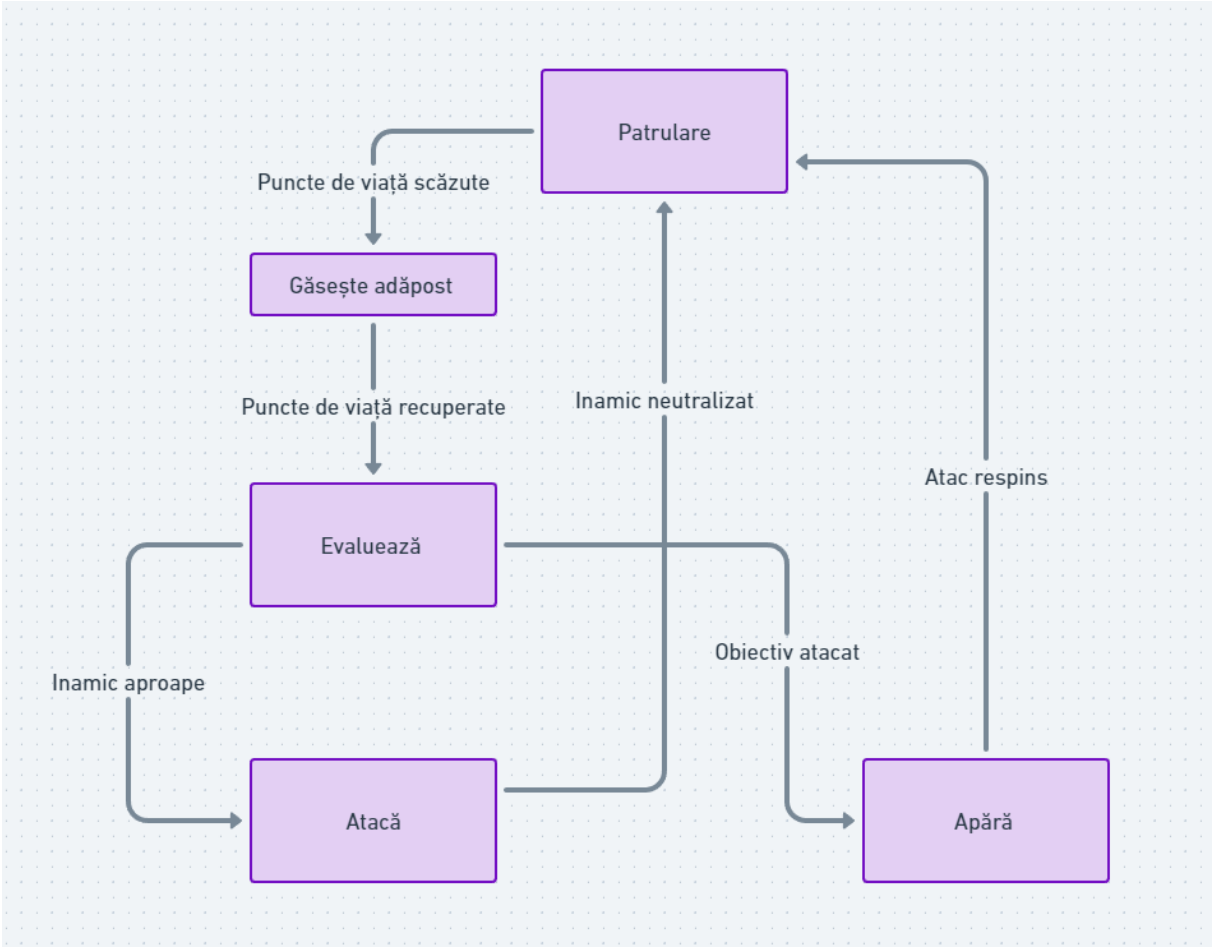

Figura 1.3: Procesul de decizie al NPC-ului

1.2 Motivație

În concluzie, ne propunem să explorăm potențialul Unity ML-Agents ca platformă pentru implementarea și testarea algoritmilor de reinforcement learning. Ne vom concentra în mod specific pe aplicarea acestor algoritmi în dezvoltarea unui joc de fotbal virtual, unde vom antrena jucători capabili să execute tactici de joc eficiente în diferite configurații de meci, cum ar fi 2v2 sau 3v3. Scopul este de a antrena agenți care să colaboreze, să se poziționeze strategic și să ia decizii rapide în timpul jocului, simulând astfel comportamentul jucătorilor umani într-un mediu competitiv. Prin utilizarea Unity ML-Agents, vom putea simula și analiza performanța acestor algoritmi într-un mediu controlat și vizual interactiv, oferind o platformă ideală pentru demonstrarea capabilităților de învățare și adaptare ale agenților inteligenți. În această lucrare vom testa eficacitatea tehnicilor de reinforcement learning în antrenarea comportamentelor complexe, dar vom oferi și o bază solidă pentru explorări ulterioare în optimizarea și scalarea acestor tehnici pentru alte aplicații de gaming sau simulări realiste.

Capitolul 2

Related Work

2.1 Unity - motor de cercetare în inteligența artificială

In lucrarea "Unity: A General Platform for Intelligent Agents," Arthur Juliani și colaboratorii săi explorează necesitatea platformelor de simulare complexe și scalabile pentru cercetarea în domeniul inteligenței artificiale. În ultimii ani, progresele semnificative în cercetarea învățării prin consolidare profundă și designul algoritmilor au fost facilitate de existența unor platforme de simulare provocatoare și scalabile. Cu toate acestea, multe dintre platformele existente prezintă limitări, cum ar fi vizualuri nerealiste, fizică inexactă, complexitatea scăzută a sarcinilor și perspectiva limitată a agenților. Autorii subliniază că aceste limitări împiedică dezvoltarea de algoritmi și soluții avansate în domeniul AI. Pentru a depăși aceste constrângeri, lucrarea propune utilizarea motoarelor de jocuri moderne, precum Unity, ca platforme generale care permit dezvoltarea de medii de învățare bogate din punct de vedere vizual, fizic, al logicii sarcinilor și social. Unity, un motor de jocuri bine cunoscut, se dovedește a fi un instrument ideal datorită capacității sale de a genera medii complexe și variate, oferind cercetătorilor un cadru robust pentru experimentare și inovare.[20]

2.1.1 Complexitatea mediiilor de simulare

Unul dintre punctele forte ale Unity este capacitatea sa de a crea medii de simulare cu un grad ridicat de complexitate senzorială și fizică. De exemplu, motorul Unity permite randarea grafică de înaltă calitate și simularea unor fenomene fizice complexe, cum ar fi dinamica corpurilor rigide și moi, particule și fluide. Aceasta este esențială pentru cercetarea învățării prin consolidare, unde agenții trebuie să interacționeze cu mediul lor într-un mod realist pentru a învăța comportamente relevante și transferabile în lumea reală.

2.1.2 Capacitatea de control și simulare distribuită

Unity oferă, de asemenea, un nivel ridicat de control asupra configurării simulărilor, permițând cercetătorilor să modifice dinamica mediului în timpul rulării experimentelor. Această flexibilitate este crucială pentru metodele avansate de învățare, cum ar fi învățarea "curriculum", unde complexitatea sarcinilor crește treptat pe măsură ce agenții devin mai performanți. De asemenea, Unity poate rula simulări rapid și distribuit, ceea ce permite colectarea unui volum mare de date într-un timp scurt, accelerând astfel procesul de antrenare al algoritmilor AI.

2.1.3 Avantajele utilizării Unity ML-Agents Toolkit

Toolkit-ul ML-Agents dezvoltat de Unity facilitează integrarea algoritmilor de învățare cu mediile de simulare create în Unity. Acesta oferă exemple de medii, algoritmi de învățare de ultimă generație, cum ar fi Proximal Policy Optimization și Soft Actor-Critic, precum și suport pentru învățarea prin imitare și auto-joc. Prin utilizarea acestui toolkit, cercetătorii pot defini cu ușurință medii personalizate și pot implementa diverse strategii de antrenare pentru agenții lor, permițând o explorare vastă a posibilităților în domeniul inteligenței artificiale.

2.1.4 Concluzie

Lucrarea subliniază că Unity, cu capabilitățile sale extinse și flexibilitatea oferită de ML-Agents Toolkit, reprezintă o platformă generală ideală pentru cercetarea în inteligența artificială, permitand crearea de medii de simulare care reflectă complexitatea și dinamica lumii reale, esențiale pentru dezvoltarea algoritmilor avansați de învățare. Aceasta deschide noi direcții de cercetare și facilitează inovarea continuă, contribuind la progresul semnificativ al domeniului inteligenței artificiale.

2.2 Reinforcement Learning aplicat în jocuri competitive bazate pe fizică

Lucrarea "Reinforcement Learning for Physics-Based Competitive Games" scrisă de Abdulla R. Albuainain și Christos Gatzoulis explorează utilizarea învățării prin întărire pentru dezvoltarea agenților inteligenți capabili să se adapteze la medii competitive bazate pe fizică. Autorii subliniază dificultățile întâmpinate în programarea agenților AI tradiționali, în special în jocurile cu fizică aplicată, unde spațiile de stare sunt vaste și interacțiunile complexe. Abordarea lor propune utilizarea acestei tehnici pentru a crea agenți care pot învăța din experiență și care sunt capabili să îndeplinească compor-

tamente intrinseci, cum ar fi apărarea unui obiectiv și atacul asupra mingii, în jocuri competitive.[21]

2.2.1 Complexitatea jocurilor

Unul dintre principalele puncte discutate în lucrare este complexitatea mediiilor bazate pe fizică, care prezintă un număr imens de stări posibile și strategii de joc. Într-un astfel de mediu, agenții pot adopta diverse stiluri de joc, cum ar fi jocul defensiv, așteptând greșelile adversarului, sau jocul agresiv, forțând greșelile acestuia. Aceste variabile fac dificilă crearea unui model bazat pe reguli prestabilite, capabil să se comporte inteligent și credibil. Lucrarea evidențiază că agenții reușesc să învețe și să se adapteze la dinamica schimbătoare a mediului, prin ajustarea funcțiilor de recompensă pentru a modela comportamente care îmbunătățesc performanța generală a agentului.

2.2.2 Implementări în jocuri competitive

Implementarea acestei tehnici într-un joc bazat pe fizică, inspirat de Rocket League, demonstrează potențialul său. Proiectul a fost realizat folosind motorul de jocuri Unity și ML-Agents Toolkit, care oferă un cadru suficient pentru dezvoltarea și antrenarea agenților. Inițial, mediul de joc și acțiunile agenților au fost simplificate pentru a testa eficacitatea funcțiilor de recompensă. Configurația inițială a inclus recompense pentru atingerea mingii și penalizări pentru fiecare pas realizat, în scopul de a încuraja comportamentele de atac și apărare corecte.

2.2.3 Optimizarea agenților și a funcțiilor de recompensă

Pe parcursul experimentelor, agenții au învățat să prioritizeze atingerea mingii, dar deseori au lovit mingea fără a direcționa loviturile către poarta adversarului. Pentru a corecta acest comportament, funcțiile de recompensă au fost ajustate, introducând recompense suplimentare pentru lovirea mingii în direcția corectă și penalizări pentru lovirea acesteia către propria poartă. De asemenea, hiperparametrii rețelelor neuronale au fost optimizați, ajustând numărul de unități ascunse și straturi pentru a îmbunătăți performanța agenților.

2.2.4 Evaluarea credibilității agenților

Un aspect important al studiului a fost evaluarea credibilității agenților antrenați. Testele au fost realizate printr-un sondaj în care participanții au fost rugați să distingă între modele de inteligență artificială și jucători umani pe baza comportamentului observat în videoclipuri. Rezultatele sondajului au arătat că agenții antrenați au fost greu de distins de jucătorii umani, demonstrând că aceștia au dobândit comportamente credibile

și realiste. Agenții cu funcții de recompensă mai sofisticate au învins versiunile anterioare, arătând că o definire detaliată a mecanismelor de recompensă poate îmbunătăți semnificativ performanța agenților.

2.2.5 Concluzie

Lucrarea de față demonstrează că utilizarea învățării prin întărire pentru antrenarea agenților în jocuri competitive bazate pe fizică poate produce agenți credibili și eficienți. Optimizarea funcțiilor de recompensă și a hiperparametrilor este esențială pentru dezvoltarea comportamentelor dorite. Studiul sugerează că această tehnică poate depăși limitările inteligenței artificiale tradiționale, oferind o abordare robustă și scalabilă pentru dezvoltarea agenților în medii complexe. Acest model poate fi extins la medii 3D și poate include tehnici de învățare prin imitare pentru a crea comportamente și mai umane, deschizând noi direcții de cercetare și aplicare în domeniul inteligenței artificiale.

2.3 Deep reinforcement learning în Unity

Abhilash Majumder, în lucrarea sa "Deep Reinforcement Learning in Unity", explorează complexitatea și eficiența implementării algoritmilor de învățare prin întărire în cadrul platformei Unity, folosind Unity ML Agents. Această lucrare marchează o contribuție semnificativă la literatura existentă, demonstrând aplicabilitatea tehnicilor avansate de învățare automată în dezvoltarea și antrenarea agenților inteligenți în medii simulate. Lucrarea evidențiază capacitatea îmbunătățită de generalizare a agenților, care le permite să se adapteze rapid și eficient la noi scenarii și medii necunoscute.[22]

2.3.1 Detalii tehnice și implementări

Majumder implementează algoritmi de învățare prin întărire, cum ar fi Proximal Policy Optimization, pentru a demonstra cum agenții pot acumula experiență și învăța din interacțiunile complexe din medii controlate, transferând aceste cunoștințe pentru a face față provocărilor din medii noi. Discuțiile sale includ și stabilizarea procesului de învățare, unde tehnicile implementate ajută la reducerea variațiilor în performanța agenților pe parcursul sesiunilor de antrenament. Aceasta este o realizare semnificativă, deoarece stabilizarea procesului de învățare asigură un progres mai predictibil și mai consecvent în dezvoltarea agenților, facilitând evaluarea și ajustarea continuă a parametrilor algoritmului.

2.3.2 Eficiența agenților în timp real

Lucrarea pune un accent deosebit pe eficiența în timp real a agenților inteligenți, o caracteristică esențială pentru aplicațiile care necesită răspunsuri rapide și acțiuni coordonate în scenarii dinamice, cum ar fi jocurile video sau simulările de formare. Eficiența în timp real permite agenților să execute decizii și acțiuni într-un mod fluid și natural, contribuind la crearea unei experiențe de utilizator mai imersive și mai realiste. Optimizările algoritmice, conform lui Majumder, pot reduce latența și pot îmbunătăți capacitatea de procesare, făcând agenții capabili să funcționeze eficient în cadrul limitărilor hardware și software ale sistemelor pe care sunt implementați.

2.3.3 Aplicabilitate în scenarii competitive și colaborative

Majumder explorează și aplicabilitatea acestor tehnici în scenarii competitive și colaborative, unde agenții trebuie să interacționeze nu doar cu mediul, ci și unii cu alții. Experimentele prezentate ilustrează cum agenții antrenați cu tehnici de învățare curriculum și joc adversarial pot dezvolta strategii complexe și pot colabora sau concura eficient. Acest aspect deschide noi perspective pentru dezvoltarea de jocuri multiplayer și simulări de antrenament, unde comportamentul adaptativ și cooperarea sunt esențiale.

2.3.4 Concluzie

În concluzie, lucrarea lui Majumder oferă o analiză detaliată și tehnică a potențialului metodelor avansate de învățare automată în îmbunătățirea dezvoltării și implementării agenților în medii simulate. Rezultatele sale subliniază importanța continuării cercetărilor în acest domeniu și extinderea aplicabilității acestor tehnologii în industria jocurilor și dincolo de aceasta. Prin demonstrarea succesului în aplicarea acestor tehnici, Majumder contribuie semnificativ la avansarea înțelegerii și utilizării învățării automate, oferind o resursă valoroasă pentru cercetători, dezvoltatori și profesioniști din industrie.

2.4 Impactul învățării curriculum în antrenarea agenților

În lucrarea lui Rigoberto Sáenz și Jorge E. Camargo, "Evaluating the impact of curriculum learning on the training process for an intelligent agent in a video game", autorii investighează eficacitatea învățării curriculum aplicate în antrenarea agenților inteligenți pentru jocuri video, folosind reinforcement learning. Studiul se concentrează pe două abordări diferite de antrenament utilizând învățarea curriculum, evidențiind cum variațiile în dificultatea mediului de antrenament și recompensele modificate afectează performanța agenților.[23]

2.4.1 Impactul vitezei adversarului în performanța agenților

Un aspect crucial al studiului este observația că performanța agentului crește semnificativ când adversarul este absent sau când viteza acestuia este redusă. Inițial, când agentul se antrenează fără prezența adversarului sau cu un adversar încetinit, acesta înregistrează performanțe mai bune. Această situație se schimbă pe măsură ce adversarul începe să se miște mai rapid; performanța agentului scade cu fiecare incrementare a vitezei adversarului, începând de la 50% din meciuri și continuând să scadă pe măsură ce se apropie de finalul antrenamentului. În scenariul în care agentul este antrenat doar cu algoritmul PPO fără un curriculum structurat, recompensa medie cumulativă obținută este de aproximativ 0.45 la sfârșitul antrenamentului. Pe de altă parte, utilizând un curriculum de învățare, scorul scade la 0.35, indicând faptul că introducerea curriculum-ului în acest caz specific nu a îmbunătățit performanța și, de fapt, pare să o împiedice. Este important de menționat că, comparația directă dintre cele două abordări este posibilă doar după 90% din meciurile jucate, când ambele experimente funcționează în condiții identice de mediu, permițând ambilor agenți să se miște la aceeași viteză.

2.4.2 Abordarea alternativă cu recompense incrementale

O a doua abordare explorează efectul recompenselor incrementale pentru atingerea mingii. În această configurație, agentul primește inițial o recompensă suplimentară de 0.3 de fiecare dată când atinge mingea, cu o reducere treptată a acestei recompense cu 0.1 la fiecare 10% din totalul meciurilor, până când nu mai există nicio recompensă suplimentară acordată după 30% din antrenament. Această strategie pare să aibă un efect pozitiv asupra performanței agentului, asigurând o îmbunătățire a recompensei medii cumulative comparativ cu experimentul de control. Rezultatele pentru această metodă indică o îmbunătățire semnificativă a performanței, cu agentul depășind linia de recompensă medie de 0.5 mult mai devreme în procesul de antrenament comparativ cu scenariul anterior. Acest lucru sugerează că acordarea de recompense suplimentare pentru interacțiuni specifice cu mingea poate accelera procesul de învățare, permițând agentului să atingă obiectivele de performanță dorite într-un timp mai scurt.

2.4.3 Concluzie

Aceste observații subliniază complexitatea și sensibilitatea antrenării agenților inteligenți în medii simulate. Efectele învățării curriculum variază în funcție de parametrii specifici ai mediului și de structura recompenselor. În timp ce ajustările în viteza adversarului au prezentat provocări, modificările strategice ale sistemului de recompense.

2.5 Abordarea mediului SoccerTwos utilizând algoritmi clasici

Lucrarea "An ML Agent using the Policy Gradient Method to win a SoccerTwos Game" de Victor Ulisses Pugliese explorează utilizarea metodelor de învățare în jocul SoccerTwos. Autorul și colaboratorii săi investighează metodele de optimizare a politicilor utilizând învățarea prin curriculum, pentru a crea agenți de învățare automată care să câștige meciuri în fața adversarilor. Această lucrare subliniază importanța metodelor avansate de reinforcement learning" în contextul antrenării agenților pentru jocuri video complexe.[24]

2.5.1 Utilizarea metodelor - policy gradient

Metodele de tip "policy gradient" sunt tehnici de învățare prin consolidare care optimizează politicile parametrizate în funcție de recompensa cumulată pe termen lung, prin coborârea gradientului. În această lucrare, autorul explorează două astfel de metode: Proximal Policy Optimization și Soft Actor-Critic. PPO este o metodă on-policy care antrenează o politică stocastică, explorând prin eșantionarea acțiunilor conform celei mai recente versiuni a politicii sale stocastice. SAC, pe de altă parte, optimizează o politică stocastică într-un mod off-policy, maximizând recompensa așteptată și entropia pentru a asigura o explorare eficientă și o învățare accelerată.

2.5.2 Rezultate și concluzii

Studiul a arătat că algoritmul PPO combinat cu învățarea curriculum a obținut cele mai bune rezultate în mediul "SoccerTwos", convergând mai rapid și obținând o recompensă medie superioară față de alte metode. Agenții antrenați cu această metodă au demonstrat abilități avansate de joc, precum repoziționarea defensivă și ofensivă și cooperarea pentru a marca goluri. Cu toate acestea, experimentele cu algoritmul SAC nu au reușit să aibă rezultatele așteptate. Lucrarea concluzionează că învățarea prin întărire cu gradient de politică, combinată cu învățarea curriculum, poate produce agenți de joc eficienți și competitivi în medii bazate pe fizică.

2.6 Abordarea mediului SoccerTwos utilizând algoritmi POCA

În lucrarea "Unity ML Agents: Wall Jump and SoccerTwos environment using Reinforcement Learning (RL) tecnique," autorii Viktória Brigitta Ilosvay și Emanuele Iaccarino explorează aplicarea tehnicilor de învățare prin consolidare folosind platforma Unity. Scopul principal este de a eficientiza algoritmul POCA, plecând de la modelul de bază oferit de Unity și aplicându-l în medii personalizate precum SoccerTwos.[25]

2.6.1 Perspectivele antrenării

Folosind setările de bază furnizate de Unity pentru procesul de antrenament al algoritmului POCA în mediul "SoccerTwo", modelul a fost antrenat pentru aproximativ 50 de milioane de pași. Evaluarea performanței agenților a fost realizată folosind sistemul de rating ELO, un mod eficient de a evalua nivelul de abilitate al agenților în jocuri competitive.

2.6.2 Optimizarea hiperparametrilor

Pentru a îmbunătăți eficiența modelului de învățare prin consolidare, autorii au făcut modificări importante ale hiperparametrilor, ghidate de observațiile obținute prin analiza TensorBoard. Aceste ajustări includ creșterea ratei de învățare de la 0.0003 la 0.0007 și trecerea de la o rată fixă la una liniară pentru a se adapta mai bine pe măsură ce antrenamentul progresează. În strategia de explorare, s-a trecut de la o rată constantă de explorare la una descrescătoare liniar, permițând mai multă explorare la început și mai multă exploatare pe măsură ce performanța modelului se îmbunătățește. Rețeaua neurală a fost modificată de la 2 la 5 straturi pentru a îmbunătăți capacitatea modelului de a procesa sarcini complexe. Valoarea gamma a fost crescută pentru a sublinia importanța recompenselor viitoare și a încuraja modelul să dezvolte strategii care consideră succesul pe termen lung. Setările de auto-joc au fost rafinate, mărind fereastra la 20 și crescând numărul de pași salvați la 200.000, pentru a oferi un set de date mai larg pentru model și a spațializa evaluările performanței.

2.6.3 Observații dupa optimizarea hiperparametrilor

Graficul "Smoothed ELO Value" a arătat o valoare ELO mai bună, indicând o îmbunătățire constantă a performanței agenților, modelul cu hiperparametri modificați având o valoare ELO mai ridicată și o variabilitate redusă. O rată de învățare inițial mai mare a permis modelului să învețe mai rapid, iar scăderea entropiei a sugerat că agentul a devenit mai sigur în acțiunile sale pe măsură ce antrenamentul a progresat. Adâncirea rețelei

neuronale a îmbunătățit capacitatea modelului de a gestiona sarcini complexe, rezultând într-o predicție mai precisă a valorilor viitoare și o scădere a pierderilor de valoare. Ajustările aduse setărilor de auto-joc au permis agenților să dezvolte strategii mai robuste și adaptabile, ceea ce a dus la o îmbunătățire a performanței generale.

2.6.4 Concluzie

Lucrarea concluzionează că algoritmul POCA este unul eficient și robust pentru antrenarea agenților în medii multi-agent complexe. Modificările hiperparametrilor au dus la o îmbunătățire semnificativă a performanței, evidențiată printr-o valoare ELO mai ridicată și o variabilitate redusă. Rezultatele sugerează că acest algoritm poate fi aplicat cu succes în diverse domenii, inclusiv jocuri video și robotică, unde coordonarea între multiple entități autonome este esențială. În viitor, cercetările ar trebui să continue pentru a explora aplicații în medii și mai complexe și pentru a optimiza în continuare hiperparametrii algoritmului.

2.7 Impactul eterogenității agenților asupra învățării

Lucrarea "Impact of Heterogeneity on Multi-Agent Reinforcement Learning" de Rodrigo Fonseca Marques și Zenilton Kleber Gonçalves do Patrocínio Júnior explorează efectele eterogenității asupra învățării multi-agent. Majoritatea studiilor utilizează agenți omogeni, unde toți agenții sunt identici. Totuși, în aplicațiile reale, agenții au adesea seturi de abilități similare, dar cu grade diferite de intensitate. Această lucrare propune un model nou în care agenții eterogeni împărtășesc același set de abilități, dar cu intensități variate, pentru a reflecta mai bine scenariile din viața reală.[26]

2.7.1 Modelul propus pentru agenți eterogeni

Modelul propus consideră agenți cu aceiași senzori și actuatori, dar cu nivele diferite de i control. De exemplu, într-un joc de "SoccerTwos", un agent poate avea o viziune periferică mai bună, în timp ce altul poate avea o viteză de rotație mai mare. Această variație permite agenților să se specializeze în anumite roluri, unul devenind mai eficient în apărare, iar altul în atac. Aceasta reflectă scenariile reale, unde jucătorii, deși au abilități similare, se diferențiază prin intensitatea acestora.

2.7.2 Configurația experimentelor

Experimentele au fost realizate folosind Unity ML-Agents Toolkit, testând performanța agenților în mediile SoccerTwos și Tennis. Au fost evaluate patru configurații de

agenți: configuratia de bază (agenți omogeni), modificarea senzorilor (S-Mod), modificarea actuatorilor (A-Mod) și modificarea combinată a senzorilor și actuatorilor (SA-Mod). Rezultatele au arătat că agenții eterogeni au obținut performanțe superioare comparativ cu cei omogeni. În acest mediu, agenții eterogeni au învățat să se organizeze eficient, cu unul dintre agenți concentrându-se pe apărare și celălalt pe atac, ceea ce a dus la o cooperare mai bună și la o creștere semnificativă a scorurilor.

2.7.3 Concluzie

Studiul demonstrează că agenții eterogeni, care împărtășesc un set comun de abilități dar cu intensități diferite, pot obține rezultate mai bune decât agenții omogeni în sistemele multi-agent. Acest lucru se datorează capacității lor de a se specializa și de a învăța din experiențe variate și unice. Modelul propus oferă o nouă posibilitate de îmbunătățire a antrenamentului agenților multipli, fiind aplicabil în scenarii reale unde agenții nu pot avea abilități complet diferite, dar se diferențiază prin intensitatea acestora.

Capitolul 3

Tehnologii si Fundamente teoretice

3.1 Tehnologii

Pentru a crea experiențe de joc din ce în ce mai sofisticate și imersive, recurgem la inovații în învățarea automată și inteligența artificială pe măsură ce explorăm frontierele tehnologice în dezvoltarea jocurilor video. În acest sens, punctul culminant al acestui efort este antrenarea agenților de cel mai înalt nivel într-un mediu virtual de fotbal cu Unity ML-Agents, care utilizează tehnologiile avansate și programele software pentru a instrumenta întregul proces, de la crearea mediului de joc, la antrenarea agenților și la vizualizarea și analiza rezultatelor.

3.1.1 Unity

Unity este un motor grafic extensiv utilizat nu doar pentru dezvoltarea de jocuri, ci și pentru implementarea tehnologiilor de inteligență artificială prin Unity ML-Agents, oferind un mediu ideal pentru simulări detaliate și interacțiuni AI complexe. Permite crearea de medii de simulare variate, de la terenuri de fotbal la orașe complexe, facilitând antrenarea agenților prin algoritmi de învățare prin întărire. Caracteristici precum animații fluide, un sistem avansat de particule, scripting flexibil în C#, și unelte puternice de debugging și visualizare în editorul Unity sunt esențiale pentru dezvoltarea și testarea agenților AI. Aceste capabilități permit dezvoltatorilor să creeze scenarii realiste și aprofundate, optimizând performanța și eficiența antrenamentelor într-un cadru controlat și adaptabil, esențial pentru antrenarea eficientă a agenților inteligenți în jocuri video, cum ar fi fotbalul virtual, unde fiecare aspect al jocului poate fi ajustat și optimizat.[14]

3.1.2 Unity Hub

Unity Hub este un instrument esențial pentru gestionarea și organizarea proiectelor și versiunilor Unity, oferind o interfață centralizată care simplifică navigarea între diferitele aspecte ale dezvoltării în Unity. Acesta permite utilizatorilor să instaleze multiple versiuni ale editorului Unity, să gestioneze licențele, să acceseze proiecte, să descarce resurse suplimentare din Asset Store și să lanseze tutoriale. De asemenea, Unity Hub este util pentru configurarea mediilor de dezvoltare, asigurând că toate proiectele sunt compatibile cu versiunile corecte ale software-ului Unity și facilitând colaborarea în echipe prin păstrarea unei structuri organizate a proiectelor și dependențelor. Această unealtă este vitală pentru dezvoltatorii care lucrează cu versiuni multiple și complexe de proiecte Unity, ajutând la menținerea eficienței și la reducerea potențialelor erori de compatibilitate, fiind un aliat de încredere în orice pipeline de dezvoltare de jocuri sau aplicații interactive.[13]

3.1.3 Python

Python este un limbaj de programare de nivel înalt, interpretat și orientat pe obiecte, recunoscut pentru sintaxa sa simplă și lizibilă, care facilitează scrierea de cod clar și logic cu mai puține linii decât ar fi necesar în alte limbaje. Este extrem de popular în rândul comunității de dezvoltatori datorită flexibilității sale și a bibliotecilor extinse, fiind utilizat pe scară largă în dezvoltarea web, automatizarea proceselor, analiza datelor, machine learning, inteligență artificială și multe alte domenii. Pentru dezvoltarea de jocuri și simulări în Unity, Python joacă un rol crucial, în special când este folosit împreună cu ML-Agents SDK pentru a implementa și antrena algoritmi de învățare automată. Python permite dezvoltatorilor să scrie scripturi complexe pentru antrenamentul agenților AI, gestionând fluxul de date între mediul Unity și modelele de machine learning. Comunitatea vastă a Python contribuie la o gamă largă de biblioteci precum NumPy, Pandas, Matplotlib, SciPy, TensorFlow și PyTorch, care sunt esențiale pentru procesarea datelor, calculul științific, și construirea rețelelor neuronale. Acest ecosistem bogat face din Python un instrument indispensabil în toolkit-ul oricărui dezvoltator care lucrează la intersecția dintre jocuri video și inteligență artificială, permițând o integrare fără probleme și extinderea capabilităților de AI în proiectele Unity.[12]

3.1.4 PyTorch

PyTorch este o bibliotecă de machine learning extrem de populară, care oferă flexibilitate și viteză în prototipare, datorită execuției imperative și a grafurilor de calcul dinamice. PyTorch este adesea preferat în comunitatea de cercetare academică și industrială pentru dezvoltarea rapidă a experimentelor cu AI, datorită simplității sale și suportului profund integrat pentru GPU-uri prin CUDA. Utilizarea PyTorch împreună cu CUDA

permite o accelerare semnificativă a antrenamentelor rețelelor neuronale, reducând timpul necesar pentru procesarea seturilor mari de date și iterarea rapidă a modelelor.[11]

3.1.5 Anaconda

Anaconda este o distribuție populară a limbajului Python, specializată în știința datelor, machine learning și dezvoltarea de aplicații științifice, fiind extrem de apreciată pentru gestionarea simplă și eficientă a pachetelor și a mediilor virtuale. Acesta vine cu Conda, un manager de pachete și medii care facilitează instalarea, rularea și actualizarea bibliotecilor și dependențelor necesare fără a intra în conflict cu alte pachete instalate pe sistem. Anaconda include o suită vastă de pachete preinstalate pentru analiza de date și machine learning, cum ar fi NumPy, Pandas, SciPy, Matplotlib, și Scikit-Learn, facilitând dezvoltarea rapidă și testarea prototipurilor de modele și algoritmi de învățare automată. În plus, Anaconda Navigator, o interfață grafică utilizator-friendly, permite accesul ușor la instrumente precum Jupyter Notebooks, Spyder IDE și RStudio, oferind un mediu integrat care sprijină cercetarea și dezvoltarea în domeniile științifice și tehnologice. Aceasta distribuție este deosebit de valoroasă în contextul Unity ML-Agents pentru crearea și gestionarea medii de dezvoltare Python, necesare pentru scripting și antrenarea algoritmilor de AI, asigurând compatibilitatea și eficiența pe parcursul întregului proces de dezvoltare.[10]

3.1.6 ML-Agents SDK

ML-Agents SDK este un toolkit avansat dezvoltat de Unity Technologies, destinat integrării tehnologiilor de inteligență artificială și învățare automată direct în motorul grafic Unity. Acesta permite dezvoltatorilor să creeze, să antreneze și să evalueze agenți inteligenți în medii simulate, folosind tehnici de învățare prin întărire (Reinforcement Learning), învățare supervizată, și alte metode de învățare automată. ML-Agents SDK utilizează Python pentru scriptarea algoritmilor de antrenament, oferind o interfață între mediile de joc Unity și algoritmi, ceea ce facilitează colectarea datelor de antrenament și aplicarea lor în procesul de învățare. O caracteristică notabilă a ML-Agents este abilitatea de a efectua antrenamente în paralel, permitând simularea și antrenarea simultană a multiple instanțe ale aceluiași agent, ceea ce accelerează semnificativ procesul de învățare. De asemenea, SDK-ul include funcții pentru vizualizarea și analiza performanței agenților, integrându-se cu instrumente precum TensorBoard pentru monitorizarea și evaluarea progresului agenților în timp real. Aceasta face ML-Agents SDK un instrument esențial pentru dezvoltatorii care doresc să exploreze și să implementeze soluții de AI avansate în aplicațiile și jocurile create cu Unity, transformând jocurile din simple programe în medii complexe de învățare și adaptare pentru AI.[9]

3.1.7 NVIDIA CUDA Toolkit

NVIDIA CUDA Toolkit este o platformă de calcul paralel dezvoltată de NVIDIA, care permite dezvoltatorilor să utilizeze unitățile de procesare grafică (GPU) pentru procesări matematice intense, optimizând performanța aplicațiilor care necesită un volum mare de calcule, precum simulările fizice, procesarea de imagini și, în mod particular, antrenamentul modelelor de învățare automată. CUDA Toolkit include compilatoare, biblioteci și API-uri care facilitează dezvoltarea software-ului care rulează pe arhitecturi multi-core, precum GPU-urile NVIDIA.[8]

3.1.8 Tensorboard

TensorBoard este un instrument de vizualizare dezvoltat de Google pentru TensorFlow, esențial pentru analiza și înțelegerea modelelor de învățare automată. Această platformă permite dezvoltatorilor să monitorizeze și să vizualizeze diverse aspecte ale procesului de antrenament al modelelor, incluzând metrice, grafice, histograme și proiecții de date de înaltă dimensiune.

În contextul Unity ML-Agents, TensorBoard este utilizat pentru a urmări performanța și comportamentul agenților pe parcursul și după antrenament. Oferă o vizualizare detaliată a evoluției și eficienței învățării, permițând dezvoltatorilor să analizeze complexitatea proceselor de învățare într-un mod intuitiv și accesibil. Această capacitate de vizualizare este extrem de valoroasă pentru optimizarea și rafinarea strategiilor de învățare prin întărire, contribuind semnificativ la dezvoltarea de agenți inteligenți mai performanți și adaptabili.[7]

3.2 Fundamente teoretice

În cadrul aplicatiei noastre, utilizăm algoritmi avansați de Reinforcement Learning pentru a optimiza performanța agenților în medii complexe. Acești algoritmi, Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), și Policy Optimization by Continuous Approximations (POCA), sunt deosebit de relevanți datorită abilităților lor de a gestiona eficient spații mari de acțiuni și stări.

3.2.1 SAC

Soft Actor-Critic (SAC) este un algoritm off-policy bazat pe actor-critic care optimizează o politică stocastică. Acest algoritm este proiectat pentru eficiență și stabilitate, utilizând o funcție obiectiv care maximizează suma recompensei așteptate și entropia politicilor. Această abordare favorizează explorarea prin încurajarea unor politici mai aleatorii, ceea ce îmbunătățește stabilitatea și accelerează convergența.

SAC utilizează trei ingrediente cheie: o arhitectură actor-critic cu rețele separate pentru politica și funcția de valoare, o formulare off-policy care permite reutilizarea datelor colectate anterior pentru eficiență, și maximizarea entropiei pentru a încuraja stabilitatea și explorarea.

Funcția obiectiv este definită astfel:

$$\pi^* = \arg\max_{\pi} \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t)) \right],$$

unde α este parametrul de temperatură care determină importanța relativă a termenului de entropie față de recompensă și, astfel, controlează stocasticitatea politicii optime.

Aceste caracteristici fac ca SAC să fie ideal pentru medii cu acțiuni continue, fiind adesea preferat pentru capacitatea sa de a naviga în spații complexe de acțiuni fără a sacrifica eficiența de calcul. În plus, SAC încorporează mai multe optimizări ale codului care modifică fundamental comportamentul agentului. Aceste optimizări includ normalizarea recompenselor și ajustarea ratei de învățare, care sunt esențiale pentru menținerea unei "regiuni de încredere" bazate pe divergența KL (Kullback-Leibler) între politicile succesive. Acest lucru asigură că pașii de actualizare rămân predictivi și eficienți, contribuind astfel la performanțele superioare ale algoritmului în diverse sarcini de învățare prin întărire.[1]

3.2.2 PPO

Proximal Policy Optimization (PPO) este unul dintre cei mai populari algoritmi de învățare prin întărire datorită echilibrului său eficient între simplitate și performanță. Acest algoritm de gradient de politică alternează între colectarea de date prin interacțiunea cu mediul și optimizarea funcției obiectiv prin ascensiunea gradientului stocastic.

Metoda tradițională de gradient de politică implică un estimator al gradientului de politică, definit astfel:

$$\hat{g} = \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

unde π_θ este o politică stochastică și $\hat{A}_t$ este un estimator al funcției de avantaj la momentul t. Funcția obiectivă corespunzătoare este:

$$L^{PG}(\theta) = \mathbb{E}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

Deși efectuarea multiplelor pași de optimizare pe această funcție obiectivă utilizând aceleași traiectorii poate fi atrăgătoare, acest lucru nu este bine justificat și adesea duce la actualizări de politică excesiv de mari și distructive.

PPO îmbunătățește politica prin minimizarea unei funcții obiectiv decupate, care previne actualizări prea mari ale politicii și ajută la menținerea stabilității învățării. Funcția

obiectiv decupată ("clipped") este definită astfel:[3]

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

unde $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ este raportul probabilităților, iar ϵ este un hiperparametru (adesea setat la 0.2).

Aceasta îl face ideal pentru scenarii unde recalcularea datelor este costisitoare, deoarece permite multiple epoci de actualizare pe aceleași date eșantionate, evitând astfel costurile asociate cu generarea frecventă de date noi. Studiile arată că optimizările la nivel de cod au un impact major asupra performanței agentului și asupra dinamicii de antrenament, subliniind dificultatea de a atribui câștigurile de performanță unui singur aspect al algoritmului.[2]

3.2.3 POCA

POCA este o arhitectură destinată mediiilor multi-agent, unde coordonarea și cooperarea între agenți sunt esențiale. Acest algoritm îmbunătățește deciziile individuale ale agenților prin luarea în considerare a impactului acțiunilor asupra performanței grupului. POCA folosește o structură critică care evaluează opțiuni multiple simultan, permițând optimizarea în concordanță cu o gamă largă de posibile politici și scenarii de interacțiune. Este particular util în medii unde acțiunile unui agent influențează direct rezultatele altor agenți, cum ar fi în jocurile de strategie de echipă sau scenarii de simulare socială. Folosind Unity ML-Agents, acești algoritmi sunt implementați și testați într-un cadru vizual și interactiv, permițând observarea directă a comportamentului și adaptării agenților. Integrarea în Unity facilitează simularea detaliată a interacțiunilor complexe și evaluarea performanței algoritmilor în condiții controlate, oferind o platformă valoroasă pentru rafinarea continuă a strategiilor de învățare și comportament.[4]

3.2.4 Curriculum Learning

Învățarea curriculum este o metodă de antrenament care structurează progresiv dificultatea sarcinilor pe care agentul trebuie să le învețe, urmărind să faciliteze învățarea continuă și eficientă. Inspirată din pedagogie, această abordare este deosebit de utilă când antrenăm agenți în medii complexe sau când vrem să optimizăm învățarea în medii cu sarcini de dificultăți variate. Ideea centrală este de a prezenta inițial exemple mai simple și de a crește treptat complexitatea acestora, pe măsură ce agentul devine mai competent.

Prezentarea organizată a exemplelor nu doar că facilitează convergența mai rapidă, dar poate și ajuta la găsirea unor minime locale de calitate superioară în cazul criteriilor de antrenament non-convexe. Astfel, învățarea curriculum poate fi văzută ca o metodă de continuare, unde se optimizează inițial o versiune netedă a obiectivului și apoi se trece

gradual la obiectivul original, mai complex. Acest proces ajută la ghidarea optimizării către regiuni favorabile ale spațiului de parametri, contribuind atât la viteza de convergență, cât și la generalizarea modelului.

Experimentele au arătat că strategiile de curriculum simplu în etape multiple pot îmbunătăți semnificativ generalizarea și pot accelera convergența. De exemplu, în sarcinile de viziune și limbaj, strategiile de curriculum au dus la obținerea unor rezultate mai bune comparativ cu abordările tradiționale de antrenament. Curriculum learning acționează, de asemenea, ca un regularizator, având un efect benefic mai pronunțat asupra setului de testare, reducând eroarea de generalizare.

În contextul antrenamentului agenților inteligenți, învățarea curriculum poate fi utilizată pentru a gestiona complexitatea sarcinilor și pentru a facilita dezvoltarea progresivă a abilităților. Prin adaptarea treptată a dificultății sarcinilor, agenții sunt capabili să învețe mai eficient și să se adapteze mai bine la noi provocări, ceea ce duce la performanțe superioare în medii variate și complexe.[5]

3.2.5 Sistem rating ELO

Recompensa cumulativă a mediului în jocurile adversariale nu este întotdeauna un indicator adecvat pentru a urmări progresul învățării. Acest lucru se datorează faptului că recompensa totală depinde în totalitate de cât de bine se descurcă adversarul. Dacă un agent cu un anumit grad de abilități se confruntă cu un adversar mai slab sau mai puternic, respectiv, acesta va primi mai multe sau mai puține recompense. Prin urmare, este mai eficient să utilizăm sistemul de rating ELO. Fiecare jucător începe cu un scor ELO inițial, definit în parametrul de configurare initial elo al antrenorului. Diferența dintre ratingurile celor doi jucători servește ca predictor pentru rezultatele unui meci. De exemplu, dacă jucătorul A are un scor ELO de 2000 și jucătorul B un scor ELO de 1500, probabilitatea ca jucătorul A să câștige este de aproximativ 94.6%, în timp ce probabilitatea ca jucătorul B să câștige este de aproximativ 5.4%, valoarea acestui rating ar trebui să crească constant.[6]

$$E = \frac{1}{1 + 10^{\frac{\text{Rating adversar} - \text{Rating propriu}}{400}}} \tag{3.1}$$

La sfârșitul jocului, în funcție de rezultat, actualizăm scorul ELO real al jucătorilor, folosind o ajustare liniară proporțională cu măsura în care jucătorul a performant peste sau sub așteptări. Jucătorul care câștigă ia puncte de la cel care pierde:

- Dacă jucătorul cu rating mai mare câștigă → se vor lua puține puncte de la jucătorul cu rating mai mic.
- Dacă jucătorul cu rating mai mic câștigă → se vor lua multe puncte de la jucătorul cu rating mai mare.

• Dacă este remiză → jucătorul cu rating mai mic câștigă câteva puncte de la cel cu rating mai mare. Actualizăm ratingul jucătorilor folosind formula:

$$\text{Rating nou} = \text{Rating actual} + K \times (\text{Scor real} - \text{Scor așteptat}) \tag{3.2}$$

unde K este un factor de scalare care ajustează sensibilitatea schimbării ratingului.

La începutul antrenamentului, toți agenții au aceleași abilități. Așadar, scorul ELO pentru fiecare dintre ei este definit prin parametrul $initial\_elo = 1200.0$. Calculăm scorul așteptat E:

$$E_a = 0.5, \quad E_b = 0.5$$

Asta înseamnă că fiecare jucător are șanse de 50% să câștige punctul.

Dacă A câștigă, noul rating R va fi:

$$R_a = 1200 + 16 \times (1 - 0.5) \rightarrow 1208$$

$$R_b = 1200 + 16 \times (0 - 0.5) \rightarrow 1192$$

Jucătorul A are acum un scor ELO de 1208, iar jucătorul B un scor ELO de 1192. Prin urmare, jucătorul A este acum puțin mai bun decât jucătorul B.

Capitolul 4

Implementare

4.1 Arhitectura mediului de învățare

În următoarea parte a proiectului, voi parcurge pas cu pas arhitectura mediului utilizat, explicând modul de configurare al scenei și agenților, precum și modul în care aceste componente interacționează pentru a obține rezultatele dorite.

SoccerTwos, unul din numeroasele medii puse la dispoziție de Unity ML-Agents, reprezintă o simulare dinamică și interactivă utilizată pentru antrenarea și evaluarea agenților de ”reinforcement learning” într-un cadru competitiv și cooperativ. Acest mediu este configurat în cadrul scenei Unity de mai jos și include mai multe componente esențiale. Terenul central este reprezentat de un spațiu de joc care servește ca arenă pentru meciurile de fotbal. La fiecare capăt al terenului se află porțile, care sunt țintele în care echipele încearcă să înscrie. Mai departe, vom analiza obiectele scenei principale.[27]

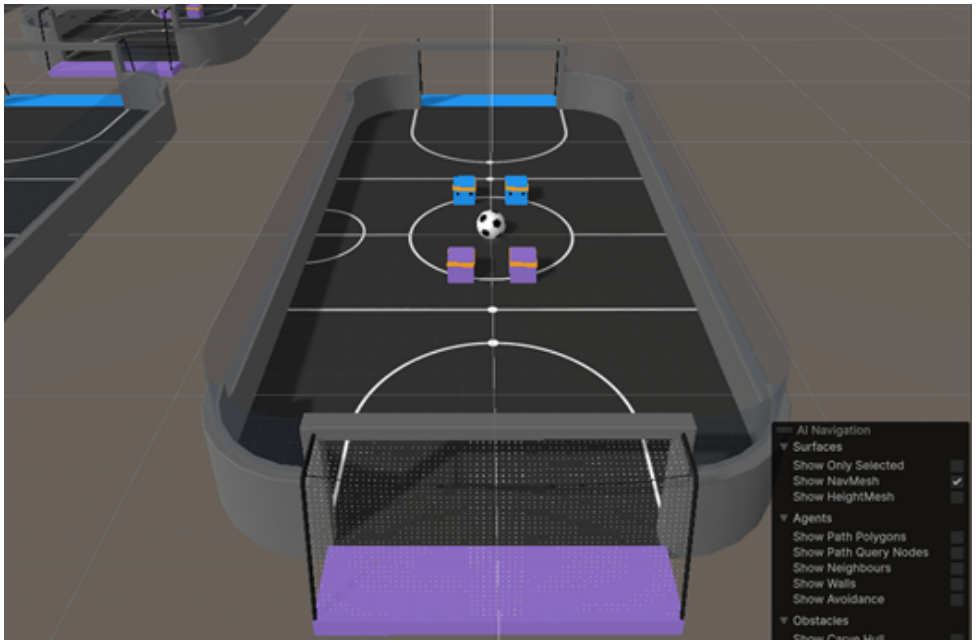

Figura 4.1: Imagine de ansamblu a mediului

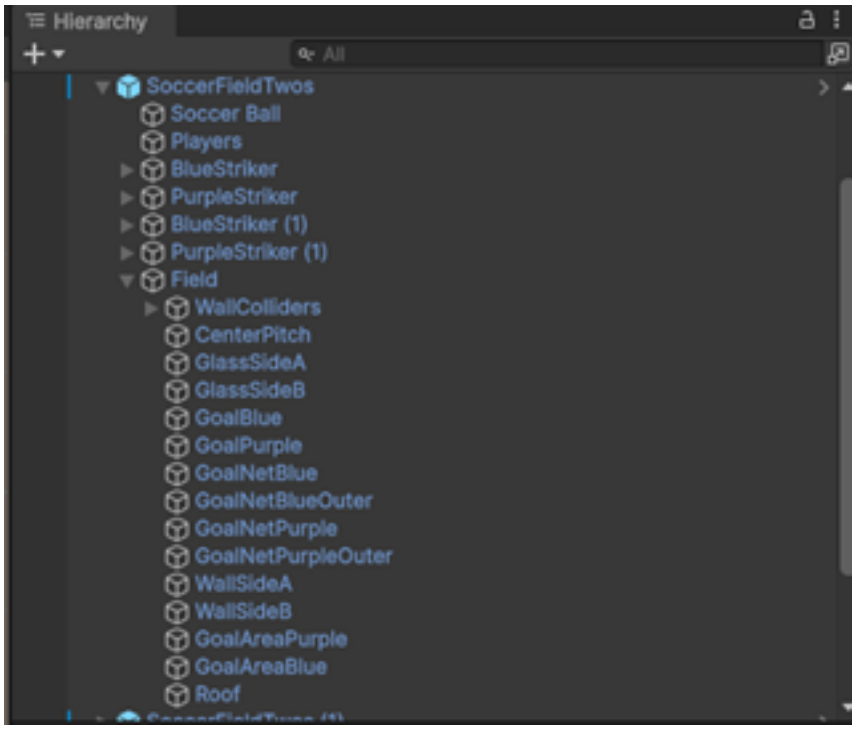

Figura 4.2: Ierarhia obiectelor scenei principale

Suprafața terenului este plană și delimitată de linii clare. În colțurile terenului, sunt amplasate camere de supraveghere virtuale care capturează fiecare mișcare a agenților, oferind date esențiale pentru analiză și optimizare. De asemenea, lumina ambientală și umbrele sunt ajustate pentru a crea un mediu vizual plăcut și realist.

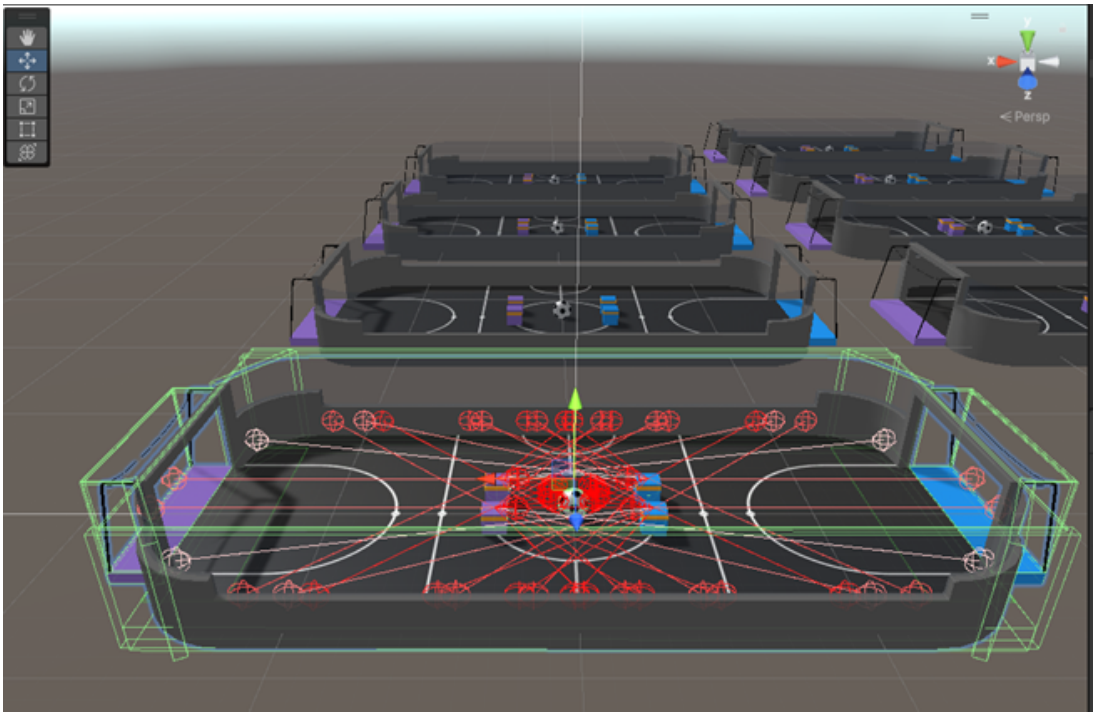

Figura 4.3: Imagine detaliată a mediului

Mai jos este prezentată ierarhia detaliată a scenei principale, reprezentând componentele care o alcătuiesc, fiecare parte având un rol important în funcționarea întregului sistem:

Componentă	**Descriere**
SoccerFieldTwos	Obiectul principal care reprezintă întreaga scenă de joc SoccerTwos. Toate celelalte obiecte sunt subcomponente ale acestuia.
Soccer Ball	Mingea de fotbal utilizată în joc. Este principalul obiect de interacțiune, pe care agenții îl controlează pentru a înscrie goluri.
Players	Conține toți jucătorii prezenți în acțiune.
Field	Reprezintă terenul de joc și include toate componentele necesare pentru definirea și limitarea spațiului de joc: • WallColliders: Collidere pentru pereții terenului. • CenterPitch: Centrul terenului. • GlassSideA și GlassSideB: Laturile terenului. • GoalBlue și GoalPurple: Porțile echipelor. • GoalNetBlue și GoalNetPurple: Plasele porților. • GoalNetBlueOuter și GoalNetPurpleOuter: Collidere externe pentru porți. • WallSideA și WallSideB: Pereții laterali ai terenului. • GoalAreaBlue și GoalAreaPurple: Zonele din fața porților. • Roof: Acoperișul terenului.

Scriptul SoccerEnvController → gestionează mediul de antrenament, pregătind totul pentru o antrenare eficientă. Acesta este responsabil pentru inițializarea agenților, resetarea scenei și actualizarea episoadelor de antrenament.

Funcționalitate	Descriere
Inițializare (Start)	• Inițializează mediul și agenții. • Setează pozițiile și rotațiile inițiale ale agenților și mingii. • Înregistrează agenții în grupurile corespunzătoare (echipa albastră și echipa mov).
Actualizare (FixedUpdate)	• Monitorizează și incrementează un timer la fiecare pas. • Resetează scena dacă numărul maxim de pași este atins, și marchează episodul ca întrerupt pentru agenții din ambele echipe.
Resetarea Mingii (ResetBall)	• Plasează mingea în zona de start și resetează viteza și rotația acesteia.
Gestionarea Golurilor (GoalTouched)	• Atribuie recompense echipei care înscrie și penalizează echipa opusă. • Încheie episodul pentru ambele echipe și resetează scena.
Resetarea Scenei (ResetScene)	• Resetează timer-ul. • Reinitializează pozițiile și rotațiile agenților și mingii pentru a pregăti un nou episod.

4.2 Arhitectura agentului

Spațiul de Observație → Spațiul de observație al agenților constă în 336 de dimensiuni, determinate de utilizarea unor ray-cast-uri pentru a detecta mediul înconjurător. Aceste dimensiuni sunt împărțite astfel:[28]

- **Ray-cast-uri frontale (Frontal):**
  - 11 ray-cast-uri distribuite pe un unghi de 120 de grade în fața agentului.
  - Fiecare ray-cast detectează 6 tipuri posibile de obiecte și distanța până la obiecte.
  - Total: 11 ray-cast-uri × 6 tipuri de obiecte × 4 (distanța inclusă) = 264 dimensiuni de stare.
- **Ray-cast-uri spre spate (Backward):**
  - 3 ray-cast-uri distribuite pe un unghi de 90 de grade în spatele agentului.
  - Fiecare ray-cast detectează 6 tipuri posibile de obiecte și distanța până la obiecte.
  - Total: 3 ray-cast-uri × 6 tipuri de obiecte × 4 (distanța inclusă) = 72 dimensiuni de stare.

Aceste ray-cast-uri permit agentului să colecteze informații precise, identificând tipurile de obiecte și distanțele până la acestea.

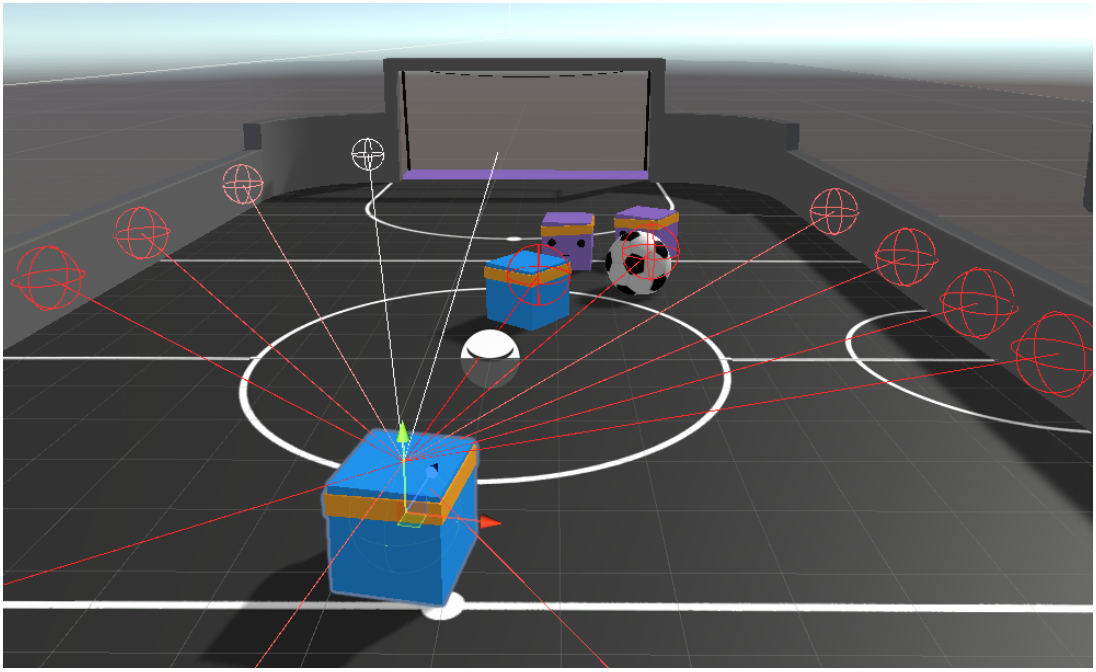

Figura 4.4: Raycast față

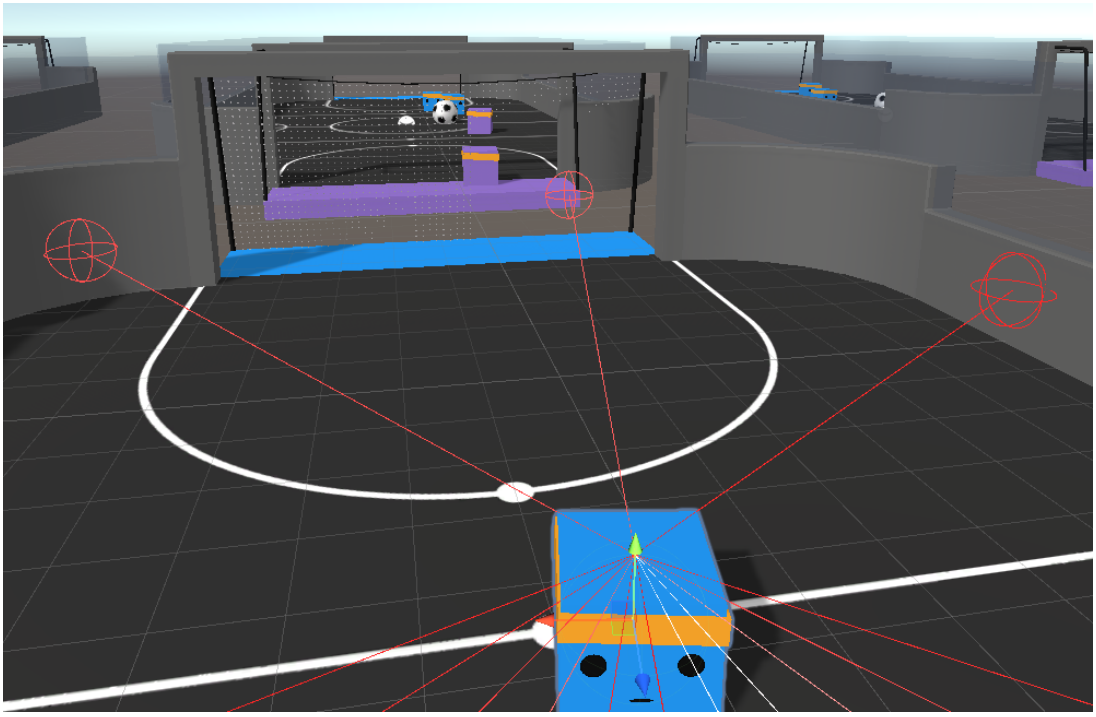

Figura 4.5: Raycast spate

Spațiul de Acțiune → Spațiul de acțiune constă în 3 acțiuni discrete ramificate (*MultiDiscrete*), care controlează mișcarea agentului și rotația acestuia.[27] Acțiunile sunt:

1. **Mișcare înainte și înapoi:**
   - Discretizată în mai multe nivele (ex. 0: fără mișcare, 1: mișcare înainte, 2: mișcare înapoi).
2. **Mișcare laterală:**
   - Discretizată în mai multe nivele (ex. 0: fără mișcare, 1: mișcare spre stânga, 2: mișcare spre dreapta).
3. **Rotație:**
   - Discretizată în mai multe nivele, pentru a permite rotația agentului în planul orizontal (ex. 0: fără rotație, 1: rotație stânga, 2: rotație dreapta).

În total, spațiul de acțiune cuprinde 27 de acțiuni discrete posibile (3 acțiuni ramificate cu 3 nivele fiecare).

Scriptul AgentSoccer → definește comportamentul agenților într-un joc de fotbal simulativ. Este responsabil pentru inițializarea agenților, mișcarea acestora în funcție de acțiuni discrete, gestionarea coliziunilor și recompensarea comportamentului adecvat.

1. Inițializare (Initialize)

- Configurează variabilele agentului, setările de echipă și pozițiile inițiale.
- Inițializează componenta Rigidbody și alte setări relevante.
- Determinează recompensa existențială pe baza duratei maxime a episodului.

2. Mișcarea Agentului (MoveAgent)

- Primește acțiuni discrete și traduce aceste acțiuni în mișcare înainte/înapoi, mișcare laterală și rotație.
- Folosește AddForce pentru mișcare și transform.Rotate pentru rotație.

3. Primirea Acțiunilor (OnActionReceived)

- Primește acțiunile calculate de modelul de învățare și apelează MoveAgent pentru a executa acțiunile.
- Atribuie recompense sau penalizări agenților.

4. Control Manual (Heuristic) → Această funcție permite testarea manuală a agentului în mediul Unity, utilizând inputuri de la tastatură pentru a genera acțiuni discrete. Acest lucru este utilizat pentru a verifica comportamentul agentului fără a rula algoritmi de învățare automată. Funcția acoperă mișcările de bază: înainte, înapoi, rotație și mișcare laterală, oferind un control complet asupra agentului.

Tabela 4.3: Schema Rezumată a Acțiunilor

Tastă	**Acțiune**	**Index în `discreteActionsOut`**	**Valoare**
W	Mișcare înainte	0	1
S	Mișcare înapoi	0	2
A	Rotație stânga	2	1
D	Rotație dreapta	2	2
E	Mișcare dreapta	1	1
Q	Mișcare stânga	1	2

5. Gestionarea Coliziunilor (OnCollisionEnter):)

- Detectează coliziunile cu mingea și aplică o forță de lovire pentru a șuta mingea.
- Oferă o recompensă pentru atingerea mingii.

4.3 Funcția de recompensă în antrenarea agenților

Funcția de recompensă este un element central în ”reinforcement learning”, deoarece aceasta ghidează comportamentul agenților către atingerea obiectivelor dorite. În contextul actual, funcția de recompensă este esențială pentru a antrena agenții să joace fotbal eficient și să colaboreze sau să concureze cu alți agenți pentru a înscrie goluri și a preveni adversarii să facă același lucru.[29]

Structura funcției de recompensă → Funcția de recompensă este implementată pentru a încuraja comportamentele pozitive și pentru a descuraja comportamentele negative. Aceasta poate fi detaliată în mai multe componente:

Recompense pentru Obiective Specifice

- **Înscrierea unui gol**: Agenții primesc o recompensă pozitivă substanțială atunci când echipa lor înscrie un gol. Aceasta este o recompensă extrinsecă care motivează agenții să colaboreze pentru a atinge scopul principal al jocului.
- **Penalizări pentru autogoluri**: Dacă un agent înscrie în propria poartă, primește o penalizare severă. Acest lucru descurajează comportamentele care duc la autogoluri.

Recompense Existențiale

- **Bonusuri pentru portari**: Agenții pot primi o recompensă existențială pentru fiecare pas de simulare, ceea ce le formează un caracter prudent, îi motivează să își păstreze poziția și să prevină golurile.
- **Penalizări pentru atacanți**: Agenții pot primi o penalizare existențială pentru un caracter prea defensiv, fiind motivați să fie ofensivi și să încerce să înscrie goluri.

Recompense pentru Interacțiuni

- **Atingerea mingii**: Agenții primesc recompense pentru atingerea mingii, ceea ce îi motivează să fie activi și să participe la joc. Această recompensă poate varia în funcție de importanța atingerii mingii în contextul jocului.

Listing 4.1: GoalTouched method in C#

```
public void GoalTouched(Team scoredTeam)
{
    if (scoredTeam == Team.Blue)
    {
        m_BlueAgentGroup.AddGroupReward(1 - (float)m_ResetTimer /
            MaxEnvironmentSteps);
        m_PurpleAgentGroup.AddGroupReward(-1);
    }
    else
    {
        m_PurpleAgentGroup.AddGroupReward(1 - (float)m_ResetTimer /
            MaxEnvironmentSteps);
        m_BlueAgentGroup.AddGroupReward(-1);
    }
    m_PurpleAgentGroup.EndGroupEpisode();
    m_BlueAgentGroup.EndGroupEpisode();
    ResetScene();
}
```

În acest exemplu, echipa care înscrie un gol primește o recompensă pozitivă proporțională cu timpul rămas din episod, în timp ce echipa adversă primește o penalizare. Aceasta încurajează echipele să înscrie cât mai repede posibil.[21]

Impactul funcției de recompensa → Funcția de recompens joacă un rol crucial în direcționarea învățării agenților. Prin atribuirea de recompense și penalizări, agenții învață să îmbunătățească comportamentele care duc la succes și să evite comportamentele care nu sunt productive. Aceasta include:

- **Dezvoltarea strategiilor ofensive**: Agenții sunt încurajați să coopereze pentru a înscrie goluri, învățând astfel să dezvolte strategii ofensive eficiente.
- **Îmbunătățirea abilităților defensive**: Prin recompense existențiale și penalizări, agenții învață să își păstreze pozițiile și să prevină înscrierea golurilor de către adversari.
- **Participarea activă la joc**: Recompensele pentru atingerea mingii și alte interacțiuni stimulează agenții să fie activi și implicați în joc.

4.4 Antrenare si testare

În diagrama de mai jos este prezentat tot procesul prin care scena Unity poate fi pusă în desfășurare, într-un mod simplu și concis.

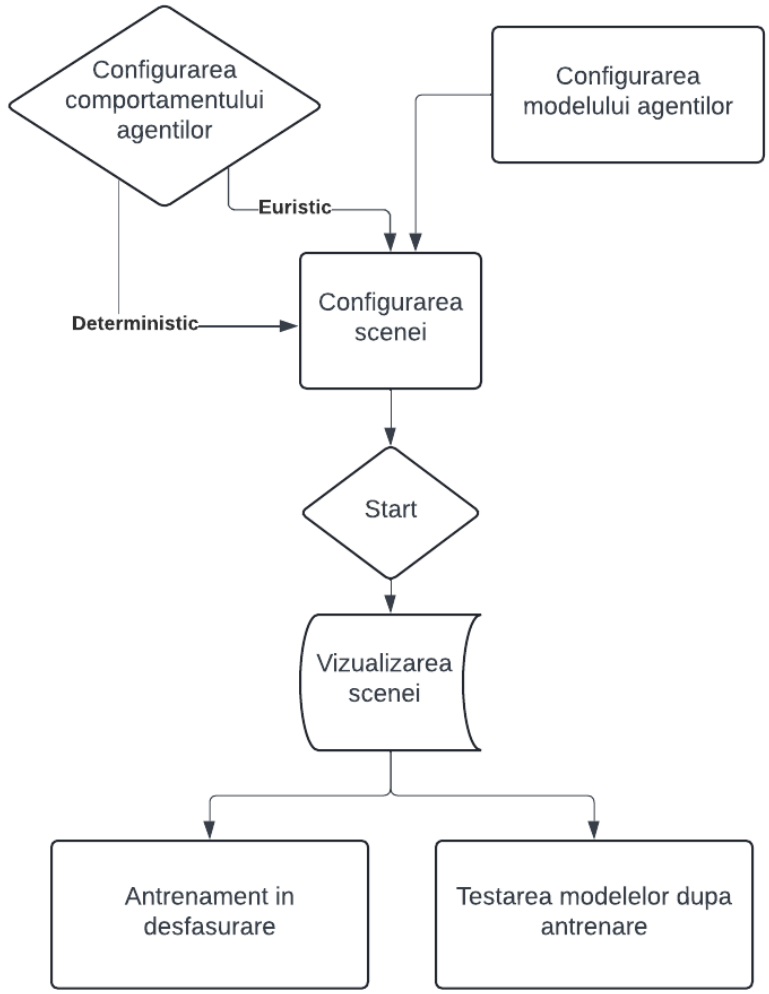

Figura 4.6: Proces de executare al scenei

Pentru a permite antrenarea în paralel, o singură scenă va stoca mai multe terenuri, duplicate, care vor antrena in paralel modelul propus. Avantajele acestei configurări sunt: [31]

- **Eficiență sporită**: Antrenarea simultană în opt instanțe permite colectarea rapidă a unei cantități mari de date, accelerând procesul de învățare.
- **Diversificarea datelor**: Agenții sunt expuși la o varietate mai mare de scenarii, ceea ce duce la un model mai robust și capabil să generalizeze mai bine.
- **Optimizarea resurselor**: Unity ML-Agents utilizează eficient resursele CPU și

GPU prin rularea în paralel a mai multor instanțe, reducând timpul total de antrenament.

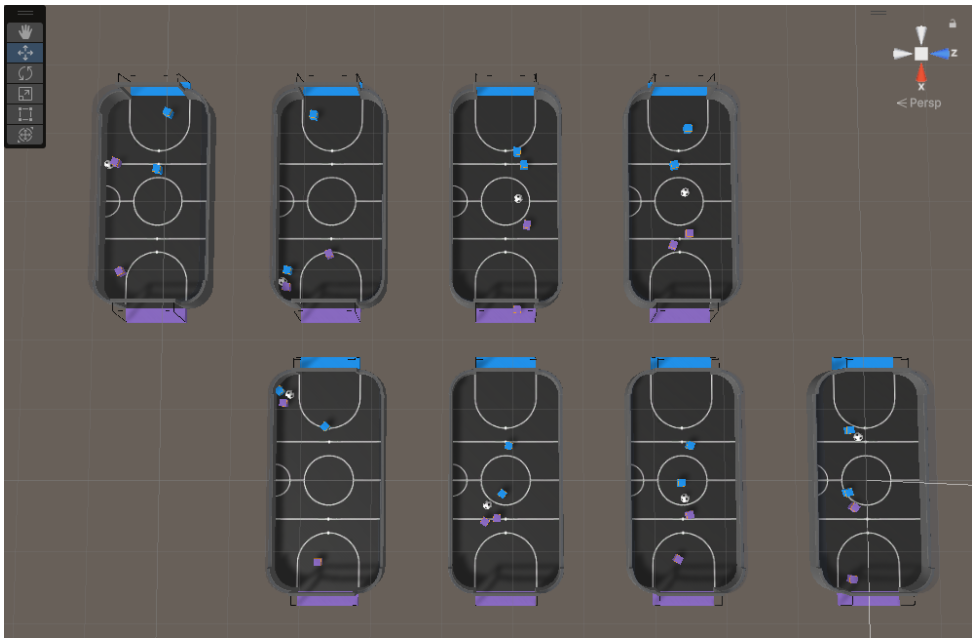

Figura 4.7: Sistem de terenuri duplicate

Configurarea Curriculum Learning pentru scena de fotbal → În continuare este prezentat un exemplu de configurare YAML pentru curriculum learning în Unity ML-Agents, specific pentru o scenă de fotbal:

Listing 4.2: Configurarea YAML pentru Curriculum Learning

```
Agent_curriculum:
    class_name: curriculum
    parameters:
      curriculums:
        lesson_length: 100000
        signal_smoothing: true
        thresholds: [0.2, 0.4, 0.6, 0.8]
        measure: reward
      lessons:
        - name: "Lectie1"
          value: 0
          config:
            ball_speed: 1.0
            opponent_difficulty: 0.2
            goal_reward: 1.0
            self_goal_penalty: -1.0
        - name: "Lectie2"
          ...
```

Această configurare definește un program de învățare curriculum pentru agentul de fotbal, progresând de la sarcini simple la sarcini mai complexe pe măsură ce agentul învață. Fiecare lecție modifică parametrii de dificultate și recompense pentru a ghida antrenamentul agentului. [23]

Senzor de gol → Agenții sunt capabili să strângă date care sunt utilizate ca semnale de obiectiv. Politica agentului depinde de un semnal de obiectiv, astfel încât dacă obiectivul se schimbă, se va modifica și politica agentului, adică modul în care acesta transformă observațiile în acțiuni. Este important să menționăm că orice observație influențează politica agentului, însă putem accentua importanța acestei condiționări. În Unity, acest lucru se face adăugând un VectorSensorComponent sau un CameraSensorComponent agentului și alegând semnalul de obiectiv ca tip de observație.[30]

Listing 4.3: Modified CollectObservations Method

```
public override void CollectObservations(VectorSensor sensor)
    {
        System.Array roles = System.Enum.GetValues(typeof(Role));
        if (cameraSensor != null)
        {
            int goalIndex = (int)CurrentGoal;
            sensor.AddOneHotObservation(goalIndex, roles.Length);
            cameraSensor.Sensor.Update();
        }
    }
```

4.5 Configurarea fișierelor de antrenare

În cadrul procesului de antrenament al agenților, fișierul de configurare joacă un rol esențial în definirea parametrilor și strategiilor utilizate pentru optimizarea învățării automate. Acest fișier, scris în format YAML, permite personalizarea diverselor aspecte ale antrenamentului, de la setările de bază ale mediului și ale agenților, până la hiperparametrii specifici algoritmului de învățare.[32]

1. Tipul Trainerului

Parametru	**Descriere**
trainer_type	Specifică tipul de algoritm de antrenare utilizat.

2.Hiperparametri

Parametru	**Descriere**
batch_size	Numărul de experiențe utilizate pentru fiecare actualizare a modelului. O valoare mare ajută la stabilizarea estimărilor de gradient, dar necesită mai multă memorie.
buffer_size	Numărul total de experiențe stocate înainte de actualizarea modelului. Un buffer mare permite acumularea de mai multe date înainte de antrenare, ceea ce poate duce la o politică mai stabilă.
learning_rate	Rata de învățare a modelului. O rată de învățare mică permite convergența treptată, reducând riscul de supraînvățare.
beta	Coeficientul de regularizare L2. Acest parametru ajută la prevenirea supraînvățării prin penalizarea valorilor mari ale greutăților.
epsilon	Parametrul de cliping PPO, care controlează cât de mult poate varia politica în timpul actualizărilor.
lambd	Parametrul de regularizare pentru GAE (Generalized Advantage Estimation), care echilibrează bias-ul și varianța în estimarea avantajului.
num_epoch	Numărul de epoci în care se actualizează modelul pentru fiecare subset de date.
learning_rate_schedule	Setat pe constant, înseamnă că rata de învățare rămâne fixă pe toată durata antrenării.

3.Setarile retelei neurale

Parametru	Descriere
normalize	Setat pe false, înseamnă că datele de intrare nu sunt normalizate înainte de a fi introduse în rețea.
hidden_units	Numărul de unități ascunse pe fiecare strat al rețelei. Un număr mare de unități permite modelului să capteze relații complexe în date.
num_layers	Numărul de straturi ascunse ale rețelei, care oferă un echilibru între complexitate și capacitatea de generalizare.
vis_encode_type	Setat pe simple, indică utilizarea unui encoder vizual simplu pentru datele de intrare vizuale.

4.Setări mediu

Parametru	Descriere
keep_checkpoints	Numărul de puncte de control de salvat. Păstrarea mai multor puncte de control permite revenirea la stări anterioare ale modelului dacă antrenamentul devine instabil.
max_steps	Numărul maxim de pași de antrenament, indicând o durată de antrenament pentru a permite modelului să învețe dintr-un volum mare de date.
time_horizon	Cuanta de timp pentru colectarea experiențelor, permite agenților să acumuleze experiențe pe o perioadă mai lungă înainte de a le trimite în buffer.
summary_freq	Frecvența la care sunt înregistrate sumarizările de antrenament.

5.Auto-joc

Parametru	Descriere
save_steps	Frecvența la care se salvează stările modelului în timpul autojoculu.
team_change	Frecvența la care agenții își schimbă echipele, pentru a asigura o diversitate a experiențelor.
swap_steps	Numărul de pași între schimbările de roluri ale agenților.
windows	Dimensiunea ferestrei pentru evaluarea performanței.
play_against_latest...	Proporția de jocuri împotriva celui mai recent model, pentru a evalua progresul față de cele mai recente versiuni.

Parametru	Descriere
initial_elo	Scorul ELO inițial pentru agenți, folosit pentru a evalua performanța relativă a agenților pe măsură ce progresează în antrenament.

Capitolul 5

Evaluarea rezultatelor

Pentru a putea înțelege cum stabilim anumite standarde de performanță ale modelelor antrenate, trebuie să înțelegem metricile și graficele utilizate în procesul de antrenament. Aceste grafice oferă informații esențiale despre progresul și eficiența agentului, permițându-ne să evaluăm și să ajustăm strategiile de învățare pentru a obține rezultate optime.[33]

1. Graficul Policy/Entropy

- **Descriere**: Entropia în contextul învățării prin întărire indică caracterul aleatoriu al deciziilor agentului. Aceasta valoare, pe parcursul unei sesiuni de antrenament reușite, ar trebui să scadă.
- **Interpretare**: Valorile mai mari sugerează că agentul explorează mai multe acțiuni, în timp ce valorile mai mici indică o politică deterministă. Scăderea entropiei este un semn că agentul învață să favorizeze anumite acțiuni care conduc la recompense mai mari. Cu toate acestea, o scădere prea rapidă a entropiei poate sugera că agentul se fixează pe o strategie suboptimă și nu explorează suficient

2. Graficul Losses/Policy Loss

- **Descriere**: Pierderea politicii măsoară cât de bine se potrivește politica actuală a agentului cu politica optimă. Acest lucru corelează cu cât de mult se schimbă politica. Valoarea ar trebui să scadă în timpul unei sesiuni de antrenament reușite.
- **Interpretare**: Valorile mai mici indică o potrivire mai bună. Creșterea pierderii politicii poate indica faptul că agentul întâmpină dificultăți în optimizarea politicii sale pe măsură ce mediul devine mai complex. Acest lucru poate fi cauzat de factori precum explorarea insuficientă sau hiperparametri suboptimali

3. Graficul Self-play/ELO

- **Descriere**: Ratingul ELO masoară performanța relativă a agenților în meciurile de auto-joc.
- **Interpretare**: Valori mai mari indică o performanță mai bună. Creșterea inițială sugerează că agentul învață rapid strategii eficiente. Fluctuațiile și scăderea ulterioară a valorii ELO indică instabilități în performanță, probabil din cauza supraînvățării sau a dificultății în adaptarea la noi strategii ale adversarilor

4. Graficul Episode Length

- **Descriere**: Acest indicator măsoară durata fiecărui episod de antrenament.
- **Interpretare**: O lungime stabilă a episodului indică faptul că agentul a învățat să mențină o performanță consistentă în cadrul mediului. Variațiile mari ale lungimii episodului pot sugera instabilități în strategie sau dificultăți în adaptarea la mediu

5. Graficul Group Cumulative Reward

- **Descriere**: Recompensa cumulativă măsoară suma recompenselor primite de agenți pe parcursul episoadelor de antrenament. Recompensele pot fi pozitive sau negative
- **Interpretare**: Fluctuațiile inițiale mari indică faptul că agenții testează diverse strategii. Stabilizarea recompenselor cumulative pe parcurs sugerează că agenții au început să adopte strategii mai eficiente. Simetria recompenselor indică încercări de ajustare a strategiilor pentru a maximiza recompensele și a minimiza penalizările

În graficul următor, vom prefața performanțele modelelor prezentate în această secțiune a lucrării.

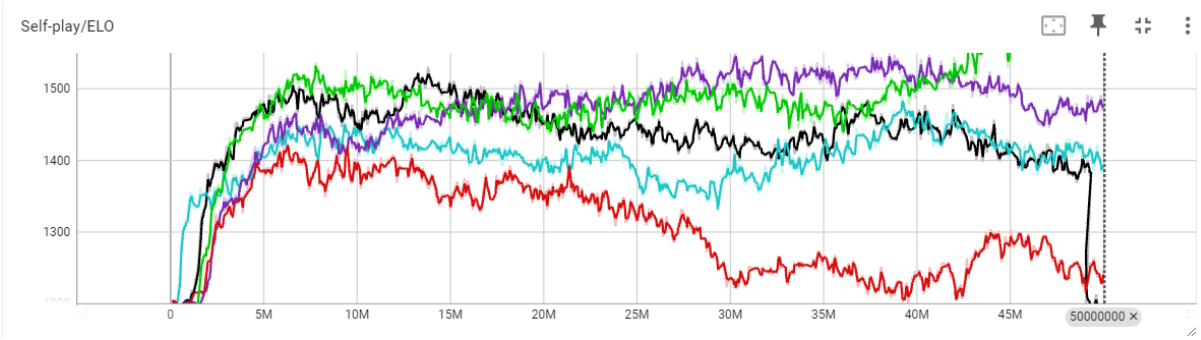

Figura 5.1: Scorurile ELO ale modelelor propuse

Analiza performanțelor modelului de bază - utilizand algoritmul POCA
Performanța inițială a modelului de bază este promițătoare, reflectată prin creșterea rapidă a ratingului ELO și scăderea entropiei, indicând că agentul învață rapid strategii eficiente și devine mai sigur pe acțiunile sale. Modelul atinge un vârf al scorului ELO de aproximativ 1500 inainte de 10 milioane de pași. Cu toate acestea, pe termen lung, modelul întâmpină provocări semnificative. Ratingul ELO fluctuează și scade spre sfârșitul antrenamentului, ajungând la o valoare finală sub 1400 la 50 de milioane de pași. Aceasta sugerează dificultăți în menținerea performanței optime, posibil din cauza supraînvățării sau a incapacității de a se adapta la noi strategii ale adversarilor. Pierderea politicii crește treptat, indicând o discrepanță tot mai mare între politica actuală și cea optimă, ceea ce poate fi cauzat de explorarea insuficientă sau de hiperparametri suboptimali. Graficul lungimii episodului arată o stabilizare, dar cu variații semnificative, sugerând inconsistențe în performanță. În concluzie, deși modelul de bază arată o performanță bună inițial, ajustările ulterioare ale hiperparametrilor și strategii mai eficiente de explorare sunt necesare pentru a îmbunătăți stabilitatea și eficiența pe termen lung.

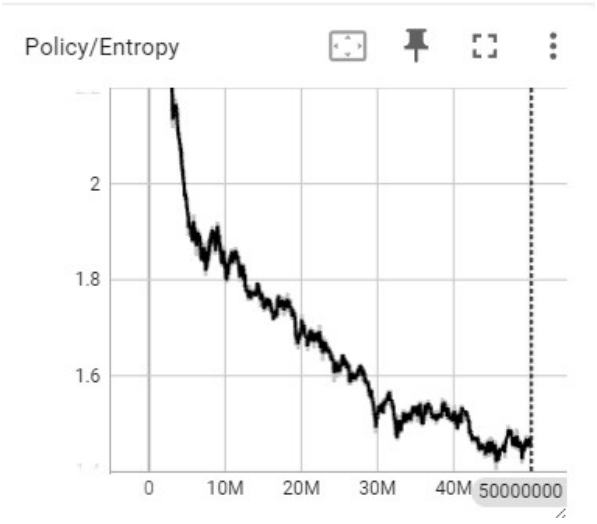

Figura 5.2: Entropie

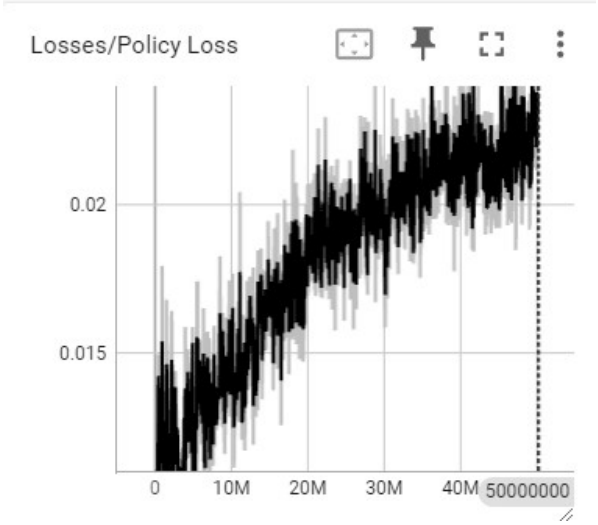

Figura 5.3: Pierdere a politicii

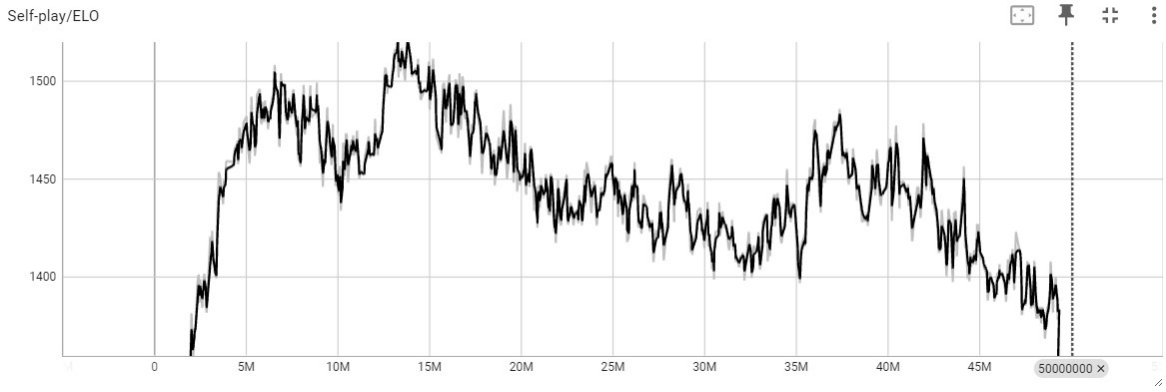

Figura 5.4: Scor ELO

Analiza Performanțelor Algoritmului SAC - Un experiment eșuat

Performanța acestul model, utilizând o configurație diferită, la început este promițătoare, cu o creștere inițială a ratingului ELO, dar care scade rapid după aproximativ 10 milioane de pași, fără o recuperare semnificativă până la sfârșitul antrenamentului. Entropia a început ridicată și a scăzut ușor pe parcursul antrenamentului, sugerând o explorare excesivă și o lipsă de convergență către o politică optimă. Pierderea politicii a crescut constant și a fluctuat semnificativ, indicând dificultăți în stabilizarea politicii pe parcursul antrenamentului. Lungimea episoadelor a variat semnificativ, fără o tendință clară de stabilizare, sugerând inconsistențe în performanță. Recompensa cumulativă a arătat fluctuații mari și o distribuție simetrică, indicând atât episoade de performanță bună, cât și de performanță slabă.

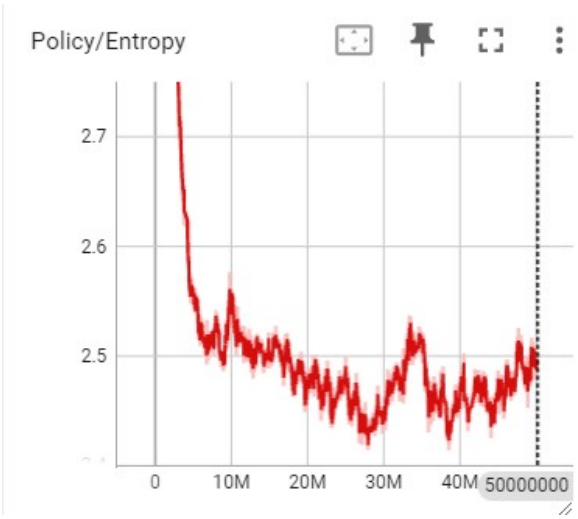

Figura 5.5: Grafic entropie

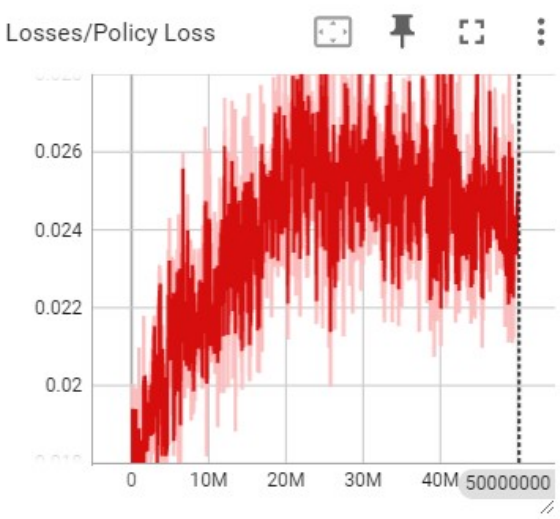

Figura 5.6: Grafic de pierdere a politicii

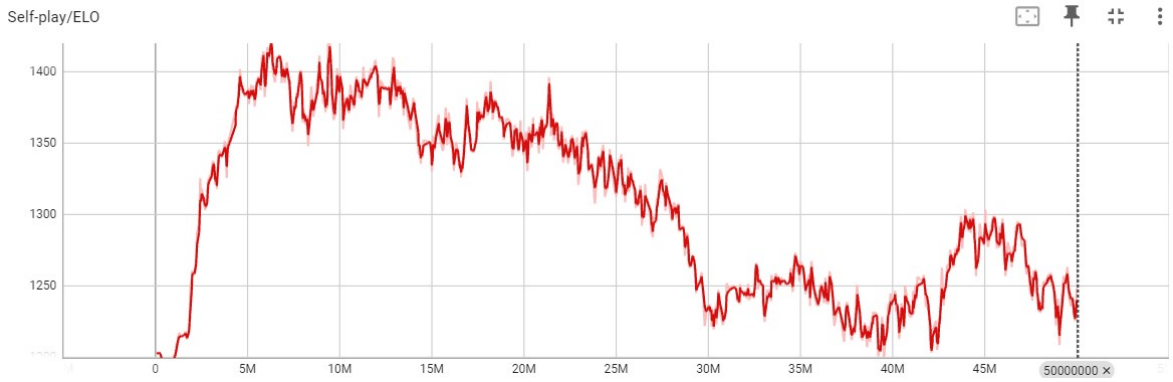

Figura 5.7: Grafic scor ELO

Experimentul a eșuat în principal din cauza instabilităților modelului în antrenament. Fluctuațiile mari în pierderi și recompense, împreună cu dificultatea de a menține o politică eficientă, sugerează că agentul nu a reușit să optimizeze politica într-un mod stabil

și consistent. Spre deosebire de POCA, care este un algoritm multi-agent și mai potrivit pentru medii complexe cu mai mulți agenți, algoritmul SAC nu a reușit să se adapteze eficient. Ajustarea hiperparametrilor și explorarea unor strategii de antrenament alternative ar putea ajuta la îmbunătățirea performanței în viitoarele experimente cu SAC.

Analiza performanțelor algoritmului PPO - diferite configurații

În această etapă, vom analiza performanțele algoritmului PPO, utilizând diverse metode pentru a ilustra eficiența și stabilitatea acestuia în diferite scenarii. Vom discuta despre cele mai bune configurații care reprezintă categoria din care fac parte. Vom defini astfel modelul A care reprezintă o variantă PPO de bază, și modelul B în care am încercat să includ atât concepte definite anterior, precum învățare curriculum și alte metode care să ajute agenții să învețe mai eficient și să se adapteze mai bine la mediu.

Grafic	Observatii si Interpretari
Losses/Baseline Loss	Modelul A are pierderi mai mici comparativ cu modelul B. Acest lucru indică o stabilitate inițială mai bună și sugerează că modelul A are o performanță mai consistentă în predicțiile inițiale, dar poate fi o indicație că nu explorează suficient.
Losses/Policy Loss	Pierderea politicii crește spre final pentru modelul A, în timp ce modelul B menține fluctuații mari, dar mai stabile. Creșterea pierderii politicii pentru modelul A indică dificultăți în menținerea stabilității pe termen lung, în timp ce modelul B ajustează continuu politica pentru a găsi soluții optime, reflectând o explorare continuă.
Losses/Value Loss	Modelul B are fluctuații mai mari în pierderile de valoare, sugerând dificultăți inițiale în estimarea corectă a valorilor stărilor. Acest comportament arată că modelul B explorează mai mult pentru a găsi valori optime pe termen lung, chiar dacă inițial acest lucru duce la pierderi mai mari. .
Policy/Entropy	Entropia pentru ambele modele scade pe parcursul antrenamentului, indicând o reducere a explorării pe măsură ce agenții devin mai siguri pe acțiunile lor. Modelul B are o entropie inițială mai mare, sugerând că începe cu o explorare mai amplă, dar scade într-un ritm similar cu modelul B, indicând convergența către politici deterministe eficiente.

Grafic	Observatii si Interpretari
Environment/Episode Length	Modelul A prezintă episoade mai lungi, indicând o performanță mai stabilă în menținerea jocului pe parcursul antrenamentului. Modelul B are fluctuații mai mari în lungimea episodului.

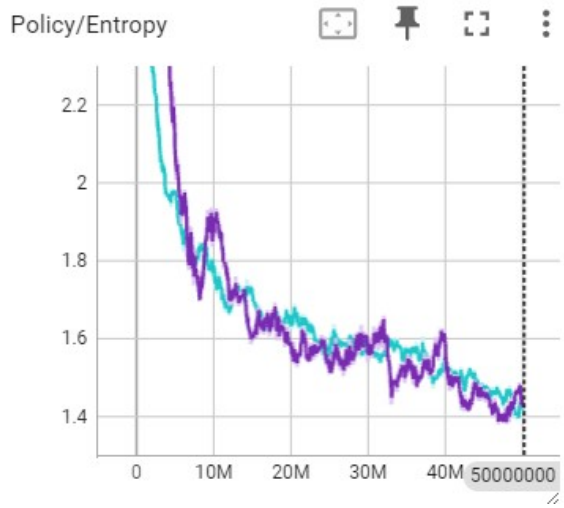

Figura 5.8: Entropie

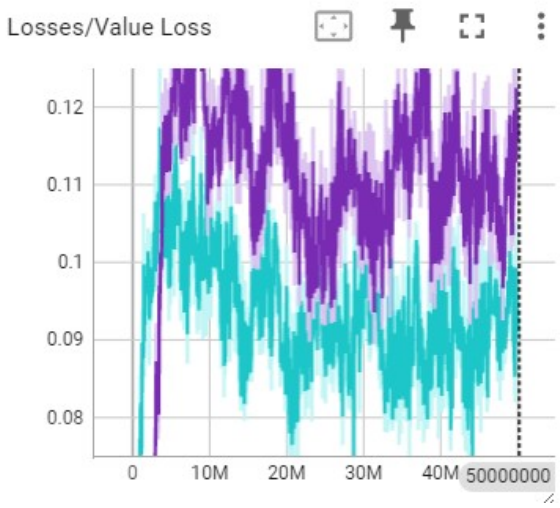

Figura 5.9: Pierdere a valorii

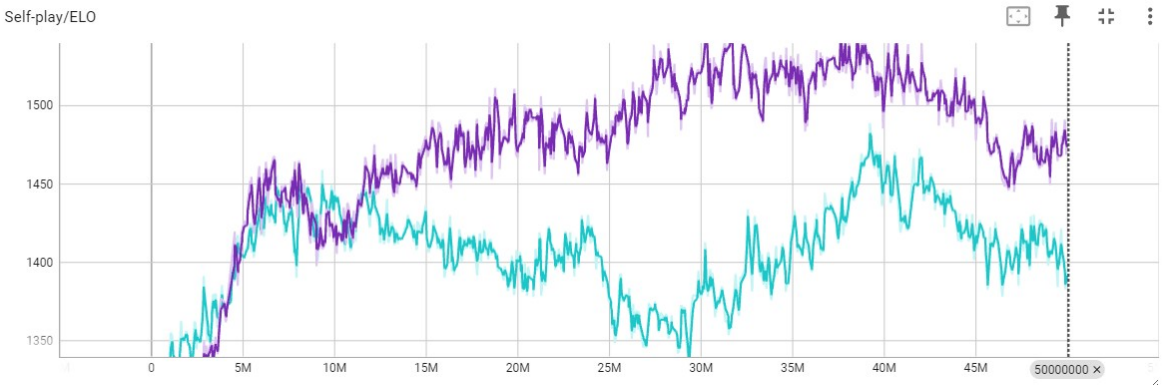

Figura 5.10: Scor ELO

PPO standard PPO îmbunătățit

În concluzie, modelul care utilizează algoritmul PPO împreună cu învățarea curriculum demonstrează o superioritate clară față de modelul care utilizează algoritmul PPO simplu, atingând scoruri ELO mai mari și menținând o performanță mai stabilă pe termen lung. Învățarea curriculum facilitează o explorare mai agresivă și o adaptabilitate crescută, rezultând într-o convergență mai robustă și eficientă în medii complexe și dinamice. Alte moduri, precum învățarea fără adversari la început, nu s-au dovedit a crește performanțele acestui algoritm în mediul nostru.

Analiza performanțelor algoritmului POCA - cel mai performant model
În continuare, vom analiza o configurație de algoritm POCA, la finalul unei acțiuni ample de a căuta cei mai buni hiperparametri pentru acesta.

Grafic	Observatii si Interpretari
Policy/Entropy	Graficul arată o scădere constantă a entropiei în timpul antrenamentului, ceea ce indică faptul că modelul devine mai sigur în deciziile sale. O entropie mai mică sugerează că modelul este mai puțin aleatoriu și mai previzibil în acțiunile sale, ceea ce este un semn pozitiv de convergență și stabilitate. .
Losses/Baseline Loss	Pierderea rămâne destul de constantă, dar cu o tendință generală de scădere. Scăderea indică faptul că modelul este capabil să minimizeze erorile în predicțiile sale pe termen lung.
Losses/Value Loss	Pierderea valorii este, de asemenea, volatilă, dar există o tendință generală de scădere. Acest lucru indică faptul că modelul își îmbunătățește capacitatea de a prezice valoarea acțiunilor sale, ceea ce este crucial pentru luarea deciziilor eficiente.
Self-play/ELO	Graficul arată o creștere constantă a valorii ELO, ceea ce indică o îmbunătățire a performanței modelului în timp. ELO-ul ajunge la valori mai mari decât cele de la început, sugerând că modelul a învățat și s-a optimizat continuu.

Așadar, aceste argumente susțin ideea că această configurație este cea mai bună, comparativ cu toate celelalte metode prezentate anterior, datorită performanței sale superioare și a tendințelor pozitive observate în timpul antrenamentului. Creșterea constantă și semnificativă în scorul ELO demonstrează o îmbunătățire clară a performanței modelului. Reducerea entropiei indică faptul că modelul devine mai sigur și mai stabil în deciziile sale. Scăderile observate în pierderile de politică și de valoare sugerează că modelul devine mai eficient și precis în predicțiile și deciziile sale.

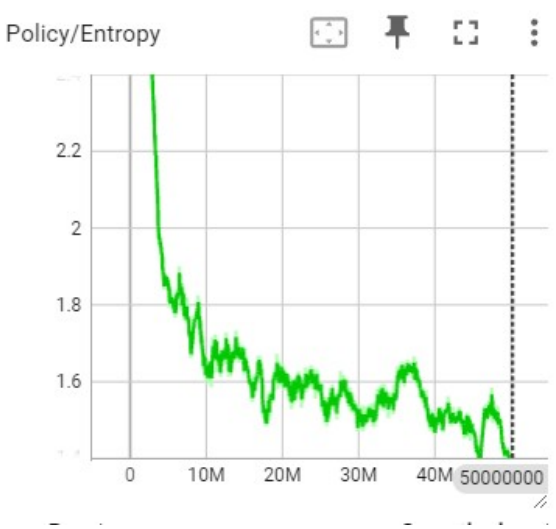

Figura 5.11: Entropie

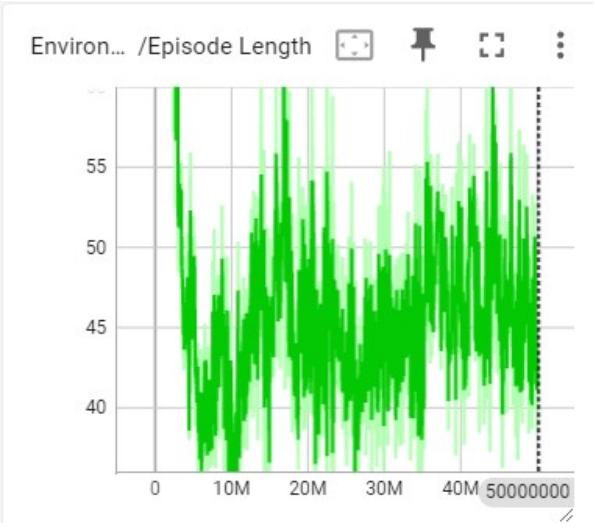

Figura 5.12: Grafic de lungime a episodului

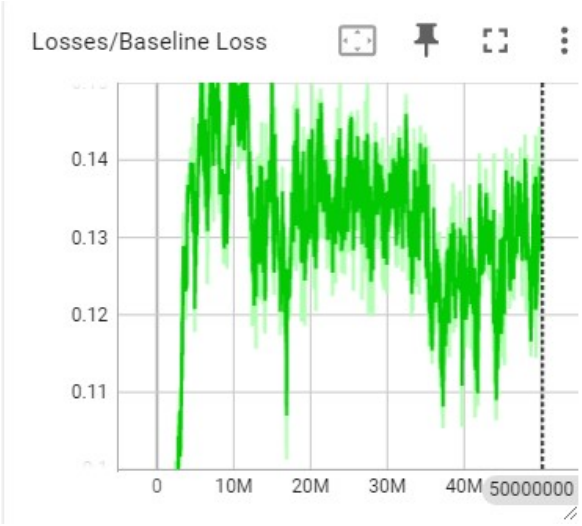

Figura 5.13: Grafic de pierdere de bază

Figura 5.14: Grafic de pierdere a valorii

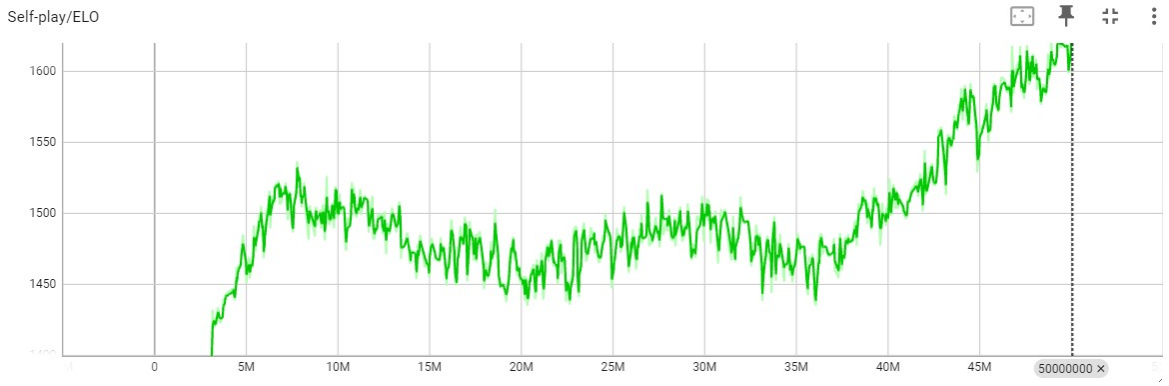

Figura 5.15: Grafic de scor ELO

Capitolul 6

Concluzii și idei viitoare

În concluzie, în această lucrare am definit și analizat performanțele unor algoritmi de învățare prin întărire și am explorat diferite configurații și hiperparametri pentru a optimiza performanțele acestora. Am reușit să creez un model care poate atinge performanțe bune, capabil să reproducă comportamente umane. Utilizând Unity, am putut simula medii complexe și dinamice, ceea ce a permis testarea și ajustarea modelul într-un mod eficient și precis.

În viitor, pentru a crește dificultatea mediului și a îmbunătăți performanțele agenților, voi implementa elemente avansate de fizică și diversificarea mediilor. Extinderea spațiului de acțiune și adăugarea de noi abilități vor obliga agenții să dezvolte strategii mai sofisticate. Agenții vor fi antrenați pentru a se adapta continuu la schimbările din mediu, folosind tehnici avansate de învățare prin întărire. Elementele de colaborare și competiție vor stimula dezvoltarea de strategii complexe, asigurând agenți mai inteligenți și adaptabili.