**O imagine care conține text

Descriere generată automat** **O imagine care conține săgeată

Descriere generată automat****Universitatea „Politehnica” Timişoara**

**Facultatea de Electronică, Telecomunicaţii şi**

**Tehnologii Informaționale Timişoara**

**PROIECT DE DIPLOMĂ**

**Automatizarea decizii agenților într-un mediu virtual**

Coordonator:

**Sl. dr. Ing**. **Drăgulescu Bogdan**

Student:

**Uicoabă Alexandru**

**Timișoara**

**2022**

**Cuprins**

[**Capitolul 1 – Introducere** 4](#_Toc106049297)

[**1.1 Motivație** 4](#_Toc106049298)

[**1.2 – Obiective** 5](#_Toc106049299)

[**3.1 Sinteza lucrării** 5](#_Toc106049300)

[**Capitolul 2: Partea introductivă** 6](#_Toc106049301)

[**2.1 Agenți Sotfware** 6](#_Toc106049302)

[**2.2 Domenii de aplicabilitate** 7](#_Toc106049303)

[**2.3 Soluții software** 7](#_Toc106049304)

[**Capitolul 3: Învățare automată** 9](#_Toc106049305)

[**3.1 Machine Learning** 9](#_Toc106049306)

[**3.2.1 Unsupervised ML** 10](#_Toc106049307)

[**3.2.2. Supervised ML** 10](#_Toc106049308)

[**3.2.3 Reinforcement Learning** 10](#_Toc106049309)

[**3.3 Platforme software** 12](#_Toc106049310)

[**Capitolul 4: Proiectarea experimentului** 13](#_Toc106049311)

[**4.1 Obiectivele experimentului** 13](#_Toc106049312)

[**4.2 Arhitectura experimentului** 13](#_Toc106049313)

[**4.2.1 Circuitul din Unity** 13](#_Toc106049314)

[**4.2.2 Colectarea datelor în C#** 16](#_Toc106049315)

[**4.2.3 Scriptul în Python – Flask API** 17](#_Toc106049316)

[**4.2.4 Sistemele de decizie implementate** 17](#_Toc106049317)

[**4.2.4.1 Algoritmi supravegheați** 17](#_Toc106049318)

[**4.2.4.2 Reinforcement Learning** 19](#_Toc106049319)

[**4.2.6 Automatizarea mediului** 20](#_Toc106049320)

[**4.3 Comparație a sistemelor de decizie** 22](#_Toc106049321)

[**4.3.1 Machine Learning supravegheat vs Reinforcement Learning** 22](#_Toc106049322)

[**Capitolul 5: Concluzii** 23](#_Toc106049323)

[**5.1 – Contribuții proprii** 23](#_Toc106049324)

[**5.2 – Dezvoltări viitoare** 23](#_Toc106049325)

[**Capitolul 6: Bibliografie** 24](#_Toc106049326)

Sinteza

*Cap1 - Introducere*1.1 - Motivatie  
1.2 - Obiective

*Parte generală*Cap2 - Agenți software  
2.1 - Domenii de aplicabilitate  
2.2 - Soluții software

Cap3 - Învățare automată  
3.1 - Introducere în domeniu (ML clasic vs Rețele cu învățare profundă)  
3.2 - Arhitecturi specifice problemei abordate  
3.3 - Platforme sotfware

*Parte specială*

Cap4 - Proiectarea experimentului  
4.1 - Obiectivele experimentului  
4.2 - Arhitectura

* Circuitul din Unity
* Colectarea datelor din C#
* Scriptul în Python (API)
* Modelul de ML (random forrest & decission tree)
* Diagrama se face folosind limbajul mermaid
* Automatizarea începerii jocului și repornirea în momentul în care jocul este încheiat, fie că agentul a câștigat sau a pierdut

4.3 - Comparație sisteme de decizie

*5. Concluzii*  
5.1 - Contribuții   
5.2 - Dezvoltări viitoare

# **Sinteza**

# **Capitolul 1 – Introducere**

## **1.1 Motivație**

Scopul lucrării este de a îmbina programarea cu partea de gaming, și mai precis de a învăța un agent șă concureze într-un mediu virtual. În cazul de față agentul este representat de un pilot de kart iar mediul virtual este un circuit de karting. În acestă direcție se implementa două experimente în vederea învățării agentului cum să conducă folosind atât algoritmi de machine learning supravegheați (Decision Tree și Random Forrest) cât și partea 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 și a poziției gps. 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.

În vederea realizării experimentului s-au folosit mai multe intrumente 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 API și Machine Learning

## **1.2 – Obiective**

Prin această lucrare se dorește experimentarea modului în care un agent dintr-un mediu virtual poate să învețe singur cum să se comporte în acel mediu. 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 checkpointuri în cel mai scurt timp.

Pentru realizarea acestora avem mai multe obiective:

* 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 machine learning 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 domenile 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 access la informațiile din mediul virtual în care acesta se găsește.[1] 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 [2]:

* Managementul fluxurilor de lucru
* Managementul rețelelor
* Controlul traficului aerian
* Data minig
* Comerț electronic
* Educație
* Biblioteci digitale
* Baze de date inteligente
* Planificarea și gestionarea agendei
* Jocuri video

## **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** (MTCS) 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).[3]

Un alt articol în care s-a implementat un agent care să joace mai multe jocuri diferite este cel al lui Chiara F. Sironi, în care utilizând algoritmul **Self-adaptive Monte-Carlo Tree Search** (SA-MCTS) se ajunge la o versiune extinsă prin reglarea parametrilor on-line (online parameter tuning). Totuși, această versiune oferă o performanță ridicată doar în unele jocuri, unde se poate ajunge la un număr mai mare de simulări. [4]

Un alt joc în care s-a implementat un agent prin intermediul aceluiași algoritm MTCS este **AI Factory Spades**. Folosind acest algoritm se dorește controlarea agenților care nu sunt controlați de către utilizator (aliații și inamicii acestuia), care să emuleze comportamentul uman, păstrându-se nivelul de performanță și competiție. Principiile aplicate în acest joc pot fi transferate către alte jocuri care folosesc algoritmul MTCS.[5]

Pac-Man nu este singurul joc clasic în care s-a realizat învățarea automată a agentului. Zpepei Wei și echipa sa au obținut un agent autonom capabil să joace bine cunoscutul joc **Snake**. Antrenarea agentului s-a realizat prin folosirea de **Convolutional Neural Network** (CNN) cu o varietate de Q-learning. Această variantă are ca și scop evitarea situațiilor în care recompensele sunt rare și vin cu întârziere. Experimentele au arătat că prin intermediul acestui model, agentul a reușit chiar să depășească nivelul atins de oameni.[6]

Un alt tip de implementare este cea care folosește Markov Decision Process for Decision Making (MDP) . Folosind acest algoritm, pornind de la comportamenul albinelor, s-au create grupuri cooperative de agenți inteligenți pentru jocuri de tip Real-Time Strategy (RTS). Rezultatele au arătat un nivel ridicat de inteligență colectivă. Un lucru interesant a fost apariția temporară de așa numiți Team Leaders, care au condus cu succes echipa în bătălie.[7]

După parcurgerea articolelor de mai sus, s-a observat că într-un procent ridicat, au fost preferat algoritmul Monte Carlo Tree Search 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.

# **Capitolul 3: Învățare automată**

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

## **3.1 Machine Learning**

Machine learning este un domeniu de actualitate în zilele noastre, fie că vorbim de domenii precum Automotive, Cyber Security, Data Sience, detecții de imagini, etc. Din punct de vedere al definiților, am expus mai jos două dintre ele[8]:

O definiție mai generală a machine learning este următoarea[8]:

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

Există și o definiție orietată 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)

Clasificarea sistemelor de Machine Learning

Există mai multe clasificări a sistemelor de ML (Machine Learning) avem următoarele clasificări[8]:

1. Modul de antrenare a sistemelor:

* Sisteme supravegheate
* Sisteme nesupravegheate
* Sisteme semi- supravegheate
* Reinforcement Learning

1. Modul de învățare

* Online prediction
* Batch prediction

### **3.2.1 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.[9]

### **3.2.2. Supervised ML**

Metodele de învățare automată supraveghetă se utilizează pentru a descrie sarcini de predicție, deoarece aceastea 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 acuratețe bune este nevoie să “hrănim” modelul de ML cu un set mare de date în ciclul de învățare.[9]

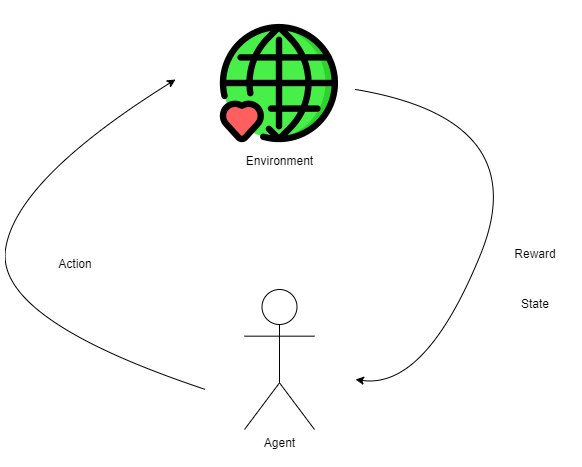
În această categorie de ML, datele sunt împărțite în mai multe seturi. Un prim set este cel de antrenare care sunt introduse în model. Al doilea set de date este cel de testare, prin intermediul căruia se determină performanța modelului, prin calcularea mai multor indicatori, cum ar fi accuratețea. De 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 situaților cu acțiuni, astfel încât să se maximizeze recompense care are au o valoare numerică. În mod essential, acestea sunt probleme în buclă închisă, deoarece acțiunile luate de sistemul de învățare influențează intrările ulterioare. [10]

Figura 3.1 Flux reinforcement learning



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. [10]

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 positive (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.

Mai jos sunt descrise două exemple de situații în care se poate folosii reinforcement learning[8]:

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 sreenshoturile jocului, realizâdu-se astfel o procesare de imagini, iar recompensele sunt reprezentate de scorul obținut în joc.

**3.3 Platforme software** Mai jos sunt desrise câteva platforme software care facilitează crearea mediului și antrenarea agenților.Anaconda

*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.[11]

Flask

*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.[12]

Scikit-learn

*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.[13]

Unity

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#.[14]

În vederea realizării experimentului, datorită faptului că dețin cunoștințe avansate de C# și Python, s-a ales dezvoltarea jocului în mediul Unity și antrenarea angentu folosind soluții Python care facilitează și ușurează acest lucru.

# **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 machine learning supravegheați 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.

### **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.

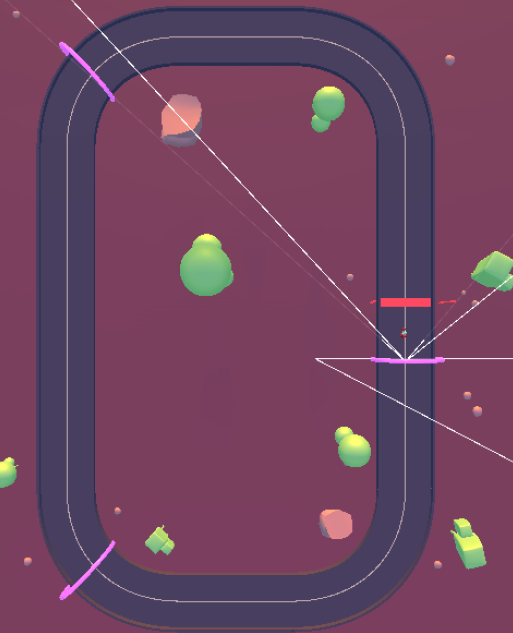
Primele elemente adăugate au fost cinci senzori, plasați în fața kartului, după cum se poate observa și în figura de mai jos. Aceștți senzori returnează distanța până la un obstacol care este în direcția lor. În cazul acestui experiment, valoarea maximă a senzorului a fost de 5.0f. În cazul în care nu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

Figura 4.1 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 de mai jos. 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 frame) și poziția din frame-ul trecut, în funcție și de zona în care se află kartul.

Figura 4.2 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 aibe valori între -1 și 1.

Valoarea axei respective se poate întelege în felul următor. Pe axa verticală, dacă nici una din cele două săgeți nu este apăsată atunci valoarea axei este 0. Dacă se apasă sageata *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 întămplă lucrurile și pe axa orizontală.

Figura 4.3 Imagina văzută de jucător

O imagine care conține text, captură de ecran, grafice vectoriale

Descriere generată automat

### **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.

Avem următoarele date de intrare (Unity -> Python):

* Timpul din Unity
* Poziția kart-ului pe axa X
* Poziția kart-ului pe axa Y
* Poziția kart-ului pe axa Z
* Senzorul din partea din stânga
* Senzorul din partea din stânga față
* Senzorul din partea din fată
* Senzorul din partea din dreapta față
* Senzorul din partea din dreapta
* Distanța senzorului din stânga
* Distanța senzorului din stânga față
* Distanța senzorului din față
* Distanța senzorului din dreapta față
* Distanța senzorului din dreapta
* Deplasarea în fată
* Zona

Avem următoarele date in ieșire(Python -> Unity):

* În față
* În stânga
* În dreapta
* În spate

### **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 environment care va conține pachetele necesare pentru implementarea soluției.O primă componentă a sciptului 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 raspunsul este dat de decizia luată de machine learning.

Acestă interfațăde tip POST este apelat de către joc la fiecare frame, din interiorul metodei *Update* din Unity. Body-ul acestei interfețeeste alcătuit din toate informațiile colectate din mediu, necesare ulterior pentru luarea deciziei. Răspunsul interfeței reprezintă decizia pe care Kartulo ia în pasul următor. Atât body-ul cât și răspunsul acestuiasunt sub formă de JSON.

Această componentă este folosită în ambele experimente, atât în cazul algoritmilor de machine learning nesupravegheați, 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 șcenarii, 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.

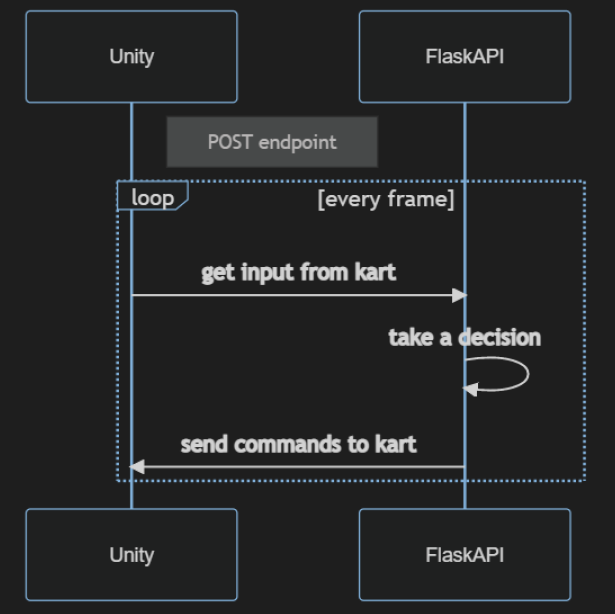
În exemplificarea experimentului, se vor folosii doi algorimti supravegheați, și anume Decision Tree și Random Forrest.

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 utlizatorul uman.

După ce s-a realizat cu success 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, la fiecare cadru din joc se trimit informațiile kartului înspre scriptul de FlaskAPI. 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 răspunsîn Unity, unde este mapată pe 4 valori booleane, 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.



#### **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 foldere având următoarea formă. 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 paremetri, cum ar fi de exemplu numărul de episoade.

O imagine care conține text

Descriere generată automat

Fluxul de date de la jocul din unity, până la scriptul în python de legătură, ajungând la scriptul de reinforcement learning se poate observa ân figura de mai jos.

O imagine care conține text, monitor, captură de ecran, negru

Descriere generată automat

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 recompe.

**Recompense și Penalizări**

Modul în care agentul învată 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 1.Aceste penalizări/recompense se dau pe fiecare senzor independent.

Tabelul 1. Recompensele în funție de distanța până la un obstacol

|  |  |
| --- | --- |
| Distanța până la obstacol | Recompensa |
| 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:

Reward = 100 – 50 + 10 +10 +10 +10

Reward = 40

### **4.2.6 Automatizarea mediului**

În vederea învățarii 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 foloseste interfața API */api/start-game* care este de tip *GET*. Prin intermediul acestuia se deschide executabilul exportat din Unity. După ce acesta se dechide, 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.

O imagine care conține LEGO, jucărie

Descriere generată automat

Interfața API /*api/end-game* de tip *GET* este folosit pentru oprirea jocului. Acest lucru se realizează prin funția 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 de mai jos se pot observa cele două interfețe, de pornire a jocului și de închidere a acestuia.

O imagine care conține text

Descriere generată automat

## **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 1. 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 1. Rezultatele obținute

|  |  |  |  |
| --- | --- | --- | --- |
|  | Jucător 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 |

### **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.

O primă contribuție proprie este adaptarea scenei din Unity prin adăugarea celor cinci senzori frontali pentru detecția obstacolelor. De asemenea, tot în mediul unity s-a mai create un script nou denumit *CollectData.cs*  prin intermediul căruia se realizează atât colectarea datelor din joc, cât și trimiterea comenzilor către kart programmatic. Tot aici se realizează și exportul datelor din joc într-un filșier *.csv* la fiecare frame și trimiterea datelor prin apelul interfeței API de tip POST pentru procesarea acestora.

Mai apoi, folosind limbajul Python și mediul anaconda, s-a create un script de FlaskAPI, care realizează atât comunicarea cu jocul din unity, cât și antrenarea modelului prin algoritmi de machine learning supravegheați. De asemenea, acest script poate fi considerat și un intermediar între scriptul care realizează partea de reinforcement learning și Unity.

Pentru reinforcement learning, s-a realizat structura proiectului, care este una necesară pentru un proiect care folosește tool kit-ul celor de la OpenAI gym. Pentru aceasta s-au create niște fișiere de configurate care au numele *\_\_init\_\_.py.* Tot aici, în fișierul *custom\_env.py* s-au create funcțile necesare pentru modelul de reinforcement learning.

## **5.2 – Dezvoltări viitoare**

În viitor, având arhitectura actuală, se poate continua pe mai multe direcții. Cea mai importană 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.

O primă direcție poate fi trecerea de la un mediu de tip single player, la un mediu de tip multi player. 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 umana de pe urma căreia să învețe agentul folosind tehnici de machine learning supravegheate.

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.

# **Capitolul 6: Bibliografie**

[1] 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.

[2] H. S. Nwana, ‘1Software Agents: An Overview’.

[3] 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.

[4] C. F. Sironi and M. H. M. Winands, ‘Analysis of Self-Adaptive Monte Carlo Tree Search in General Video Game Playing’, in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2018, pp. 1–4. doi: 10.1109/CIG.2018.8490402.

[5] 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.

[6] 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.

[7] 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.

[8] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition. O’Reilly Media, Inc., 2019.

[9] 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.

[10] 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

[11] ‘Anaconda Distribution — Anaconda documentation’. https://docs.anaconda.com/anaconda/ (accessed Jun. 13, 2022).

[12] 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.

[13] F. Pedregosa *et al.*, ‘Scikit-learn: Machine Learning in Python’, *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.

[14] J. Craighead, J. Burke, and R. Murphy, ‘Using the Unity Game Engine to Develop SARGE: A Case Study’, *Computer*, vol. 4552, Jan. 2007.