Cuprins

Listă figurilor

Listă tabele

Sinteza

Această lucrare este dezvoltată în cadrul Facultății de Electronică, Telecomunicații și Tehnologii Informaționale, parte a Universității Politehnica din Timișoara. Lucrarea reprezintă munca depusă pe parcursul a doi ani de zile 2020-2022.

Pentru partea teoretică, s-au parcurs mai multe articole științifice și alte resurse de documentare care ating subiecte precum *Machine Learning*, *Reinforcement Learning* și altele. De asemenea, un rol important pentru înțelegerea mai ușoară și mai rapidă a conceptelor de învățare automată l-a avut și materia de *Modelarea și Analiza datelor pentru Decizii de Management* din cadrul acestui programului de studiu *Tehnologii, Sisteme și Aplicații pentru eActivități*.

Pentru partea practică s-a realizat un experiment prin intermediul căruia se compară mai mulți algoritmi de învățare automată, atât supravegheași cât și reinforcement learning. Pe lângă partea de învățare automată mai este nevoie de cunoștințe pe partea de *Unity*, atât pe partea de design și interfață grafică, cât și pe partea logică de cod scris în limbajul de programare C#. De asemenea, au fost nevoie și de cunoștințe pe partea de realizare de *Flask API* prin care se realizează comunicarea dintre cele două medii: *Unity* și scriptul *Python*.

În final se prezintă concluziile, limitările, dezvoltările viitoare și referințele bibliografice.

Capitolul 1 – Introducere

1.1 Motivație

Având în vedere faptul că dorim să jucăm jocuri cât mai aproape de realitate, în care vrem să demonstrăm că suntem cei mai buni într-o confruntare directă, se pot crea agenți inteligenți care să simuleze adversarul. Dacă acești agenți sunt învățați într-un mod corect, vom avea parte de o adevărată provocare într-o luptă cu aceștia, fie că vorbim de curse cu mașini, de un meci de șah, de un meci de GO, etc.[1]

Așadar, această lucrare de diploma își dorește îmbinarea programării cu partea de gaming, și mai precis de a învăța un agent să concureze într-un mediu virtual. În cazul de față agentul este reprezentat de un pilot de kart iar mediul virtual este un circuit de karting. În această direcție se vor implementa două experimente în vederea învățării agentului cum să conducă folosind atât algoritmi de machine learning supravegheați cât și parte de reinforcement learning.

Direcția în care se îndreaptă lumea de automotive este spre autonomous driving, în care mașinile învață să conducă singure pe baza imaginilor captate din exterior, pe baza senzorilor, a poziției GPS și schimbul de informații cu celelalte vehicule aflate în trafic. Având în vedere acest aspect, această lucrarea poate fi considerată un punct de plecare și un prin pas în această direcție, în care se simulează partea de autonomous driving.[2]

În vederea realizării experimentului e vor folosii mai multe instrumente digitale:

- Unity Hub pentru mediul virtual
- Anaconda – pentru realizarea API-ului care comunică cu mediul din *Unity* și în care se realizează partea de ML
- Visual Studio Code – editor text pentru *Python* si *Mermeid*

Limbaje de programare utilizate:

- *C#* - în mediul *Unity*
- *Python* – pentru realizarea API-ului și pentru algoritmii de învățare automată

1.2 Obiective

Prin intermediul acestei lucrări de diplomă să se răspundă la câteva întrebări de cercetare:

Întrebarea 1: Ce tip de algoritmi de învățare automată sunt mai eficienți pentru învățarea agentului?

Întrebarea 2: Ce date are nevoie algoritmul ca și intrare?

Întrebarea 3: Este un agent antrenat cu algoritmi de învățare automată superiori față de o persoană umană?

Pornind de la aceste întrebări de cercetare, sunt necesare mai multe etape în vederea găsirii răspunsurilor la acestea. În cazul de față, agentul software este reprezentat de un kart dintr-un joc create în mediul *Unity* care are ca scop să învețe să conducă pe un circuit predefinit, care nu se modifică, atingând cele trei checkpoint-uri în cel mai scurt timp. În partea experimentală, învățarea agentului se va face folosind două tipuri de învățare automată.

Pentru a răspunde la întrebările de cercetare trebuie îndeplinite următoarele etape:

- Adaptarea codului din *Unity* pentru a facilita colectarea datelor din joc și controlul agentului programatic
- Realizarea infrastructurii pentru comunicarea dintre mediul *Unity* și codul în *Python* prin care se va face învățarea agentului
- Antrenarea modelului folosind algoritmi de învățare automată supravegheați
- Antrenarea modelului folosind Reinforcement Learning
- Compararea performanțelor agentului în cele două scenarii de mai sus

Capitolul 2: Partea introductivă

În cadrul acestui capitol se vor prezenta conceptele teoretice de care este nevoie în implementarea experimentelor, cum ar fi agenți software și domeniile în care se pot folosii acești agenți inteligenți. De asemenea, se vor prezenta și câteva lucrări conexe în care s-au realizat antrenarea unor agenți într-un mediu virtual.

2.1 Agenți Sotfware

Un agent software este un obiect software autonom, care are acces la informațiile din mediul virtual în care acesta se găsește.[3] În cazul acestui experiment, agentul software este reprezentat de un kart care are misiunea să obțină cel mai rapid timp pe traseul respectiv.

Agenții inteligenți sunt sisteme informatice care sunt plasați în diverse medii virtuale, în cazul de față, în lumi virtuale, care sunt capabili de acțiuni autonome în vederea îndeplinirii unor obiective bine definite.[1] Aceștia au ca scop rezolvarea problemelor din joc, care ar putea fi reprezentate de înfrângerea unui adversar în luptă sau navigarea printr-un labirint.[2]

Lumile virtuale din interiorul jocurilor video reprezintă un teren de joacă pentru o varietate mare de algoritmi, prin intermediul cărora se încearcă lucruri noi. În funcție de rezultatele obținute, aceste idei pot fi transferate și aplicate în viața reală. Scopul inteligenței artificiale este să creeze agenți inteligenți care să fie capabil să ia cele mai bune decizii, atât în mediile virtuale, cât și în lumea reală.[3]

În vederea antrenării acestor agenți, se pot utiliza jocuri deja existente, sau se pot crea jocuri de la zero, într-o multitudine de limbaje de programare, pentru a crea mediul de care avem nevoie.

În ultimul deceniu, s-au realizat din ce în ce mai multe lucrări în această direcție, datorită avansului tehnologic în care ne aflăm, fiind vorba de componente hardware avansate pentru utilizarea algoritmilor dezvoltați care au nevoie de putere mare de calcul. Astfel, au apărut jocuri în care grafica este din ce în ce mai aproape de realitate.

Așadar, prin intermediul agenților inteligenți, se pune accentul pe obținerea unui comportament mai natural al caracterelor din jocuri, prin folosirea de **fuzzy logic** și rețele neuronale. Astfel, se obțin caractere care se comportă natural și care se încadrează în decorul jocului într-un mod mai plăcut.[4]

2.2 Domenii de aplicabilitate

Din punct de vedere al domeniului de aplicabilitate, agenți software inteligenți pot fi folosiși în foarte multe domenii, cum ar fi:

- Furnizarea de recomandări

Agenții inteligenți joacă un rol important în partea de furnizare de recomandări. Prin intermediul acestora, se poate realiza prin mai multe tehnici, cum ar fi cea bazată pe conținut, cea colaborativă sau un mix între cele două. [4]

- Educație

Agenții inteligenți pot fi utilizați și în această ramură, mai ales în zona de cursuri online unde aceștia realizează mai multe sarcini, cum ar fi planificarea procesului educațional, creare de conținut educațional, evaluarea participanților la cursuri, etc.[5]

- Jocuri video

Implicarea agenților inteligenți în jocurile video este una foarte vastă, acești făcându-și prezența în jocuri clasice precum șahul[6], jocul Go[7], sau alte jocuri mai complexe cum ar fi FIFA [8]. De asemenea, agenți inteligenți pot înfrânge și cea mai bună echipă într-un joc MOBA(Multiplayer Online Battle Arena), cum s-a întâmplat în cazul jocului Dota 2 [9].

2.3 Lucrări conexe

Mai jos sunt prezentate câteva soluții software existente în care agenții software au fost învățați să joace diferite jocuri.

Monte-Carlo Tree Search (MCTS) a fost implementat pentru a-l învăța pe **Pac-Man** cum să joace și cum să obțină un scor cât mai mare. În articolul realizat de către Tom Pepels, este foarte importantă consistența rezultatelor obținute de agent. Aceasta reiese din faptul că agentul a obținut un scor ridicat, concurând atât contra unor echipe de fantome mai slabe, cât și împotriva unora mai puternice. Fiind un joc pe mai multe nivele, agentul trebuie să țină cont atât de obiectivele pe termen scurt (short-term goals) cât și de cele pe termen lung (long-term goals).[10]

În cel de-al doilea articol, care este realizat de către Ibrahim Fathy, ni se prezintă un agent inteligent bazat atât pe RL cât și pe **Online Case-Based Planing** (OLCBP), acesta fiind evaluat prin intermediul jocului **Wargus**. Prin hibridizarea celor doi algoritmi, alegerea planurilor de către agent se face într-un mod mai eficient și cu o rată ridicată de succes.[11]

Pornind de la RL, pentru ca agenții să rezolve task-urile împreună și în vederea obținerii unui câștig maximal în jocuri repetitive, Zhen Zhang propune algoritmul **Probability of Maximal Reward based on the Infinitesimal Gradient Ascent** (PMR-IGA). Un astfel de rezultat se obține indiferent de condițiile inițiale, având un număr finit de jucători și un număr finit de acțiuni. Folosind o astfel de abordare, ne permite abordarea unor probleme mai practice. În articol, algoritmul prezintă rezultate bune și o consistență în scenariile stocastice.[12]

Articolul realizat de către Damon Daylamani-Zad și Marios C. Anfelides se focusează pe folosirea **Deep Reinforcement Learning** (DRL), cu precădere a PPO, cu scopul de a descoperii impactul pe care îl are altruismul și egoismul asupra credibilității agenților. În urma experimentelor, jucătorii găsesc comportamentul altruist mai credibil decât cel egoism.[13]

În articolul realizat de către Yoshina Takano, s-au implementat doi agenți într-un joc de lupte (figthing game), utilizându-se atât DRL cât și MCST. În cadrul acestui joc, unul din agentul implementat folosind DRL poate fi considerat un agent dublu, deoarece acesta are atât un rol ofensiv, cât și defensiv. S-a realizat o comparație între cei doi, și a rezultat că agentul dublu DRL a avut cu 30% mai multe șanse de câștig ca agentul simplu MCTS. [14]

Tot prin intermediul DRL se pot antrena agenți în jocuri de tipul **First-Person Shooter**. În experimentul realizat de către Paulo B. S. Serafim s-a realizat împărțirea agenților în 3 grupuri de câte 5. Primele două grupuri s-au antrenat concurând una versus celeilalte, iar ultima dintre ele s-a antrenat concurând versus oponenți care se comportă aleatoriu. Din rezultate reiese faptul că agenții care s-au antrenat contra unor agenți autonomi au avut rezultate mult mai bune decât cei care s-au antrenat cu oponenți aleatorii. De asemenea, un lucru interesant este faptul că agenții au dobândit și comportamente foarte complexe, cum ar fi anticiparea mișcărilor adversarului.[8]

După parcurgerea articolelor de mai sus, s-a observat că într-un procent ridicat, au fost preferat utilizarea de Reinforcement Learning sau Deep Reinforcement Learning pentru învățarea agentului. De asemenea, în articolele parcurse s-a preferat învățarea unui singur agent. În partea de experiment, se vor implementa două experimente, unul folosind algoritmi de machine learning supravegheați, iar în celălalt se va folosi un algoritm de reinforcement learning.

2.4 Platforme software

Mai jos sunt descrise câteva platforme software care facilitează crearea mediului și antrenarea agenților.

Anaconda este o distribuție pentru știința datelor Python/R care conține conda, un manager de pachete și medii, care ajută utilizatorii să gestioneze o colecție de peste 7000 de pachete open-source disponibile.[18]

Flask este un framework web pe care îl putem folosi pentru a construi cu ușurință o aplicație web. *Flask* oferă pachetele și modulele necesare astfel încât, în calitate de dezvoltator, trebuie să ne concentrăm doar pe logica propriu-zisă a aplicației.[19]

Scikit-learn este un modul *Python* care integrează o gamă largă de algoritmi de învățare automată de ultimă generație pentru probleme supravegheate și nesupravegheate. Accentul este pus pe ușurința în utilizare, performanță, documentație și coerența API. Acest modul va fi folosit pentru antrenarea modelului de învățare automată supravegheată.[20]

Motorul de joc Unity este dezvoltat de *Unity* Technologies și integrează un motor de randare personalizat cu motorul de fizică nVidia PhysX și Mono, dar și cu implementarea bibliotecilor .NET open source a celor de la Microsoft. Editorul *Unity* este foarte ușor de utilizat. Conținutul acestuia este listat într-un arbore și se poate adăuga la mediu într-o manieră de tip drag-and-drop. Fiecărui obiect din interfață i se pot atribui mai multe scripturi scrise în C#.[21]

OpenAi Gym este un set de instrumente pentru cercetarea în domeniul de reinforcemenet learning. Aceasta include o colecție tot mai mare de probleme de referință care expun o interfață comună. Acesta se concentrează pe episoade. Scopul agentului este să maximizeze recompensa totală și de a atinge un nivel ridicat de performantă în cât mai puține episoade posibile.[22]

Datorită faptului că există deja o scenă care pune la dispoziție kartul și circuitul și am experiență în limbajul C#, s-a ales *Unity* pentru gestionarea jocului. Pentru partea de realizare a infrastructurii de comunicare între *Unity* și scripturile *Python* se vor folosii soluțiile *Anaconda* și *Flask API.*

Pentru antrenarea agentului, pentru algoritmii de învățare automată se va folosi *Scikit-Learn*, deoarece are o documentație și o comunitate bine pusă la punct. Pentru partea de reinforcement learning se va utiliza *OpenAI Gym*, deoarece este ușor de folosit și pune la dispoziție o sumedenie de exemple din care te poți inspira.

Capitolul 3: Învățare automată

În cadrul acestui capitol, se prezintă concepte teoretice de învățare automată, se face o clasificare a acestora după modul de antrenare și se prezintă soluțiile software folosite pentru realizarea experimentului.

3.1 Machine Learning

Învățarea automată este un domeniu de actualitate în zilele noastre, fie că vorbim de domenii precum Automotive, Cyber Security, Data Science, detecții de imagini, etc. Din punct de vedere al definițiilor, am expus mai jos două dintre ele [23].

O definiție mai generală a machine learning este următoarea [23]:

- Machine learning-ul este domeniul de studiu care oferă abilitatea calculatoarelor să învețe fără ca acestea să fie programate în mod explicit (Arthur Samuel, 1958)

Există și o definiție orientată spre tehnică care zice că:

- Unui program software îi este spus să învețe din experiența E respectând niște cerințe C și având niște unități de măsură a performanței P. Dacă performanțele respectă cerințele C, măsurate în P, atunci programul software își îmbunătățește experiența E. (Tom Mitchel, 1997)

Mai jos sunt prezentate cele principalele sisteme de învățare automată după modul de antrenare al acestora.

3.2.1 Învățare automată nesupravegheată (Unsupervised ML)

Metodele de învățare automată nesupravegheată sunt deosebit de utile în sarcinile de descriere, deoarece urmăresc să găsească relații într-o structură de date fără a avea un rezultat măsurat. Această categorie de învățare automata este denumită nesupravegheată deoarece îi lipsește o variabilă de răspuns care să poată supraveghea analiza.[24]

3.2.2. Învățare automată supravegheată (Supervised ML)

Metodele de învățare automată supravegheată se utilizează pentru a descrie sarcini de predicție, deoarece aceasta au scopul de prognoza și clasifica un anumit rezultat de interes (dacă o anumită persona este predispusă anumitor boli pe baza informațiilor medicale). Învățarea supravegheată a fost aplicată structurilor mari de date, deoarece în vederea obținerii unei acurateț̦e bune este nevoie să “hrănim” modelul de ML cu un set mare de date în ciclul de învățare.[24]

În această categorie de ML, datele sunt împărțite în mai multe seturi. Un prim set este cel de antrenare care sunt introduse în model. Al doilea set de date este cel de testare, prin intermediul căruia se determină performanța modelului, prin calcularea mai multor indicatori, cum ar fi acuratețea. De obicei, raportul dintre cele două seturi este de 9 la 1.

Setul de date de antrenare cu care este “hrănit” algoritmul de ML poartă numele de etichete (labels).

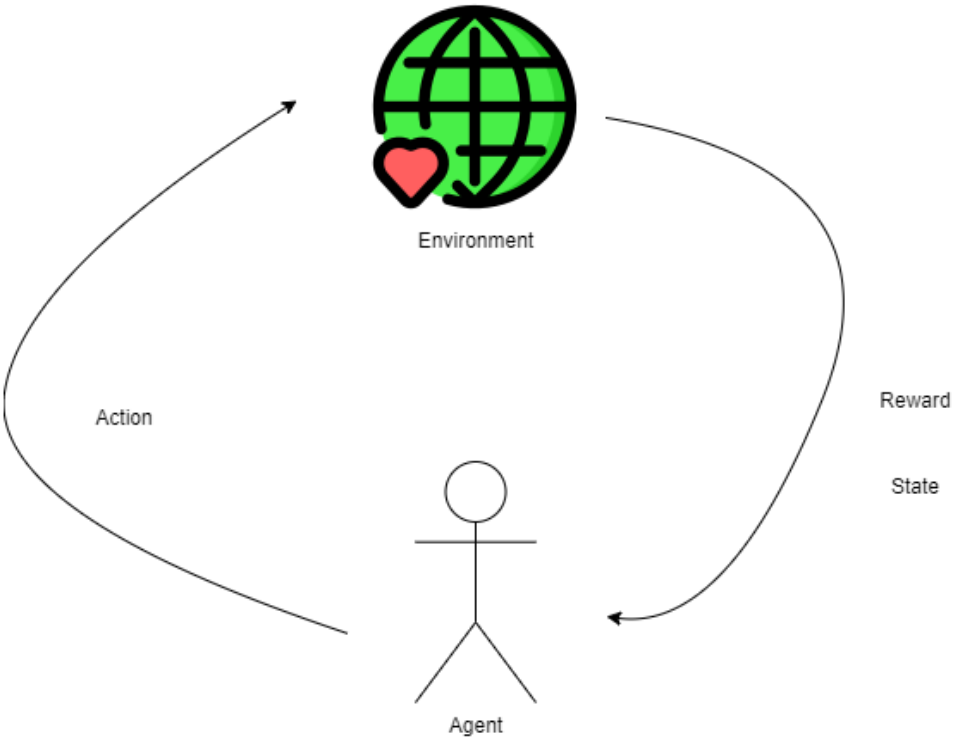
3.2.3 Reinforcement Learning

Problemele de tip *Reinforcement Learning* implică învățarea agentului prin corelarea situațiilor cu acțiuni, astfel încât să se maximizeze recompense care are au o valoare numerică. În mod esențial, acestea sunt probleme în buclă închisă, deoarece acțiunile luate de sistemul de învățare influențează intrările ulterioare. [25]

Mai mult, în acest tip de învățare, nu i se zice modelului ce acțiuni să întreprindă, ci trebuie să descopere singur ce acțiuni aduc cea mai mare recompensă printr-o serie de încercări. În cele mai multe situații, acțiunile luate de agent nu afectează numai recompensa imediată, ci și următoarele situații, respective influențând toate recompensele ulterioare. [25]

Așadar, în reinforcement Learning, un agent software primește ca input niște observații și alege niște acțiuni într-un mediu virtual. Obiectivul acestuia este să învețe cum să se comporte în anumite situații prin maximizarea recompenselor în timp. Ne putem gândii la câștigurile pozitive (recompensele) ca fiind o plăcere și la câștigurile negative (penalizările) ca fiind niște dureri. Pe scurt, agentul ia niște decizii în mediul respective și învață exersând, încercând astfel să crească plăcerea și să scadă durerea. Acest flux se poate observa in Figura 3.1.

Figura 3.1 Flux reinforcement learning



Mai jos sunt descrise două exemple de situații în care se poate folosii reinforcement learning [23]:

1. Agentul poate să fie un program care controlează un robot real. În acest caz, mediul este lumea reală, agentul observând mediul prin intermediul unor senzori cum ar fi camere și senzori radar. Acțiunile agentului pot fi trimiterea unor semnale pentru activarea/dezactivarea motoarelor. Recompensa poate să fie pozitivă în cazul în care agentul se deplasează înspre o țintă, respectiv o penalizare dacă se îndepărtează de țintă.
2. Un al exemplu de agent poate fi un program care îl controlează pe *Ms. Pac Man*. Aici mediul virtual este simulația jocului Atari, acțiunile sunt cele 9 posibile poziții ale joystick-ului. Observațiile se iau din capturile de ecran ale jocului, realizându-se astfel o procesare de imagini, iar recompensele sunt reprezentate de scorul obținut în joc.

Capitolul 4: Proiectarea experimentului

4.1 Obiectivele experimentului

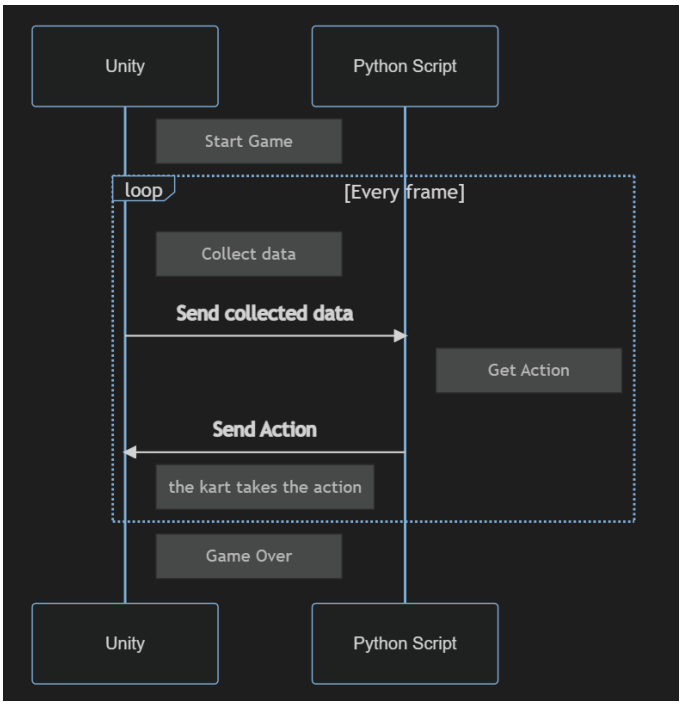
Experimentul presupune învățarea unui agent sub forma unui kart ca să conducă pe un circuit. Se vor aborda două implementări, una în care agentul învață prin intermediul unor algoritmi de învățare automată supravegheată dar și prin algoritmi de reinforcement learning.

Se dorește stabilirea abordării celei mai potrivite din cele 2 tipuri de învățare automată, în funcție de scenariul și mediul din *Unity*.

4.2 Arhitectura experimentului

Experimentul este alcătuit din mai multe componente în mai multe limbaje de programare, cum ar fi *C#*, *Python*, etc. In Figura 4.1 este prezentat fluxul de date dintre aceste componente.

Figura 4.1 Fluxul datelor între componente



4.2.1 Circuitul din *Unity*

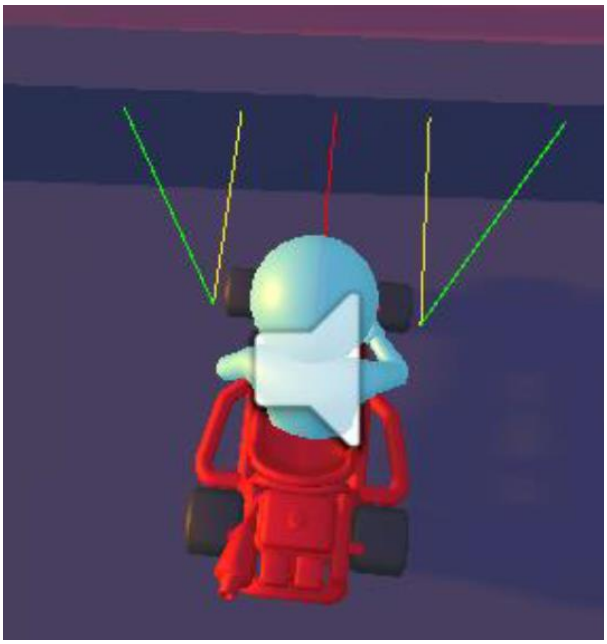
Prima componentă este cea de *Unity* care folosește limbajul *C#.* În vederea învățării agentului să piloteze kartul, se va folosi un mediu virtual deja existent în *Unity* care conține kartul și mai multe scenele. Contribuția autorului este să adapteze mediul și codul din *Unity* pentru a putea prelua datele necesare din acel mediu, dar și ca să poată să controleze kart-ul programabil.

Acest mediu pune la dispoziție un kart și un circuit. Scopul kartului este să treacă prin trei checkpoints într-un timp cât mai scurt. În momentul în care începe jocul începe un timer de 60 de secunde să scadă. Pentru fiecare checkpoint atins, se mai primesc încă cinci secunde în plus. Jocul se termină în momentul în care timer-ul ajunge la zero sau kartul trece prin cele trei checkpointuri.

Pentru automatizarea și colectarea datelor din mediul *Unity*, a fost nevoie de adaptare de cod. În primul rând, este nevoie de colectarea unor date care erau deja prezente în mediu, cum ar fi timpul curent, poziția kartului pe cele trei axe 0x, 0y și 0z și starea jocului. În vederea învățării kartului să conducă singur a fost nevoie de adăugarea unor elemente suplimentare.

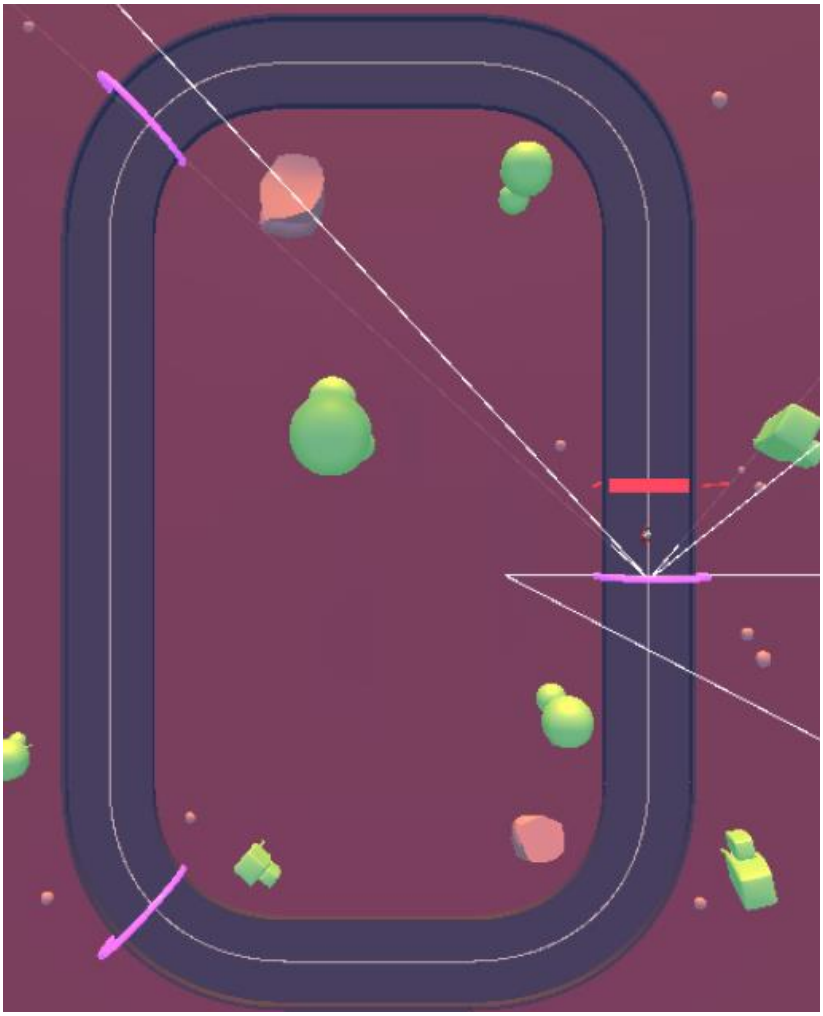
Primele elemente adăugate au fost cinci senzori, plasați în fața kartului, după cum se poate observa și în Figura 4.2. Acești senzori returnează distanța până la un obstacol care este în direcția lor. În cazul acestui experiment, valoarea maximă a senzorului a fost de 5.0f. În cazul în care nu se află nimic în aria de acțiune a unui senzor, acesta va returna 5, în caz contrar acesta returnează distanța până la obiectul respectiv.

Figura 4.2 Distribuția senzorilor frontali



Un alt element esențial este zona circuitului. Circuitul are o formă de oval, asemănătoare circuitelor din Nascar. Forma circuitului se poate observa în figura de mai sus. Configurația circuitului este prezentă în Figura 4.3. În vederea stabilirii faptului că kartul se deplasează înainte față de direcția normal de circuit, s-a împărțit circuitul în patru zone, una pentru fiecare latură. Pentru calcularea direcției de deplasare a kartului, se calculează diferența dintre poziția actuală (din acest cadre) și poziția din cadrul trecut, în funcție și de zona în care se află kartul.

Figura 4.3 Forma circuitului

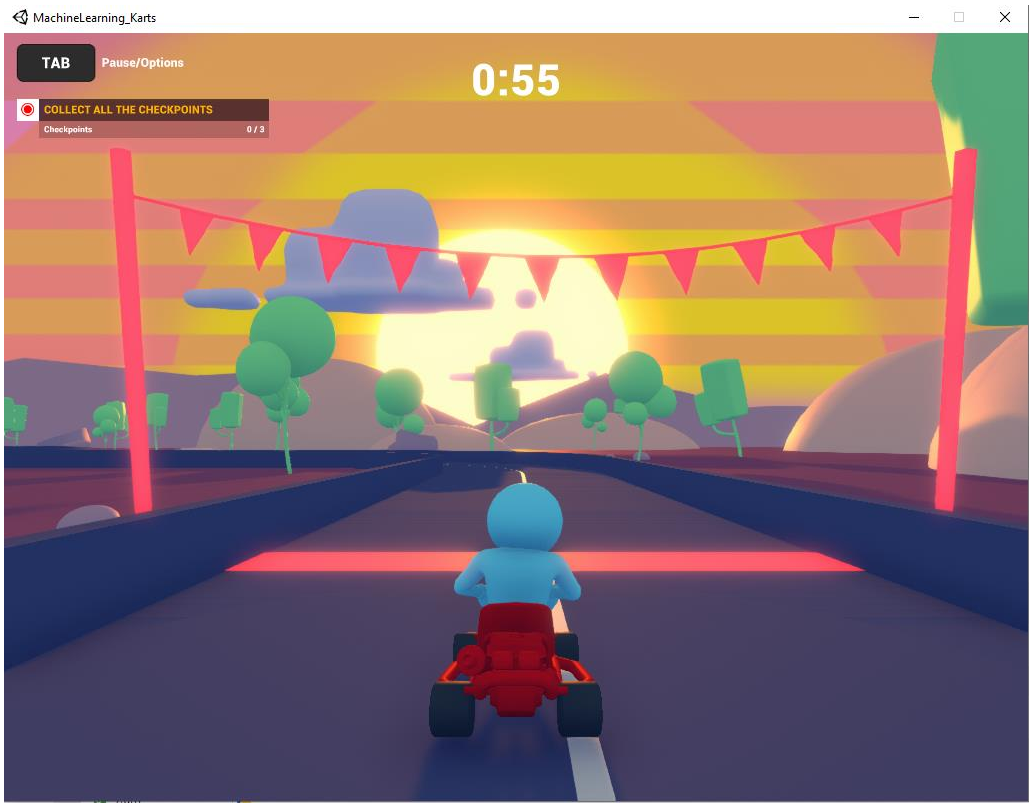


Un alt element important este decizia luată de kart, care este practic inputul acestuia. Kartul primește ca și input doar direcția de deplasare, în față, în spate, la stânga sau la dreapta. Pentru a reuși controlul kartului direct din limbajul C#, a fost nevoie de o adaptare în fișierul *KeyboardInput.cs*. Mai precis, în momentul în care un utilizator uman controlează kartul, prin apăsarea unei săgeți pentru o direcție se preia valoarea pe axa respectivă. Pentru deplasarea înainte / înapoi se folosește axa verticală, iar pentru deplasarea stânga / dreapta se folosește axa orizontală. Ambele axe pot să aibă valori între -1 și 1.

Valoarea axei respective se poate înțelege în felul următor. Pe axa verticală, dacă nici una din cele două săgeți nu este apăsată atunci valoarea axei este 0. Dacă se apasă

săgeata *Up Arrow* atunci valoarea axei va fi 1. În cazul în care se apasă *Down Arrow* valoarea axei va fi -1. În mod similar se întâmplă lucrurile și pe axa orizontală.

Figura 4.4 Imagina văzută de jucător



4.2.2 Colectarea datelor în C#

Pentru învățarea agentului cum să conducă este nevoie de mai multe date de intrare și se ieșire. Aceste date sunt preluate la fiecare cadru din joc, se trimit către scriptul în *Python* și procesate, după care sunt trimise înapoi în *Unity* unde se acționează agentul.

Din punct de vedere al modelului, Kartul prezintă următoarele proprietăți, prezente în Tabelul 4.1.

În funcție de valoarea state-ului, se ia decizia activării uneia dintre direcțiile de mai jos sau a unei combinații ale acestora: în față, în spate, în stânga sau în dreapta.

Tabel 4.1 Proprietăți kart

Proprietate	Descriere	Intrare/ Ieșire	Tip
Time	Timpul din joc	Intrare	Float
xPos	Pozitia pe axa Ox	Intrare	Float
yPos	Pozitia pe axa Oy	Intrare	Float
zPos	Pozitia pe axa Oz	Intrare	Float
leftSideDistance	Distanța până la obstacol	Intrare	Float (0<=dis<=5)
leftForwardDistance	Distanța până la obstacol	Intrare	Float (0<=dis<=5)
centralForwardDistance	Distanța până la obstacol	Intrare	Float (0<=dis<=5)
rightForwardDistance	Distanța până la obstacol	Intrare	Float (0<=dis<=5)
rightSideDistance	Distanța până la obstacol	Intrare	Float (0<=dis<=5)
Zone	Zona circuitului	Intrare	String
movingForward	Direcția de deplasare	Intrare	Boolean
gameOver	Starea jocului	Intrare	Boolean
State	Decizia luată de model	Ieșire	Int (0<=state<=15)

4.2.3 Scriptul în *Python – Flask API*

Scriptul în *Python* a fost implementat folosind pachetul software anaconda, prin intermediul căruia s-a creat un mediu care va conține pachetele necesare pentru implementarea soluției. O primă componentă a scriptului *Python* este reprezentată de *Flask API*, prin intermediul căruia s-a realizat o interfață API de tip *POST*, prin intermediul căreia datele de intrare sunt trimise de către componenta *Unity*, iar răspunsul este dat de decizia luată de machine learning.

Această interfață de tip *POST* este apelat de către joc la fiecare cadru, din interiorul metodei *Update* din *Unity*. Body-ul acestei interfețe este alcătuit din toate informațiile colectate din mediu, necesare ulterior pentru luarea deciziei. Răspunsul interfeței reprezintă decizia pe care kartul o ia în pasul următor. Atât body-ul cât și răspunsul acestuia sunt sub formă de *JSON*.

Această componentă este folosită în ambele experimente, atât în cazul algoritmilor de învățare automată nesupravegheată, cât și când vorbim de reinforcement learning.

4.2.4 Sistemele de decizie implementate

Pentru a răspunde la întrebarea care algoritmi de învățare automată sunt mai performanți, , se vor realiza două experimente prin care un agent software, în cazul de fată un kart dintr-un mediu virtual, va învăța să conducă în mediul respectiv.

Cele două experimente vor folosi același circuit în mediul *Unity*, însă în primul experiment agentul va lua decizia folosind un model de machine learning supravegheat, prin intermediul a doi algoritmi: *Random Forest* și *Decision Tree*. În cel de-al doilea exemplu, agentul va învăța printr-o abordare de reinforcement learning prin recompense și penalizări.

4.2.4.1 Algoritmi supravegheați

O primă categorie de algoritmi pentru învățare automată folosiți în acest experiment sunt cei de tip supervizați. Aceștia se bazează pe un set de date deja colectat din joc, asupra cărora se aplică procedeul de antrenare și validare.

Pentru obținerea acestui set de date, un utilizator parcurge traseul de circuit, alegând cel mai optim traseu. La fiecare episod, datele din interiorul mediului de *Unity* sunt salvate într-un fișier *.csv*. După ce se completează câteva scenarii, fișierele *.csv* sunt fuzionate într-un singur fișier, asupra căruia se realizează operații de curățare.

După ce s-a realizat curățarea datelor, se aleg caracteristicile pentru algoritmul de machine learning. În acest caz, caracteristicile sunt următoarele: poziția kartului pe cele 3 axe(0x,0y și 0z), timpul din joc, direcția de deplasare a kartului, datele de la cei cinci senzori, starea jocului.

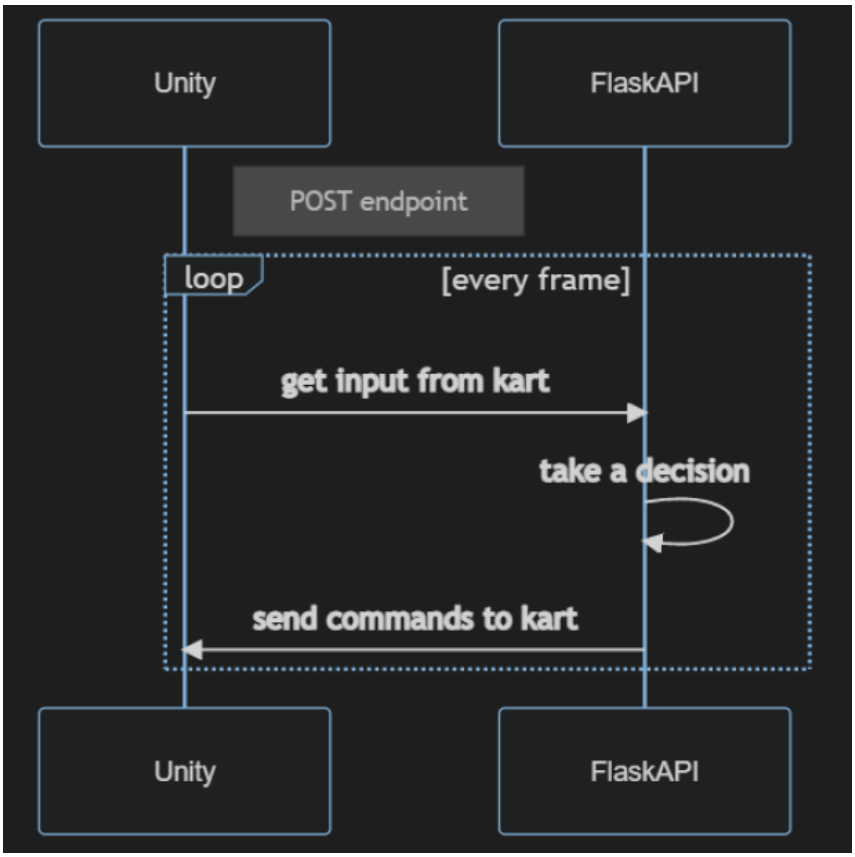
În exemplificarea experimentului, se vor folosii doi algoritmi supravegheați, și anume *Decision Tree* și *Random Forrest*.

În cazul ambilor algoritmi, fiind vorba de algoritmi de machine learning supravegheați, setul de date se împarte în date de antrenare și date de testare. În ambele cazuri s-a folosit un raport de 7 la 3.Din punct de vedere al acurateței al performanței modelului, se obține o valoare de 83%, reprezentând precizia cu care modelul ia aceeași decizie cu utilizatorul uman. După ce s-a realizat cu succes învățarea modelului, acesta a fost supus conducerii kartului în mai multe episoade.

Fluxul de date este prezentat și în figura de mai jos. După cum se poate observa îm Figura 4.5, la fiecare cadru din joc se trimit informațiile kartului înspre scriptul de *Flask API*. Datele respective sunt folosite ca input pentru modelul de machine learning antrenat în prealabil, acesta furnizând decizia luată. Această decizie este reprezentată ca un număr întreg în intervalul 0-15. Această valoare se primește ca răspuns în *Unity*, unde este mapată pe 4 valori de tip boolean, una pentru fiecare direcție. Pe baza acestora, se trimite comanda cu care kartul continuă deplasarea.

Acești pași sunt repetați până în momentul în care kartul trece prin cele trei checkpoint-uri sau până în momentul în care expiră timpul.

Figura 4.5 Fluxul pentru învățarea automată supravegheată



4.2.4.2 Reinforcement Learning

Pentru învățarea agentului folosind tehnici de machine learning de tip reinforcement learning, este nevoie de folosirea altor interfețe din scriptul *Flask API* și de crearea mediului de antrenare de tip *OpenAi gym.*

Pentru realizarea mediului de antrenare, s-a realizat o structură de directoare având următoarea formă, prezentă în Figura 4.6. Pe lângă fișierul custom_env.py care conține implementarea modelului de reinforcement learning, avem nevoie de câteva fișiere de configurare cu numele *__init__.py* și de fișierul *setup.py.* De asemenea, mai avem nevoie și de fișierul *main.py* în care se definesc mai mulți parametri, cum ar fi de exemplu numărul de episoade și este practic punctul de declanșare a algoritmului de reinforcement learning. Conținutul acestui fișier se poate vedea în Figura 4.7.

Figura 4.6 Structura de directoare pentru reinforcement learning

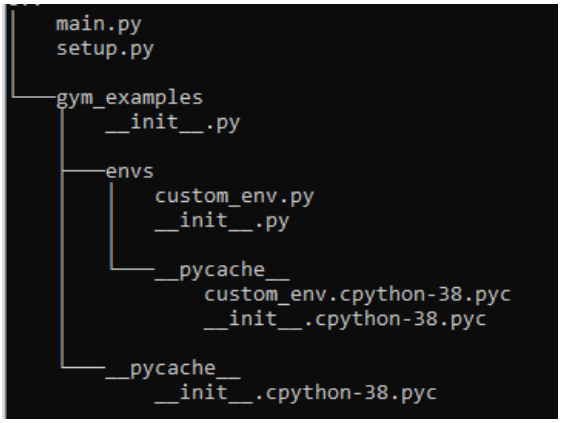


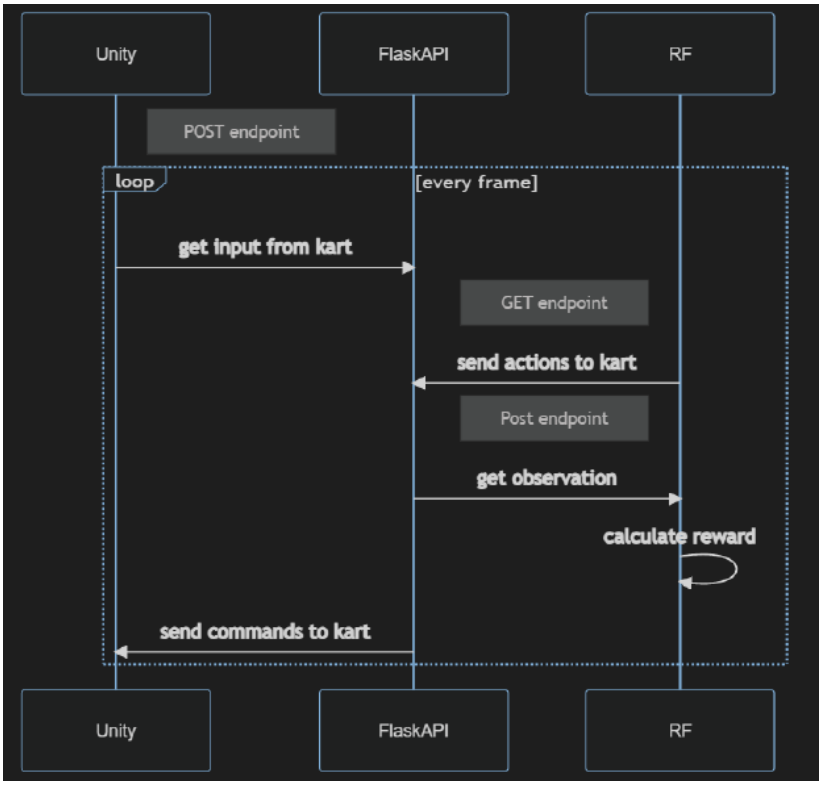
Figura 4.7 Fișierul *main.py*

```
import gym
import gym_examples
env = gym.make('CustomEnv-v1')
episodes = 0
observation = env.reset()
for _ in range(10000000):
    env.render()
    print(observation)
    action = env.action_space.sample()
    observation, reward, done, info = env.step(action)
    print(observation, reward, done, info)
    if done:
        observation = env.reset(  )
        print("Finished after{} timesteps".format(_+1))
        print("Episode nb{}".format(episodes))
env.close
```

Fluxul de date de la jocul din *Unity*, până la scriptul în *Python* de legătură, ajungând la scriptul de reinforcement learning se poate observa în Figura 4.8.

Diferența față de abordarea cu machine learning supravegheat o reprezintă scriptul de reinforcement learning în care se face învățarea agentului pe durata a unor mai multe episoade.

Figura 4.8 Fluxul datelor pentru reinforcement learning



Recompense și Penalizări

Modul în care agentul învață folosind algoritmi de reinforcement learning este prin intermediul unor recompense, recompense a unor penalizări. Dacă agentul performează bine, acesta primește o recompense pozitivă, iar în cazul în care se abate de la ținta lui, acesta primește o penalizare. În cazul de față, calcularea recompense / penalizării se face în funcția *__step__*, unde se observă comportamentul agentului după ce acesta a luat o anumită decizie în pasul anterior.

O primă recompensă pe care agentul o primește este dată de direcția de deplasare. Dacă acesta merge în direcția de deplasare a circuitului, atunci agentul primește o recompense de +100. În caz contrar, dacă acesta stă pe loc sau dacă se deplasează în direcția opusă, acesta primește o penalizare de -5000.

O altă recompensă este dată de distanța până la un obiect, în cazul de față a parapetului, a celor cinci senzori frontali. Penalizările pot fi observate și în Tabelul 4.2. Aceste penalizări/recompense se dau pe fiecare senzor independent.

Tabelul 4.2. Recompensele în funcție de distanța până la un obstacol

Distanța până la obstacol	Recompensă
dis >= 5	+ 10
dis < 5 && dis >= 2.5	-50
dis < 2.5 && dis >= 1	-1000
dis < 1	-5000

Dacă de exemplu kartul se deplasează pe direcția de deplasare a circuitului, un senzor returnează distanța între 2.5 și 5 și pentru celelalte patru distanța este mai mare decât 5, atunci kartul va avea următoarea recompensă, prezentă în relația 4.1, respectiv în relația 4.2.

$$\text{Recompensă} = 100 - 50 + 10 + 10 + 10 + 10 \quad (4.1)$$

$$\text{Recompensă} = 90 \quad (4.2)$$

4.2.6 Automatizarea mediului

În vederea învățării agentului folosind algoritmul de reinforcement learning, a fost nevoie de implementarea mai multor interfețe API, atât pentru pornirea și oprirea jocului. Jocul a fost exportat din mediul *Unity*, rezultând astfel un executabil pentru sistemul de operare de *Windows* pe care s-a realizat experimentul. Jocul poate fi exportat și pentru alte sisteme de operare, cum ar fi *Mac OS, Linux* și altele.

Pentru start-ul jocului se folosește interfața API */api/start-game* care este de tip *GET*. Prin intermediul acestuia se deschide executabilul exportat din *Unity*. După ce acesta se deschide apare imaginea din Figura 4.9, se așteaptă câteva secunde să apară ecranul de mediu, după care se trimite comanda care emulează tasta *Space* pentru a selecta butonul *Start game,* se așteaptă câteva secunde după care retrimite comanda pentru apăsarea butonului.

Figura 4.9 Fereastra de meniu al jocului

Interfața API */api/end-game* de tip *GET* este folosit pentru oprirea jocului. Acest lucru se realizează prin funcția TASKKILL de OS din *Python* unde se trimite și numele executabilului care se dorește a fi oprit, în cazul de față fiind *MachineLearning_Karts.exe* .

În Figura 4.10 se pot observa cele două interfețe, de pornire a jocului și de închidere a acestuia.

Figura 4.10 Interfețele de start-game, respectiv end-game

```
################################################################
#
# START/END GAME
#
################################################################
# A route to start the game
@app.route('/api/start-game', methods=['GET'])
def start_game():
    # start game
    #os.startfile(game_location)
    app1 = Application(backend="win32").start(cmd_line="C:\Poli\Dizertatie\Repo_Github\KartML\Export\ControlledByML\MachineLearning_Karts.exe")
    time.sleep(12)
    send_keys("{SPACE}")
    time.sleep(1)
    send_keys("{SPACE}")
    time.sleep(4)
    return jsonify("OK")

# A route to start the game
@app.route('/api/end-game', methods=['GET'])
def end_game():
    # start game
    os.system("TASKKILL /F /IM MachineLearning_Karts.exe")
    return jsonify("OK")
```

4.3 Comparație a sistemelor de decizie

În urma realizării experimentelor, atât a celor prin învățare supravegheată, cât și a celor prin folosirea de reinforcement learning, se poate realiza o comparație asupra performanțelor obținute de agent pe circuit.

O primă comparație de interes este comparația performanțelor dintre un utilizator uman și agentul antrenat prin algoritmii de machine learning supravegheați *Random Forest* și *Decision Tree*.

Având în vedere că acești algoritmi au nevoie de date colectate anterior, rezultatele obținute de aceștia sunt apropiate de rezultatele obținute de un utilizator uman, după cum se poate observa și din Tabel 4.3. Trebuie ținut cont că datele respective au fost colectate când un jucător uman a luat deciziile optime. În cazul în care alt utilizator se joacă și obține rezultate mai slabe, la fel va obține și agentul respectiv.

În urma antrenării agentului, în ambele cazuri s-a obținut în faza de testare a modelului acesta a obținut o acuratețe de aproximativ 83%, care reprezintă practic acuratețea cu care agentul ia o decizie pe care ar lua-o și utilizatorul uman.

Pentru partea de reinforcement learning, având în vedere că vorbim despre o învățare pe încercări consecutive, în care agentul încearcă mai multe acțiuni și este penalizat/recompensat, se poate observa o îmbunătățire a performanței acestuia după câteva sute de episoade, în care agentul reușește să treacă de câteva ori prin cele trei checkpoint-uri.

Partea de reinforcement learning va fi continuată prin creșterea numărului de episoade, pentru a vedea până unde poate să atingă maximul de performanță agentul.

Tabelul 4.3. Rezultatele obținute

	Utilizator uman	Random Forest	Decision Tree
Turul 1	27 secunde	29 secunde	29 secunde
Turul 2	27 secunde	28 secunde	28 secunde
Turul 3	28 secunde	27 secunde	28 secunde
Turul 4	26 secunde	27 secunde	29 secunde
Turul 5	26 secunde	28 secunde	28 secunde

Din punct de vedere al datelor necesare ca și intrare pentru algoritmi, avem nevoie de datele celor cinci senzori, starea jocului și de direcția de deplasare pentru experimentul ce folosește tehnici de reinforcement learning. Pentru cazul învățării automate supravegheate, mai avem nevoie ca și date de intrare, pe lângă cele enumerate mai sus de timpul din joc, poziția kartului pe cele trei axe și zona circuitului.

4.3.1 Machine Learning supravegheat vs Reinforcement Learning

Dacă ne raportăm la experimentele realizate, în cazul în care circuitul este unul cunoscut și nu suferă modificări, și avem acces la date precum poziția kartului pe circuit și date anterioare pe care se poate face antrenarea modelului, pot fi utilizați algoritmi de machine learning supravegheați care pot să aibă rezultate foarte bune, comparabile cu cele ale unui utilizator uman.

În cazul în care circuitul suferă modificări, este generat aleatoriu și nu avem acces la informații precum poziția kartului, ci doar senzorii frontali, cea mai bună variantă este folosirea de reinforcement learning.

Pornind de la aceste experimente se pot aduce dezvoltări viitoare, prezentate în capitolul dedicat mai jos în această lucrare.

Capitolul 5: Concluzii

5.1 – Contribuții proprii

În vederea realizării experimentelor pentru această temă în care se abordează învățarea automată a agenților cum să se comporte într-un mediu virtual, s-au adus mai multe contribuții proprii.

Din punct de vedere teoretic, s-a realizat un studiu cu privire la agenții inteligenți si la aplicabilitatea acestora. De asemenea, s-a realizat o analiză a unor lucrări conexe asupra algoritmilor folosiți pentru antrenarea acestor agenți. În final, s-a făcut și un studiu asupra platformelor software pe care să le folosim în realizarea experimentului.

Din punct de vedere practic, s-a adaptat codul în *Unity* pentru a permite colectarea datelor și furnizarea comenzilor către agentul din mediul virtual. Partea de comunicare între scriptul *Python* și *Unity* a fost realizată prin intermediul *Flask API-ului* prin utilizarea mediului *Anaconda*.

Antrenarea agenților s-a realizat prin implementarea unor algoritmi de învățare automată supravegheată, cu ajutorul librăriilor *Scikit-Learn*. Pentru partea de reinforcement learning s-a utilizat instrumentul *OpenAI Gym* care a facilitat și ușurat antrenarea agentului.

Codul poate fi găsit în repository-ul celor de la GitHub accesând următorul link: https://github.com/alexuicoaba/KartML/ .

5.2 – Dezvoltări viitoare

În viitor, având arhitectura actuală, se poate continua pe mai multe direcții. Cea mai importantă direcție este creșterea numărului de episoade necesare ca agentul să învețe să conducă folosind reinforcement learning. Este de interes observarea unor îmbunătățiri vizibile după un număr crescut de episoade.

Altă direcție poate fi trecerea de la un mediu de tip single agent, la unul de tip multi-agent. De interes în această abordare ar fi de văzut modul în care agenții concurează între ei.

O altă direcție poate fi generarea aleatorie de circuit. În acest mod se dorește observarea evoluție agentului în momentul în care acesta nu cunoaște traseul în prealabil. În cazul acestei abordări, din cele două implementări, se va putea folosii doar învățarea agentului folosind reinforcement learning, deoarece nu a existat o persoană umană de pe urma căreia să învețe agentul folosind tehnici de învățare automată supravegheată.

O ultimă direcție poate fi mărirea numărului de tururi de circuit pe care agentul trebuie să le completeze. În momentul de față, agentul trebuie să treacă print trei checkpoint-uri care reprezintă practic un singur tur de circuit. Este de interes, în cazul în care vor fi mai multe tururi, de urmărit modul în care performează agentul de la tur la tur.

Bibliografie

[1] M. Lent *et al.*, 'Intelligent Agents in Computer Games', Sep. 1999.
[2] S. Shalev-Shwartz, S. Shammah, and A. Shashua, 'Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving'. arXiv, Oct. 11, 2016. Accessed: Jun. 16, 2022. [Online]. Available: http://arxiv.org/abs/1610.03295
[3] G. S.Bhamra, A. K. Verma, and R. B. Patel, 'Intelligent Software Agent Technology: An Overview', *IJCA*, vol. 89, no. 2, pp. 19–31, Mar. 2014, doi: 10.5120/15474-4160.
[4] S. Vijay and A. Sethi, 'Congestion Aware and Stability Based Routing with Cross-Layer Optimization for 6LowPAN', Apr. 2020.
[5] G. Moise, 'THE ROLE OF INTELLIGENT AGENTS IN ONLINE LEARNING ENVIRONMENT', p. 7.
[6] N. Lassabe, S. Sanchez, H. Luga, and Y. Duthen, 'Genetically programmed strategies for chess endgame', in *Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06*, Seattle, Washington, USA, 2006, p. 831. doi: 10.1145/1143997.1144144.
[7] J.-B. Hoock, C.-S. Lee, A. Rimmel, F. Teytaud, M.-H. Wang, and O. Teytaud, 'Intelligent Agents for the Game of Go', *Computational Intelligence Magazine, IEEE*, vol. 5, pp. 28–42, Dec. 2010, doi: 10.1109/MCI.2010.938360.
[8] M. P. P. Faria, R. M. S. Julia, and L. B. P. Tomaz, 'Evaluating the Performance of the Deep Active Imitation Learning Algorithm in the Dynamic Environment of FIFA Player Agents', in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Dec. 2019, pp. 228–233. doi: 10.1109/ICMLA.2019.00043.
[9] OpenAI *et al.*, 'Dota 2 with Large Scale Deep Reinforcement Learning'. arXiv, Dec. 13, 2019. Accessed: Jun. 16, 2022. [Online]. Available: http://arxiv.org/abs/1912.06680
[10] T. Pepels, M. H. M. Winands, and M. Lanctot, 'Real-Time Monte Carlo Tree Search in Ms Pac-Man', *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 3, pp. 245–257, Sep. 2014, doi: 10.1109/TCIAIG.2013.2291577.
[11] I. Fathy, M. Aref, O. Enayet, and A. Al-Ogail, 'Intelligent online case-based planning agent model for real-time strategy games', in *2010 10th International Conference on Intelligent Systems Design and Applications*, Nov. 2010, pp. 445–450. doi: 10.1109/ISDA.2010.5687225.
[12] Z. Zhang, D. Wang, D. Zhao, Q. Han, and T. Song, 'A Gradient-Based Reinforcement Learning Algorithm for Multiple Cooperative Agents', *IEEE Access*, vol. 6, pp. 70223–70235, 2018, doi: 10.1109/ACCESS.2018.2878853.
[13] D. Daylamani-Zad and M. C. Angelides, 'Altruism and Selfishness in Believable Game Agents: Deep Reinforcement Learning in Modified Dictator Games', *IEEE Transactions on Games*, pp. 1–1, 2020, doi: 10.1109/TG.2020.2989636.
[14] Y. Takano, S. Ito, T. Harada, and R. Thawonmas, 'Utilizing Multiple Agents for Decision Making in a Fighting Game', in *2018 IEEE 7th Global Conference on Consumer Electronics (GCCE)*, Oct. 2018, pp. 594–595. doi: 10.1109/GCCE.2018.8574675.
[15] H. Baier, A. Sattaur, E. J. Powley, S. Devlin, J. Rollason, and P. I. Cowling, 'Emulating Human Play in a Leading Mobile Card Game', *IEEE Transactions on Games*, vol. 11, no. 4, pp. 386–395, Dec. 2019, doi: 10.1109/TG.2018.2835764.
[16] Z. Wei, D. Wang, M. Zhang, A. Tan, C. Miao, and Y. Zhou, 'Autonomous Agents in Snake Game via Deep Reinforcement Learning', in *2018 IEEE International Conference on Agents (ICA)*, Jul. 2018, pp. 20–25. doi: 10.1109/AGENTS.2018.8460004.
[17] D. Daylamani-Zad, L. B. Graham, and I. T. Paraskevopoulos, 'Swarm intelligence for autonomous cooperative agents in battles for real-time strategy games', in *2017 9th*

International Conference on Virtual Worlds and Games for Serious Applications (VS-Games), Sep. 2017, pp. 39–46. doi: 10.1109/VS-GAMES.2017.8055809.
[18] 'Anaconda Distribution — Anaconda documentation'. https://docs.anaconda.com/anaconda/ (accessed Jun. 13, 2022).
[19] J. Chan, R. Chung, and J. Huang, *Python API Development Fundamentals: Develop a full-stack web application with Python and Flask*. Packt Publishing Ltd, 2019.
[20] F. Pedregosa *et al.*, 'Scikit-learn: Machine Learning in Python', *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.
[21] J. Craighead, J. Burke, and R. Murphy, 'Using the Unity Game Engine to Develop SARGE: A Case Study', *Computer*, vol. 4552, Jan. 2007.
[22] G. Brockman *et al.*, 'OpenAI Gym'. arXiv, Jun. 05, 2016. Accessed: Jun. 03, 2022. [Online]. Available: http://arxiv.org/abs/1606.01540
[23] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition. O'Reilly Media, Inc., 2019.
[24] T. Jiang, J. L. Gradus, and A. J. Rosellini, 'Supervised Machine Learning: A Brief Primer', *Behavior Therapy*, vol. 51, no. 5, pp. 675–687, Sep. 2020, doi: 10.1016/j.beth.2020.05.002.
[25] B. Dk, 'Reinforcement Learning: An Introduction second edition', Accessed: Jun. 13, 2022. [Online]. Available: https://www.academia.edu/39631493/Reinforcement_Learning_An_Introduction_second_edition