UNIVERSITATEA DIN BUCUREŞTI

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

SmartAI Race Simulator

Absolvent
Ilie George Ciprian

Coordonator ştiinţific
Conf.dr. Ciprian Păduraru

Bucureşti, septembrie 2022

Rezumat

Lucrarea își propune studierea librăriei MLAgents din Unity și a tehnicilor oferite de aceasta pentru machine learning în scopul încercarii de a da „train” unui AI cât mai bun, ce se va întrece într-o cursă cu un player uman.Lucrarea își propune și crearea unei aplicații aferente, a unui joc, în cadrul căruia se va întampla întrecerea dintre om și AI.

Librăria MLAgents permite instruirea unor AI ce vor avea un creier cu NN (neural network), creier ce va fi instruit cu metode precum Reinforcement Learning (RL), Genrative Adversial Imitation Learning (GAIL) sau Behavioral Cloning (BC), ce pot fi folosite individual sau simultan, cu diferiți parametrii.O comparație a eficiențelor acestora în funcție de combinarea lor între ele va fi efectuată ulterior în teză.

Ca și funcționalități de bază, aplicația va avea trei moduri de joc.Un mod în care jucătorul se va putea duela cu un agent AI, un mod în care doi jucători se vor putea întrece unul contra celălalt cu un ecran de tip „split screen” și un al treilea mod în care utilizatorul va putea urmări agentul antrenându-se.O funcționalitate adițională ar fi aceea de personalizare a setărilor graficilor destul de adaptabilă pentru orice tip de sistem, fie el performant sau nu.

În capitolul final, denumit „Concluzii” se vor prezenta pe lânga concluziile rezultate în urma realizării acestui proiect și câteva dintre problemele întâmpinate și rezolvarea acestora.

.

Abstract

The thesis proposes the study of the MLAgents library from Unity and of the various methods provided by this library with the purpose of training an agent which will challenge the player to a race. Another target of the paper is the creation of an appplication, a game where the player is going to compete against the AI.

The MLAgents librery allows the training of an AI with a neural network brain by providing machine learning methods such as Reinforcement Learning (RL), Generative Adversial Learning (GAIL) oe Behavioural Cloning (BC), methods which could be used simultaniously or individually, with a wide selection of paramethers.A comparation of the different efficiency of this methods will be later discussed in the thesis.

As the base features, the application will have three game modes.One mode will allow the player to race again an AI agent, the other one will facilitate the player with the option to race against another player in a split-screen type of mode and the last one will permit the user to watch the AI train.As aditional feature will be represented by the option of a wide variety of graphic settings.

In the final chapter, apart from the conclusions gathered, a few of the major problems and their solutions will be presented.

Cuprins

Cuprins

Capitolul 1

Introducere

1.1 Prezentare Generală

Lucrarea se încadrează în domeniul aplicațiilor de tip joc, mai precis a jocurilor ce au la bază elemente mai avansate de inteligență artificială, în special elemente de învățare automată, sau cum sunt cunoscute mai pe larg ca ML (Machine Learning).

Aplicația este un joc de curse care a fost dezvoltat folosind Unity Engine si limbajul C#.Pentru partea de creare a unui AI a fost utilizată librăria MLAgents din Unity, librărie dedicată pentru Machine Learning.Pe parcursul lucrării, pe lângă prezentarea jocului și al procesului de dezvoltare al acestuia se va prezenta și procesul de dezvoltare al Agentului (AI) din joc, precum și grafice și comparația metodelor de învățare aplicate acestuia.

Prima pagină a aplicației este reprezentată de un meniu, care conține butoane pentru modurile de joc „player vs player”, „player vs ai” și „watch ai”, un buton pentru tab-ul de opțiuni și unul pentru a închide aplicația.Un meniu de pauză este și el prezent din momentul intrării în modurile de joc, meniu care poate fi accesat prin apăsarea tastei „P” și care are funcții precum „restart”, „back to menu” și „resume”.

Pagina de opțiuni permite ajustarea graficilor aplicației în funcție de nevoile oricărui posibil utilizator, aceasta având features precum schimbarea rezoluției, schimbarea calității graficii, ajustarea sunetului precum si activarea modului de ecran complet sau a funcției de Vsync.

Fiecare mod de joc are scena sa, astfel modul „player vs player” permite jucarea jocului de către 2 persoane ce se vor putea întrece într-o cursă, modul „player vs AI” permite întrecerea cu un agent ce posedă inteligența artificială, iar modul „watch AI” permite observarea unui agent în timp ce se antrenează.

1.2 Obiectivele şi motivaţia alegerii temei

O temă care mi-a stârnit mereu curiozitatea a fost vastul domeniu al inteligenţei artificiale, un domeniu tânăr care este într-o creștere rapidă. Realizarea unui agent care poate imita comportamentul uman, şi poate dacă nu, chiar sa il întreacă, m-a intrigat dintotdeauna și mi s-a părut foarte interesantă integrarea acestei provocări în lucrarea de licență.

O altă motivație pentru această alegere a temei a fost interesul meu pentru jocuri şi modul de dezvoltare al acestora. Încă de la o vârstă fragedă am fost pasionat de modul de creare al acestora și mereu mi-am dorit să pot ajunge să dobândesc cunoștințele necesare în vederea realizării unei astfel de aplicații.

Obiectivul lucrării este acela de a crea o aplicație cât mai interesantă, atât din punct de vedere vizual cât şi din cel al caracteristicilor acesteia, studierea și prezentarea aspectelor de inteligență artificială oferite de Unity cât și încercarea de a realiza un AI cât mai performant.

1.3 Starea aplicaţiilor de inteligenţă artificială dezvoltate în Unity

Inteligența artificială în jocuri are ca scop complementarea calității acestora, fiecare joc având nevoie de un AI unic care sa îi satisfacă nevoile in scopul realizării obiectivului propus şi anume acela de a oferi o satisfacţie cât mai mare utilizatorului și de a produce o atmosferă cât mai distractivă şi amuzantă.

În jocurile dezvoltate până în prezent, precum „Assasin's Creed", „Hearthstone", „Pokemon Go", cât și multe altele, scopul elementelor de AI este să îmbunătățească calitatea, prin oferirea de provocări cât mai complexe cât și prin comportamentul realist al NPC-ilor în lumea jocului.

În concluzie, obiectivul nu este neapărat acela de a replica tot procesul de gândire al oamenilor ci acela de a face AI-ul să pară inteligent prin a îl învața cum să reacționeze în anumite situații, într-un mod în care are sens pentru lumea jocului.

Raționamentul de a nu dori crearea unor AI foarte complicați care încearcă să gândească precum un om, este legată de problema resurselor foarte mari necesare unui astfel de AI.

Pentru obținerea unui agent cât mai complex este nevoie de foarte multă putere de procesare, putere ce de multe ori nu este disponibilă intr-un joc, care are ca obiectiv secundar şi asigurarea accesării acestuia de către utilizatorii cu sisteme mai puțin performante. De regulă, accentul în jocuri, pică de cele mai multe ori pe design-ul de gameplay, elemente grafice si simularea fizică a fenomenelor, urmând ca numai resursele care rămân disponibile sa fie direcționate către algoritmul unui agent cu inteligenţă artificială.

Având în vedere aceste probleme prezentate mai sus, adevarata provocare a industriei de jocuri este optimizarea algoritmilor și al proceselor de învățare astfel încât acestea să utilizeze cât mai puține resurse.

1.4 Structura lucrării

Documentația tezei de licența va fi structurată in cinci capitole, fiecare dintre acestea prezentând aspectele relevante ale aplicației.

Primul capitol, denumit Introducere, descrie pe scurt aplicația, motivația care a contribuit la alegerea acestei teme, dar şi starea şi contextul elementelor de Inteligență Artificială ale jocurilor. La finalul acestui capitol se regăsește şi structura lucrării, explicată.

Al doilea capitol, denumit Tehnologii utilizate descrie câteva noțiuni de interes general despre materialele și tehnologiile folosite în dezvoltarea acestui proiect, în special pentru implementarea acestuia.

In cel de-al treilea capitol, Implementarea Aplicaţiei, sunt prezentați, în detaliu, pașii care au dus la crearea acesteia, procedeele folosite pentru realizarea agentului cu inteligenţă artificială, precum și câteva comparații între aceste metode.

Al patrulea capitol conține un scurt ghid de utilizare, ilustrat cu imagini explicative pentru fiecare feature al acesteia.

În ultimul capitol se află concluziile acestei lucrări, îmbunătățiri ulterioare, dar și probleme si dificultățile întâmpinate. De asemenea, bibliografia este prezentă şi ea în acest capitol.

Capitolul 2

Tehnologii Utilizate

2.1 Unity

Unity este o platformă de dezvoltare de jocuri creată de „Unity Technologies” care poate fi folosită pe majoritatea sistemelor de operare precum Windows / Linux / Mac OS. Unity este recunoscut pentru capacitățile sale de dezvoltare a jocurilor pe PC, dar şi pe Android/ IOS, iar mai nou acesta şi-a extins suportul și pentru dezvoltarea aplicațiilor de tip Virtual Reality.

Unity are mai multe funcționalități ce permit dezvoltarea de jocuri, funcționalitățile principale ce au fost utilizate în aplicația mea fiind cele relatate de dezvoltarea jocurilor in 3D. Relația între entitățile jocului poate fi realizată într-un mod inteligent folosind interfața interactivă (drag and drop), iar opțiunile pentru elementele de fizică se pot regăsi şi ele într-un tab separat.

În partea dreaptă putem observa elementele componente ale obiectului selectat, cu funcția „add component” se pot atașa componente acestuia, iar cu ajutorul funcției „drag and drop” putem adăuga scripturi.

În partea stângă superioară se află scena curentă cu obiecte aferente acesteia, în timp ce în partea de jos putem observa folderul proiectului prin care putem naviga cu ajutorul interfeței interactive.

În partea de sus se regăsește bara de opțiuni, în care avem elemente relevante pentru dezvoltarea proiectului, precum tab-ul de „project settings” în care putem modifica axele/ elemente de fizică pentru jocul nostru.

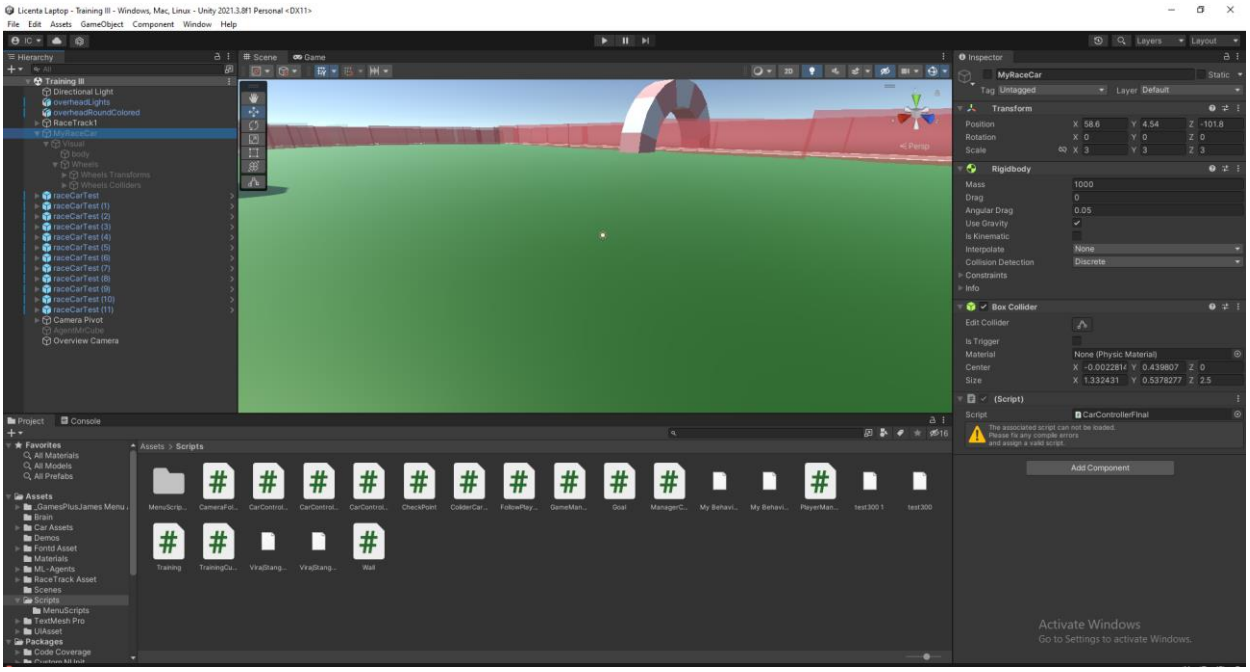


Figura 2.1 – Interfața Unity

Pentru partea de scripting şi scriere a codului, Unity permite utilizarea limbajului C# şi a mai multor librării specifice acestuia, câteva dintre ele fiind: UnityEngine, UnityEngine, SceneManagement, Unity.MLAgents etc, pentru utilizarea lor fiind necesară sintaxa „using ...” la începutul scriptului.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;
```

Figura 2.2 – Exemplu de uitilizare librării

2.2 Visual Studio

Ca şi IDE folosit pentru scrierea codului în C# am utilizat Visual Studio, editorul de cod recomandat de Unity. Visual Studio este un IDE dezvoltat de către cei de la Microsoft, fiind frecvent utilizat pentru dezvoltarea de jocuri video, aplicații web sau chiar si aplicații mobile.

Un feature important denumit „IntelliSense” permite completarea codului oferind posibile

opțiuni în funcție de bucata de cuvânt scrisă, astfel oferind eficiență în timpul alocat scrierii codului și în reducerea numărului de greșeli de tip typo.

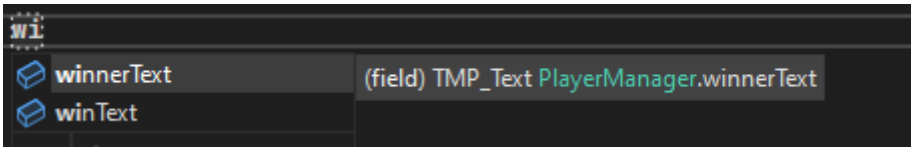


Figura 2.3 – IntelliSense

Un alt avantaj oferit de acest editor de cod este reprezentat de existența posibilității de a redenumi toate aparițiile unui cuvânt cu ajutorul funcției de code refactoring:

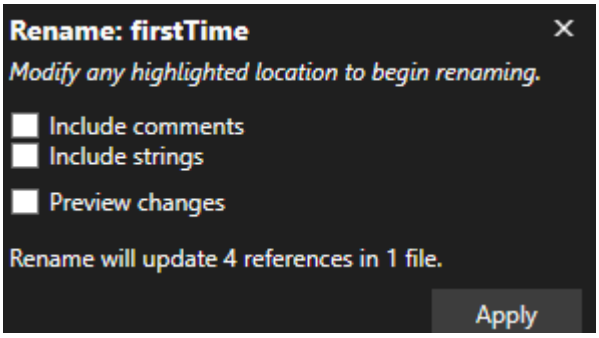


Figura 2.4 – Code refactoring

2.3 Utilizarea C# pentru scripting

Pentru crearea scripturilor obiectelor, Unity permite utilizarea limbajului C# în acest scop. Acest limbaj de programare este proiectat pentru a fi modern, simplu dar și bun pentru programarea orientată pe obiecte.

Un alt avantaj al acestuia este asemănarea cu celelalte limbaje C, adică asemănarea cu C si C++, astfel învățarea lui nefiind dificilă pentru programatorii ce sunt familiarizați deja cu limbajele precizate mai sus.

În cadrul dezvoltării de jocuri in Unity, C# este folosit mai mult pentru descrierea elementelor de gameplay, scripturile fiind deja predefinite cu o funcție „ void start()” care este apelată o singură dată la încărcarea scenei și o funcție void Update() / void FixedUpdate() care este apelată la fiecare frame.

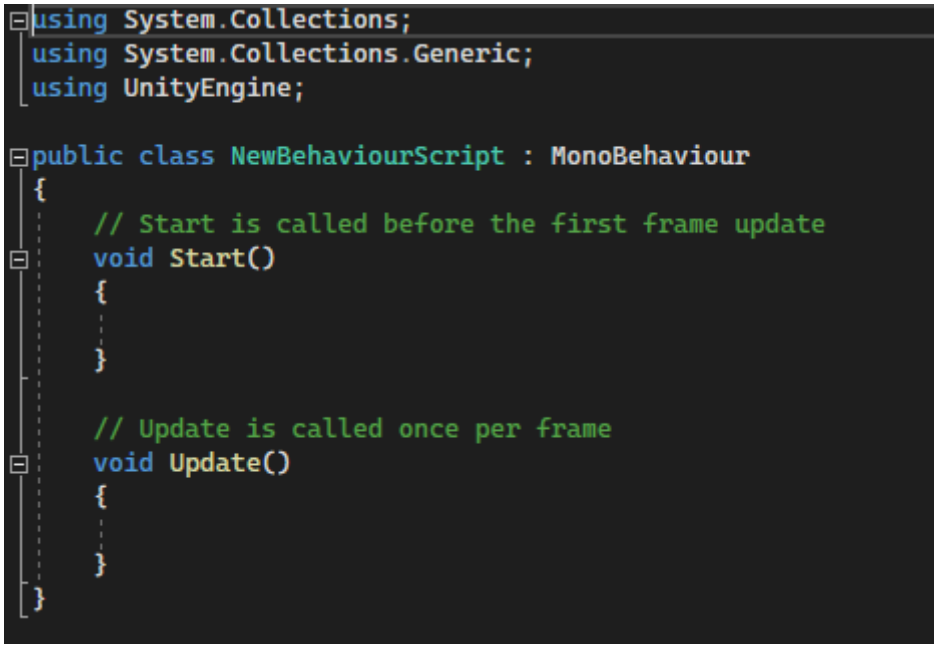


Figura 2.5 - exemplu script prestabilit în C# pentru Unity

2.4 Librăria MLAgents

Librăria MLAgents este o librărie ce permite antrenarea agenților folosind machine learning. Aceasta permite transformarea anumitor scene din joc în medii de antrenare a agenților precum și accesul la diverși algoritmi precum RL, GAIL si BC, implementări bazate din Pytorch.

2.4.1 Instalare

Deoarece MLAgents nu este o librărie inclusă în mod normal în Unity, aceasta trebuie instalată separat. Cea mai mare problemă în acest sens este găsirea unor versiuni compatibile între ele, ale acestor pachete. Următorul set de pachete a fost cel care a permis rularea librăriei pentru mine, după nenumărate încercări şi erori de instalare.

2.4.1.1 Instalare Python

Pentru a reuși sa rulăm librăria, mai întâi trebuie instalată o versiune de Python 3.6.1 sau mai recentă .Python poate fi descărcat de pe https://www.python.org/ apoi rulat si instalat. Verificarea instalării sale se poate face rulând următoarea comandă in cmd:

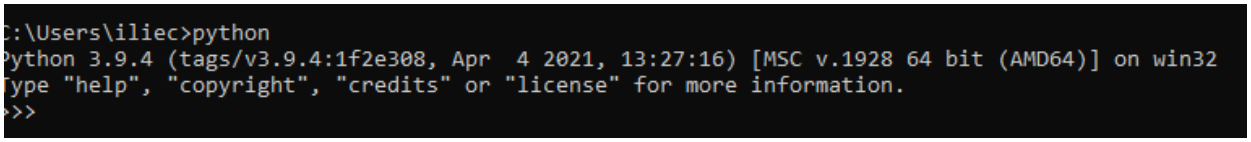


Figura 2.6 – comanda pentru verificarea instalarii Python

2.4.1.2 Crearea unui environment virtual

Crearea unui python virtual environment în interiorul folderului proiectului prin executarea comenzii următoare în cmd, în folderul proiectului:

```
python -m venv venv
```

Figura 2.7 – rularea comenzii pentru creare

Pentru a activa environmentul vom utiliza comanda: cd venv/ Scripts/ activate

```
C:\Users\Ciprian\Licenta v2\venv>cd Scripts

C:\Users\Ciprian\Licenta v2\venv\Scripts>activate

(venv) C:\Users\Ciprian\Licenta v2\venv\Scripts>
```

Figura 2.8 – comanda rulată

2.4.1.3 Instalarea Pytorch

Pytorch este un framework pentru machine learning care ajută la accelerarea procesului de la prototip la produsul final. Aceasta la rândul său are la baza librăria Torch din Python. Pentru instalare se rulează comanda.

pip install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html

2.4.1.4 Instalarea mlagents din cmd

Pentru instalare se ruleaza comanda:

python -m pip install mlagents==0.28.0

2.4.1.5 Instalarea pachetului din Unity

După multe încercări si ore de debugging / căutat soluţii pe forumuri, singurul pachet care a reușit să îmi meargă este cel în versiune experimentală, ultimul pachet released, adică v2.2.1-exp1. Ca o precizare ultima versiune stabilă este cea de acum 4-5 ani, versiunea 1.0.7 din care lipsesc foarte multe features.

Pentru instalarea pachetului din Unity mergem in Window > Pachet Manager > Unity Registry

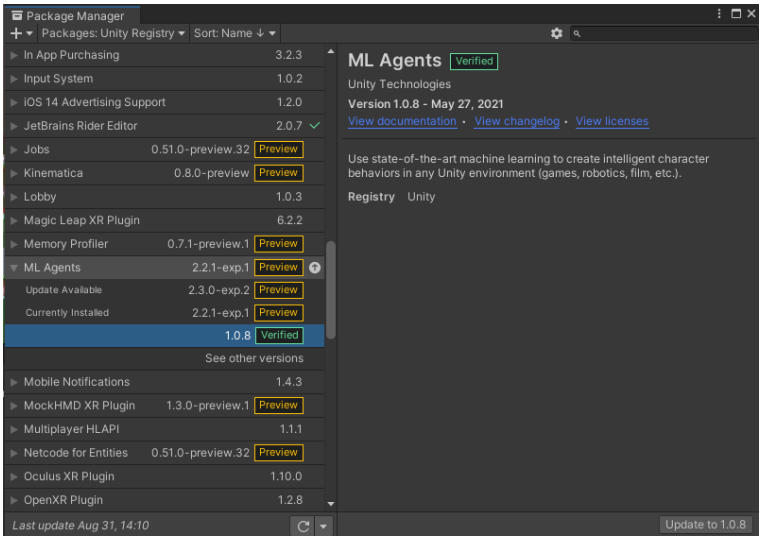


Figura 2.9 – Fereastra pentru Pachet Manager

Vom da enable la pachetele experimentale, apoi vom apăsa butonul de install:

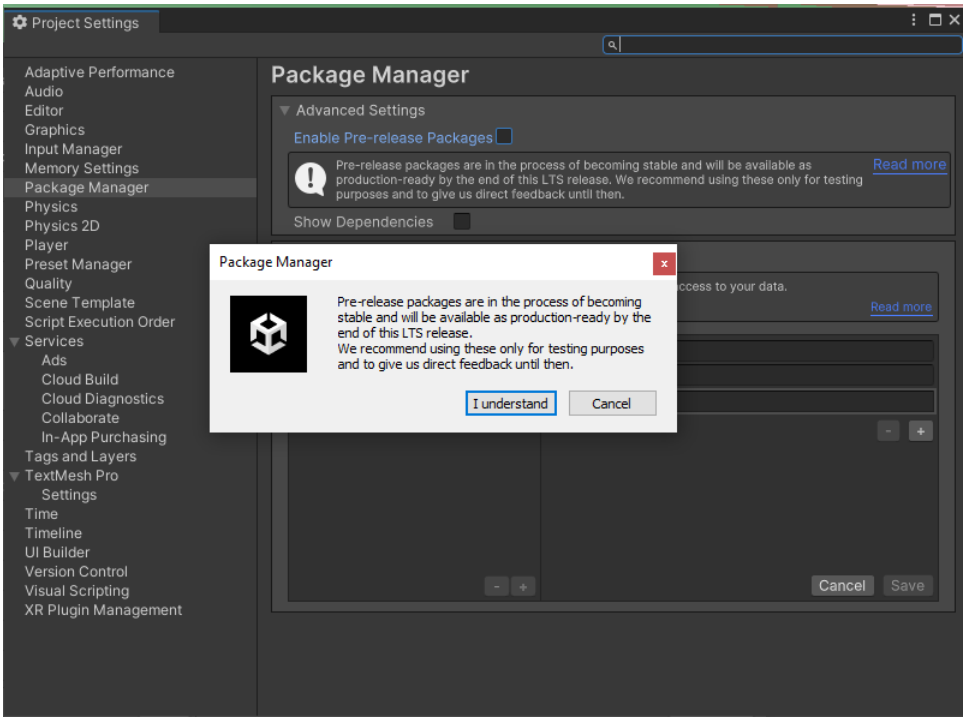


Figura 2.10 – Fereastra pentru activarea pachetelor experimentale

2.4.2 Algoritmi

Folosind anumite principii de bază, este posibil să antrenam anumiți agenți utilizând mai multe metode. Ca idee de bază agentul adună informații denumite „Observații" cu ajutorul senzorilor. În exemplul nostru, agentul colectează datele referitoare la distanța până la perete/checkpoint ca repere, apoi ia acțiuni corespunzătoare folosind acțiuni discrete (mișcare fată/spate sau viraj stânga/dreapta). Cu ajutorul funcțiilor de recompense, agentul știe ca a făcut o acțiune corectă precum mersul printr-un checkpoint datorită recompensei pe care o primește, respectiv acesta învață ca a făcut o acțiune greșită dacă primește o recompensă negativă (penalizare) în urma ciocnirii cu un obstacol.

Scopul agentului este sa găsească un comportament care îi maximizează punctajul prin intermediul numărului de recompense primite.

Principalii algoritmi din librărie sunt RL, GAIL si BC.

Cu ajutorul RL/ Reinforcement Learning agentul încearcă diferite acțiuni aleatoare pentru a observa care din ele îl duc la recompense cat mai mari, apoi începe sa învețe pe baza experiențelor acestora.

GAIL sau Generative Adversial Imitation Learning, folosește o abordare diferită faţă de RL. Agentul va observa o demonstrație oferită în prealabil, apoi o a doua rețea neuronală numită „the discriminator" este învățată sa își dea seama dacă o acțiune provine dintr-o demonstrație sau de la agent. Discriminatorul poate în acest mod să examineze o nouă acțiune primită și să o recompenseze în funcție de cât de aproape crede că a fost aceasta faţă de demonstrație. In acest mod cu cat agentul devine mai bun sa mimeze acțiunile din demonstrație și discriminatorul devine din ce în ce mai strict cu demonstrațiile.

BC sau Behavioural Cloning este tot o metodă de învățare prin imitare, în care agentul învață să imite exact acțiunile din demonstrație. Dezavantajul faţă de GAIL este că agentul nu va putea niciodată să devină mai bun decât demonstrația primită.

2.5 Asseturi folosite

Pentru realizarea părții grafice am utilizat mai multe asset-uri modulare. Pentru a importa un asset in Unity, acesta trebuie descărcat şi apoi tras pe editor, in folderul „Assets", după care trebuie urmate instrucțiunile din fereastră.

Formatele de fișiere recomandate pentru Unity sunt FBX si DAE, dar sunt acceptate si OBJ.

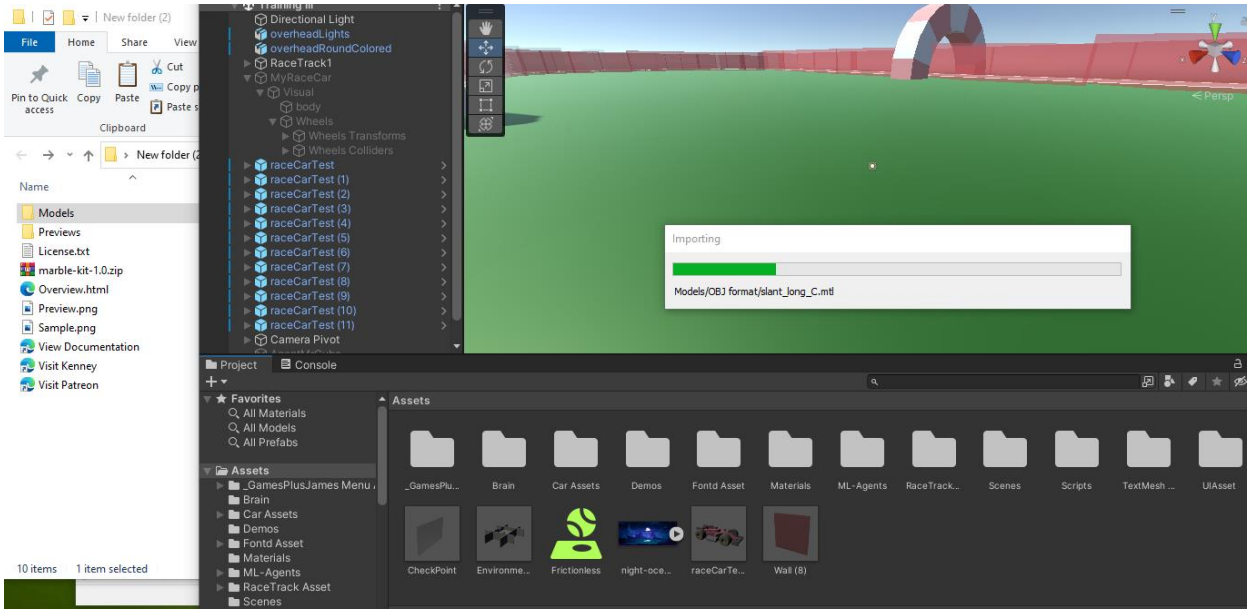


Figura 2.11 – Importare asset

Arena si mediile de antrenare sunt create cu ajutorul imaginației proprii. Asset-ul din care s-a creat stadionul / arena conține bucăți de pistă și de decor modulare, care au fost așezate într-o scenă cu sens, de către mine.

Un link către site-ul de unde au fost descărcate asset-urile se poate găsi în bibliografie.

Figura 2.12 – Imaginile din asset-ul pentru pista de curse

Capitolul 3

Implementarea aplicației

3.1 Aplicația

Ca şi structură, componentele aplicației sunt așezate în foldere cu titluri sugestive, astfel încât asseturile se află în folderele ce au această denumire în nume, folderul Materials conține materialele (culorile) utilizate, folderul Scenes conține scenele, folderul Scripts conține scripturile atașate obiectelor, iar folderele MLAgents si TextMashPro conțin librăriile aferente acestora.

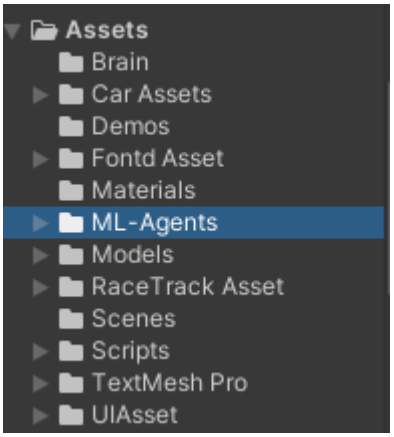


Figura 3.1 – Structura folderelor aplicației

3.1.1 Meniu

Prima pagină a aplicației este formată din meniul principal. La deschiderea jocului, pe un fundal sonor, apare meniul principal constând în 5 butoane cu rol specific. Primele 3 butoane conduc utilizatorul în scena aferentă fiecărui mod de joc, butonul Options conduce utilizatorul către pagina special dedicată de opțiuni, iar ultimul buton, cel de Quit, închide aplicația.

Ca și structură, meniul se află într-un canvas denumit „Main Menu”. In Game Object Buttons se află butoanele, împreună cu ecranul de setări aflat în obiectul cu numele „Options Menu Background”. Imaginea de fundal este atribuită pe un obiect de tip Image ce acoperă canvasul.

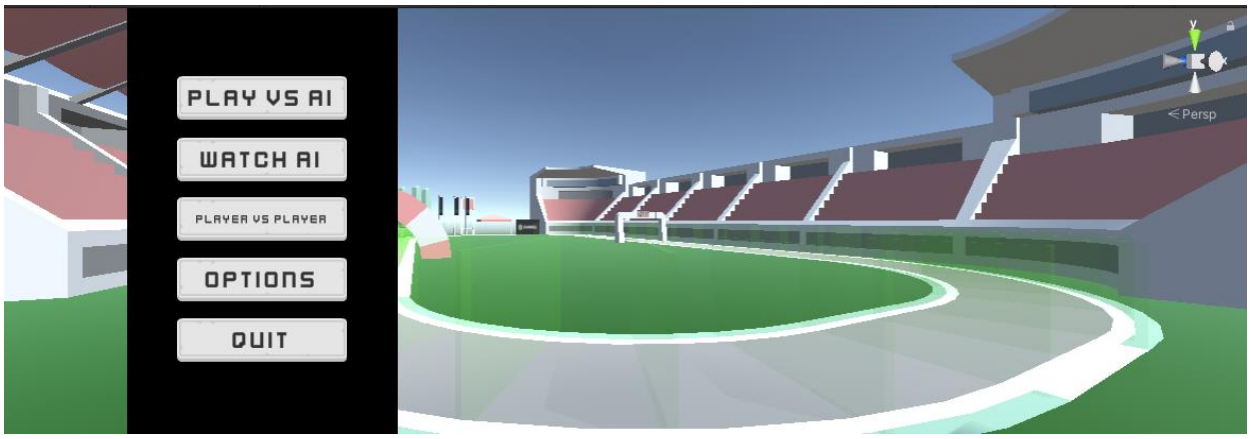


Figura 3.2 – Meniul aplicației

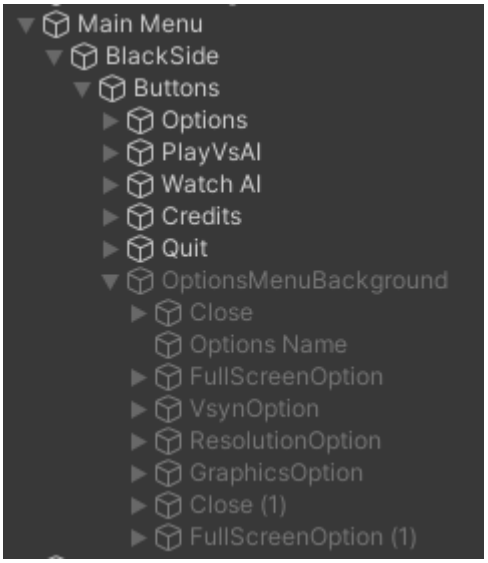


Figura 3.3 – Ierarhia elementelor meniului

3.1.1.1 Creare butoane si functionalitaţi

Pentru crearea unui buton, facem click dreapta în interiorul listei în dreptul canvasului apoi UI > Button.

O dată ce a fost creat obiectul îl putem personaliza cu o textură şi un text ales de noi. Fiecare buton are un text specific atașat, iar în câmpurile : „Source Image” – Image şi „Target Graphic, Highlighter Graphic, Highlighter Sprite” au fost încărcate texturi corespunzătoare.

Pentru fiecare buton vom selecta funcția specifică și o vom încărca acestuia pentru On Click(). (Vezi figura 3.4)

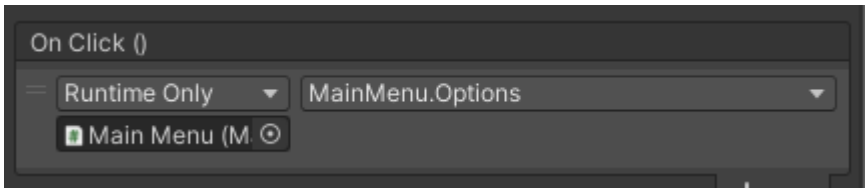


Figura 3.4 – Incarcarea functie pe On Click()

Obiectul „Main Menu” are atașat scriptul „Main Menu” în care se regăsește implementarea funcțiilor butoanelor.

Pentru butonul „Play vs AI” avem scriptul:

```
public void PlayVsAI()
    {
        SceneManager.LoadScene("GameMain");
        //incarcam scena PlayVSAI
    }
```

Pentru „Watch AI”:

```
public void WatchAI()
    {
        SceneManager.LoadScene("Training III");
        //incarcam scena de urmarit AI
    }
```

Pentru “Player vs Player”

```
public void Credits()
    {
        SceneManager.LoadScene("1vs1");
        //incarcam scena jocului payer vs player
    }
```

Pentru Options si Quit:

```
public void Options()
    {
        optionsBgScreen.SetActive(true);
        //setam vizibilitatea meniului de optiuni true
    }
    public void QuitGame()
    {
        Application.Quit();
        print("Game was quit");
        // inchidem jocul
    }
```

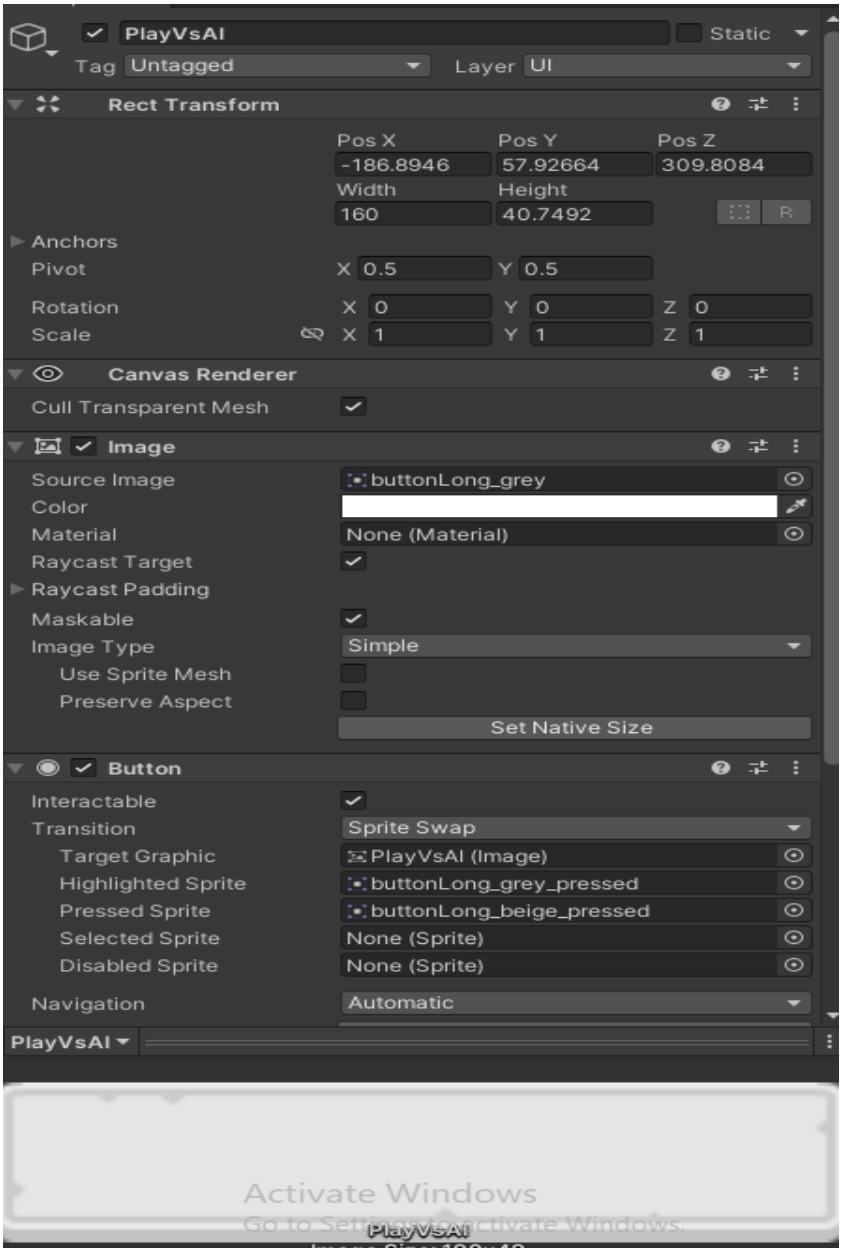


Figura 3.5 – Componentele butonului

3.1.1.2 Ecranul de setări

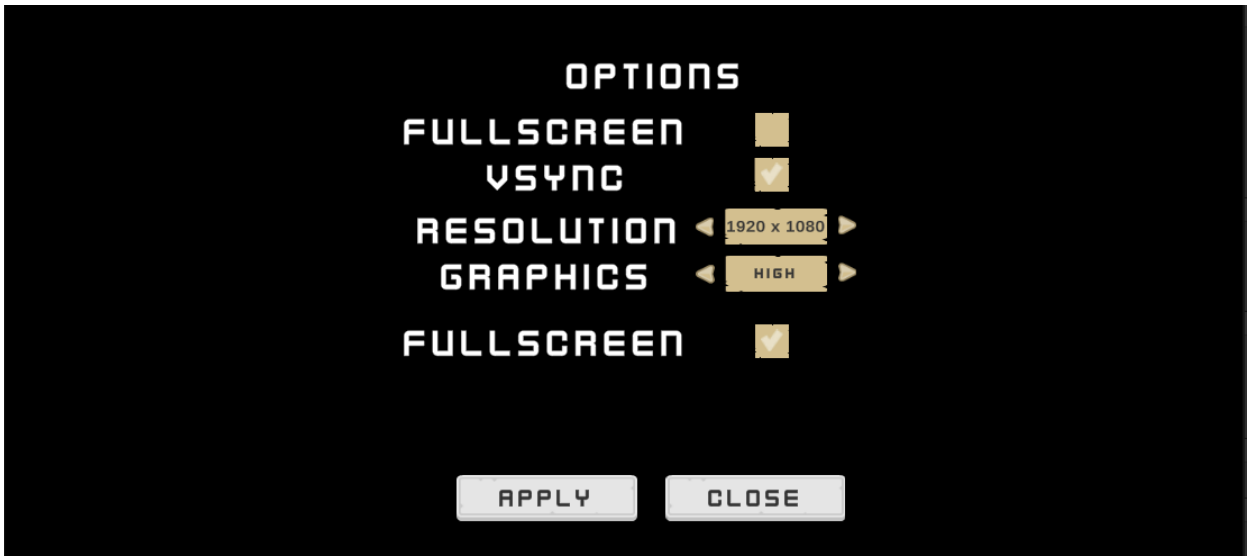


Figura 3.6 – Meniul de setari

Elementele acestui ecran sunt copii ale obiectului Options Menu Background, care este un obiect de tip imagine. În momentul activării meniului obiectul este activat, astfel devenind vizibil, iar odată cu apăsarea butonului „close” acesta dispare.

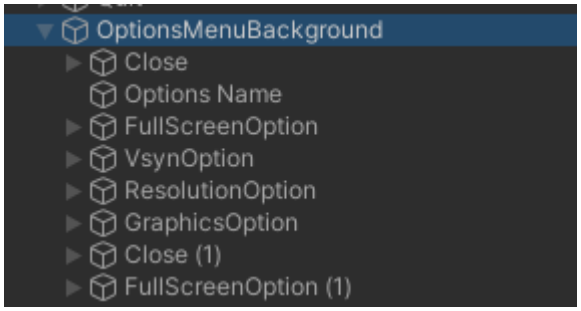


Figura 3.7 – Ierarhia obiectelor pentru ecranul de setari

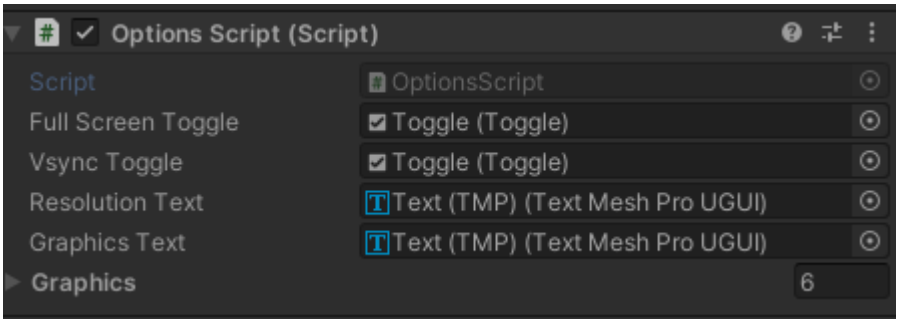


Figura 3.8 – Scriptul pentru meniul de opțiuni

Scriptul ce execută acțiunile acestui meniu este atașat obiectului părinte „Options MenuBackground „. Toate setările vor fi aplicate la apăsarea butonului de „Apply”.

Pentru setarea opțiunii de fullScreen avem următoarea bucată de cod, care verifica dacă toggle-ul este bifat sau nu, iar în caz ca acesta este bifat, se setează ecranul in fullScreen.

```
fullScreenToggle.isOn = Screen.fullScreen;
```

Pentru opțiunea de **Vsync**, mai întâi la încărcarea meniului verificăm daca acesta este pornit sau nu, iar dacă este pornit o să bifăm automat:

```
if ( QualitySettings.vSyncCount == 0)
        {
            vsyncToggle.isOn = false;
        }
        else
        {
            vsyncToggle.isOn = true;
        }
```

Pentru a seta rezoluțiile am creat o clasă resolution în care ținem caracteristicile de **width** si **height**:

```
public class ResolutionsItem
    {
        public int firstPart, secondPart;

        public ResolutionsItem(int x, int y)
        {
            this.firstPart = x;
            this.secondPart = y;
        }
    }
```

In void Start() adăugam cele 3 rezoluții in lista de rezoluții:

```
public List<ResolutionsItem> resolutions = new
List<ResolutionsItem>();
resolutions.Add(new ResolutionsItem(1920, 1080));
resolutions.Add(new ResolutionsItem(1280, 720));
resolutions.Add(new ResolutionsItem(854, 480));
```

De asemenea verific și ce rezoluție este pe ecran si updatez interfața intr-un mod corespunzător:

```
for (int contor = 0; contor< resolutions.Count; contor++)
        {
            if( Screen.width == resolutions[contor].firstPart &&
Screen.height == resolutions[contor].secondPart)
            {
                i = contor;
                UpdateTextResolution();
            }
        }
```

În momentul în care butoanele de stânga/dreapta sunt apăsate se apelează funcțiile specifice care updateaza interfața si rețin modificările.

```
public void UpdateTextResolution()
    {
        resolutionText.text = resolutions[i].firstPart.ToString()
+ " x " + resolutions[i].secondPart.ToString();
        //dam update la textul de grafica
    }
    public void LeftArrowResolution()
    {
        if (i >= 1)
            i--; // scad contorul

        UpdateTextResolution(); //updatez
    }

    public void RightArrowResolution()
    {
        if (i <= 1)
            i++;

        UpdateTextResolution();
    }
```

Pentru a seta grafica, procesul se întâmpla asemănător celui de rezoluție. Avem un array în care ținem minte grafica, iar procesul de modificare a acesteia este asemănător, existând şi aici 2 butoane ce ne ajută să parcurgem posibilele opțiuni de grafică. Singura diferență este reprezentată de faptul ca opțiunea default este aceea de High.

```
public string[] Graphics = new string[10];
private int graphicsContor = 3;

//in start()

        Graphics[0] = "Very Low";
         Graphics[1] = "Low";
         Graphics[2] = "Medium";
         Graphics[3] = "High";
         Graphics[4] = "Very High";
         Graphics[5] = "Ultra";

         // Graphics[5] = "Ultra";
         GraphicsSettings();
         QualitySettings.SetQualityLevel(graphicsContor);
```

// funcții

```
public void RightArrowGraphics()
    {
        if (graphicsContor >= 1)
            graphicsContor--;

        GraphicsSettings();
    }

    public void GraphicsSettings()
    {
        graphicsText.text = Graphics[graphicsContor];
       // QualitySettings.SetQualityLevel(graphicsContor);
        print(Graphics[graphicsContor]);
    }

    public void LeftArrowGraphics()
    {
        if (graphicsContor < 5)
            graphicsContor++;

        GraphicsSettings();
    }
```

La apăsarea butonului de apply, se apelează următoarea funcție care aplică modificările:

```
public void ApplyChanges()
    {
        //Screen.fullScreen = fullScreenToggle.isOn;
        print(fullScreenToggle.isOn);

        //daca vSync toggle este bifat, dam update la vSync
        if( vsyncToggle.isOn)
        {
            QualitySettings.vSyncCount = 1;
        }
        else
```

```
        {
            QualitySettings.vSyncCount = 0;
        }

        //update la rezolutie cu optiunea selectata
        Screen.SetResolution(resolutions[i].firstPart,
resolutions[i].secondPart, fullScreenToggle.isOn);
        //update la setarile de calitate a graficii
        QualitySettings.SetQualityLevel(graphicsContor);
    }
```

3.1.2 Car Controller

Deoarece mișcarea mașinii reprezintă o componentă de bază a aplicației, am ales să o prezint într-un subcapitol separat. A fost simulată o mașina cu tracțiune pe față, realizarea acestei caracteristici fiind simulată prin tracțiune pe fiecare roată din față independent, acestea fiind de asemenea singurele ce pot vira la stânga sau la dreapta, într-un mod cat mai realist.

Ca și ierarhie de componente, mașina are un „Rigidbody” si „Box Collider”, apoi pe fiecare roată este setat câte un „Wheel Collider”.Mașina are atașat scriptul „Car controller Final” cu referințele obiectelor atașate.

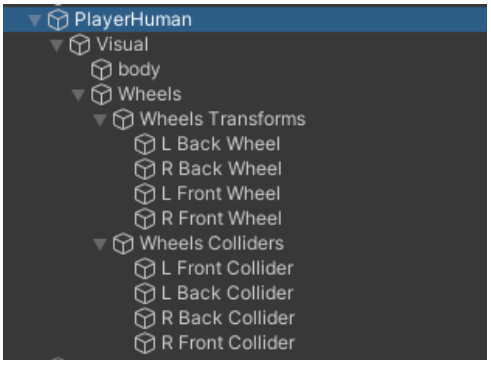


Figura 3.9 – Ierarhia componentelor masinii

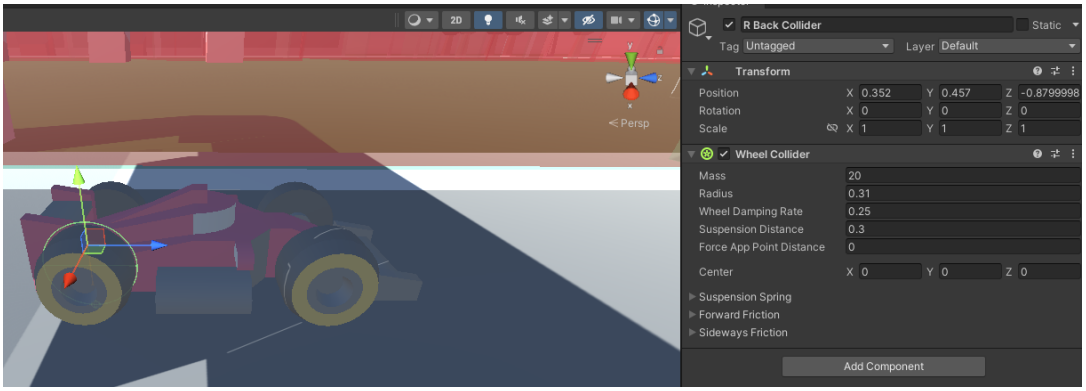


Figura 3.10 – Setarea componentelor „wheel collider”

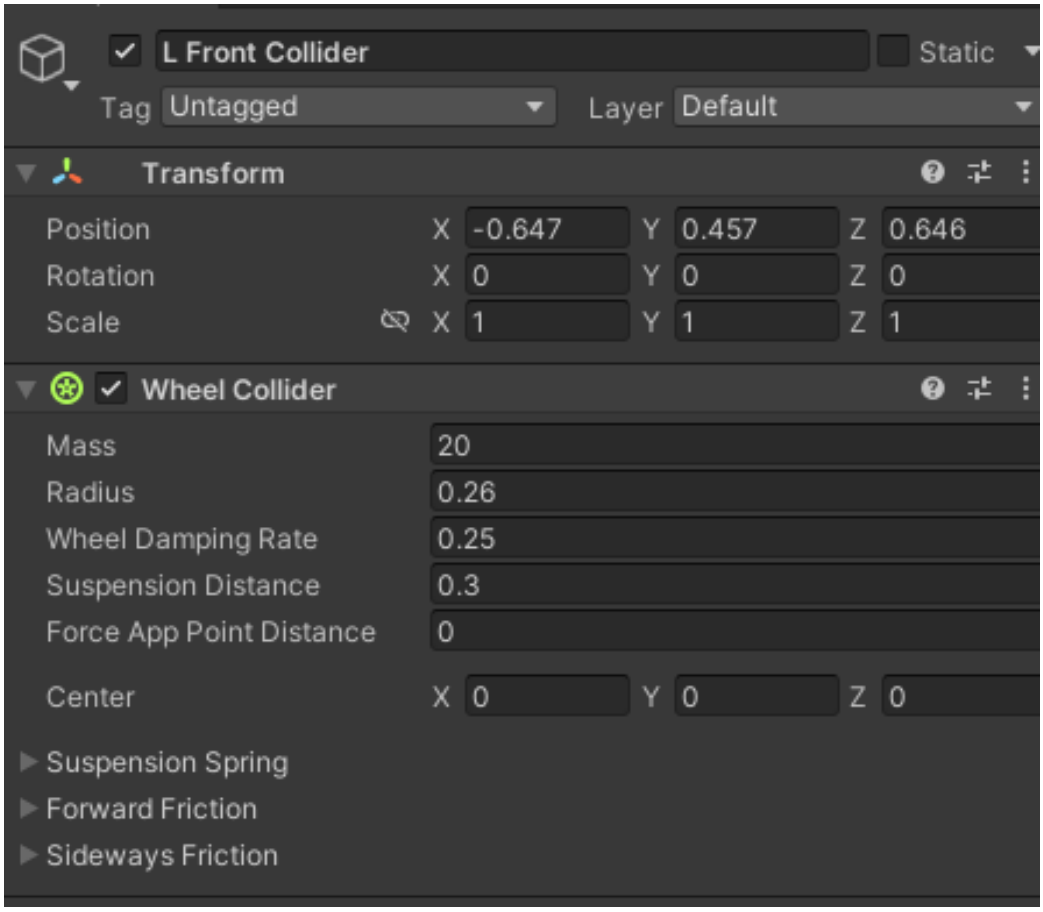
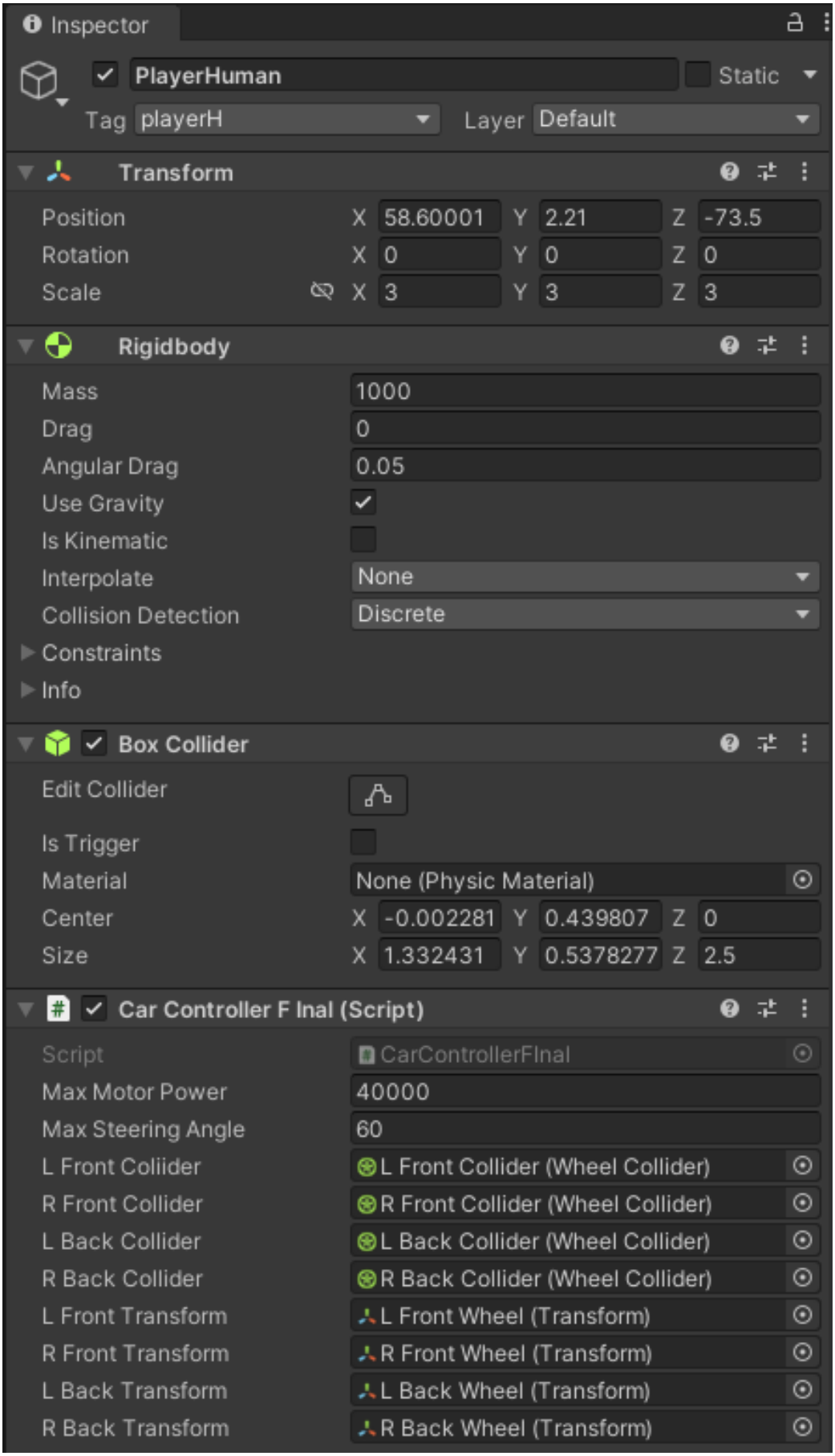


Figura 3.11 – Componentele roții mașinii

Figura 3.12 – componenta de „Box Collider”



Fiura 3.12 – Componentele mașinii

Ca și variabile avem legate referințe catre „wheel colliders” și „wheel transforms”, iar maxMotorPower reprezintă puterea maximă a motorului, în timp ce maxSteeringAngle reprezintă unghiul maxim de viraj al roților.

```
[SerializeField] private float maxMotorPower;
    [SerializeField] private float maxSteeringAngle;

    //wheen colliders
    [SerializeField] private WheelCollider LFrontColiider;
    [SerializeField] private WheelCollider RFrontCollider;
    [SerializeField] private WheelCollider LBackCollider;
    [SerializeField] private WheelCollider RBackCollider;

    //wheel transforms
    [SerializeField] private Transform LFrontTransform;
    [SerializeField] private Transform RFrontTransform;
    [SerializeField] private Transform LBackTransform;
    [SerializeField] private Transform RBackTransform;
```

In void start() inițializam componenta de rigid body a mașinii în variabila rigidBodyCar.

```
Rigidbody rigidBodyCar;
    void Start()
    {
        rigidBodyCar = GetComponent<Rigidbody>();
    }
```

In funcția void Update() apelăm funcția Move() care se ocupă de mișcare și funcția UpdateVisuals() ce se ocupa de grafica de rotire a rotilor.

În functia de Move() adăugăm componentelor de wheel colliders putere,iar direcția este oferită de: Input.GetAxis(“Vertical”) care returnează un input între -1 si 1, în funcție de tasta apăsată.Variabila “aditionalMotorPower” oferă putere suplimentară motorului prin adăugarea unei forțe la “Rrigidbody”.Pentru viraj “currentTurnAngle” primește maximul unghiului de viraj inmultit cu o variabila între -1 si 1 generată de Input.GetAxis(“Horizontal”).

```
//Movement car
        LFrontColiider.motorTorque = Input.GetAxis("Vertical") *
maxMotorPower * 10;
        RFrontCollider.motorTorque = Input.GetAxis("Vertical") *
maxMotorPower * 10;
        float aditionalMotorPower = Input.GetAxis("Vertical") *
2000;
        rigidBodyCar.AddForce(transform.forward *
aditionalMotorPower);
```

```
        //Steering
        float currentTurnAngle = maxSteeringAngle *
Input.GetAxis("Horizontal");
        LFrontColiider.steerAngle = currentTurnAngle;
        RFrontCollider.steerAngle = currentTurnAngle;
```

Funcția Update Visuals() updatează grafica roților, aceasta este preluată din documentația de la Unity pentru crearea unui „car controller”, exista link către această pagină în bibliografie.

3.1.3 Scena „Player vs AI”

În această scenă se desfașoară cursa dintre jucator si AI, cursă care are 3 ture.Jucătorul uman are o interfață ce îi permite să vadă numărul de curse rămase, timpul de când a început cursa și cel mai bun timp în care a reușit să termine o tură.

Ca și ierarhie de obiecte, în scenă se află 5 obiecte importante: Stadionul (alcătuit din mai multe biecte), mașina jucătorului, mașina agentului, un obiect denumit “camera pivot” care se ocupă de deplasarea camerei și un canvas pentru interfața jucătorului.Obiectul “camera pivot” conține un script care modifică poziția camerei în funcție de poziția și direcția jucătorului.Scripturile și logica agentului vor fi discutate în capitolul următor.

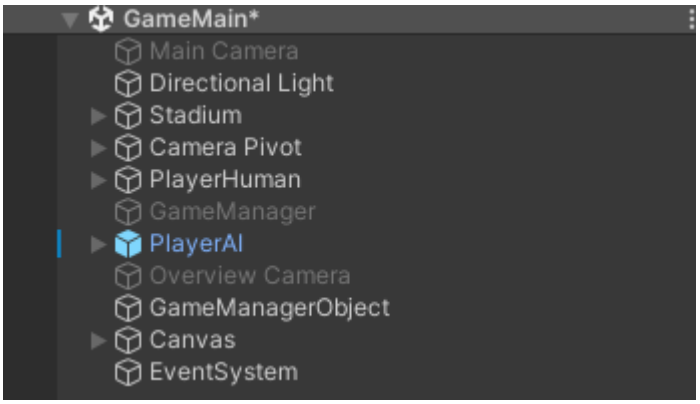


Figura 3.13 – Ierarhia obiectelor scenei

Pista este înconjurată cu un rând de pereți invizibili, iar pe pistă se află checkpointuri orientative.

Interfața pe care o vede jucătorul este creată pe un obiect de tip „canvas”.În acest obiect se află un „TextMeshPro” pentru cronometru, textul pentru cel mai bun timp în care s-a parcurs o cursă, textul pentru numărul turei curente, dar și un text care devine vizibil la finalul cursei.

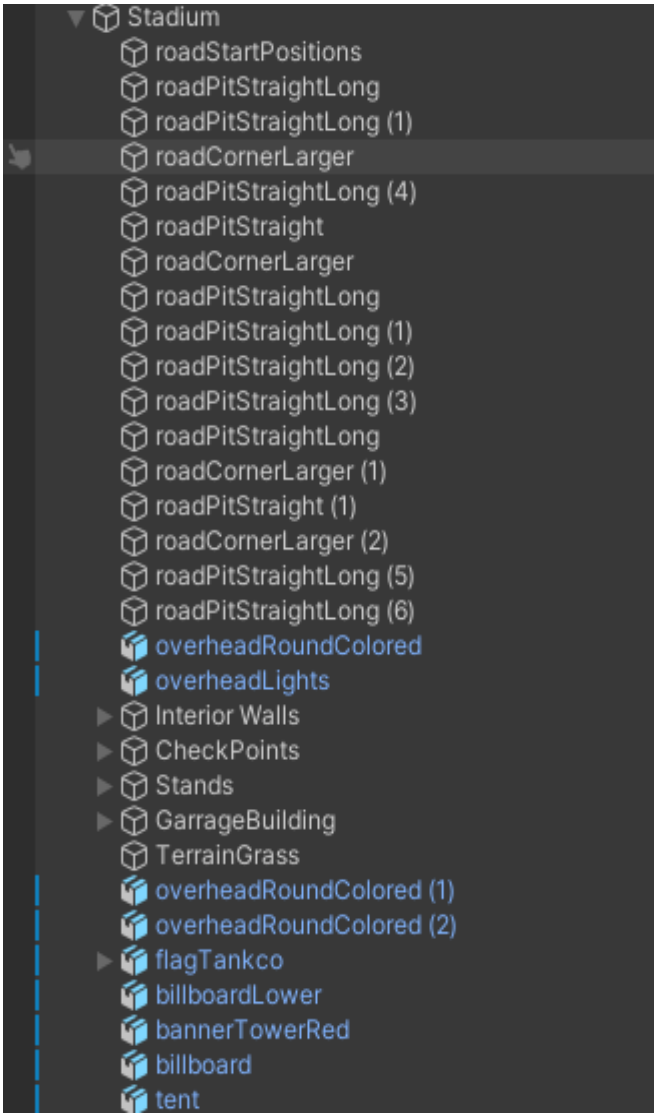


Figura 3.14 – Ierarhia obiectelor din care este alcătuit stadionul

Figura 3.15 – Interfața jucătorului

Pentru logica jocului avem scriptul “Player Manager”, script atasat jucatorului.

In funcția start() ne facem inițializările necesare logicii jocului:

```
 void Start()
{
    lap = 0; // numarul de ture
    //Time.timeScale = 0;
    startTime = 0; // start time e 0
    tempMax = 0;
    bool firstTime = false; // variabila ajutatoare
    tempMax = 1000000; // best lap time = 0


     }
```

In funcțiile de update() :

```
 if(lap > 0)
    {
        firstTime = true;
    }

    if (firstTime == true)
    {
        //update la textul pentru cursa curenta
        lapText.text = "Current lap: " + lap.ToString() +
"/3";
    }

    //timer masina
    timeCar = timeCar + Time.deltaTime;
    int minutes = (int)timeCar / 60;
    string minutesString;
    minutesString = minutes.ToString();
    if ( minutes < 10)
    {
        minutesString = "0" + minutes.ToString();
    }
    int seconds = (int)timeCar % 60;
    string secondsString;
    secondsString = seconds.ToString();
    if ( seconds < 10)
    {
        secondsString = "0" + seconds.ToString();
    }
    int fraction = (int)(timeCar * 100) % 100;
   // double fraction2 = System.Math.Round(fraction, 2);

    //dam update la fiecare frame la timer
    timerText.text = minutesString + ":" + secondsString + ":"
+ fraction.ToString();
```

În momentul în care trecem linia de final avem o funcție de OnTriggerEnter care mărește numărul de ture și modificăm textul pentru cea mai bună cursă dacă este cazul.

```
private void OnTriggerEnter(Collider other)
    {
       if(other.TryGetComponent<Goal>(out Goal goal))
        {
           // print(firstTime);
            lap++; // marim nr de ture
            if( firstTime == true)
            {
                float tempTime = timeCar - startTime; //1st lap
duration
                print(tempTime);

                //daca timpul turei curente este mai mic
                if (tempTime < tempMax)
                {
                    tempMax = tempTime;
                    //bestTimer.text = tempTime.ToString();
                    print(tempTime);

                    int minutesBest = (int)tempTime / 60;
                    string minutesBestString;
                    minutesBestString = minutesBest.ToString();
                    if(minutesBest < 10)
                    {
                        minutesBestString = "0" +
minutesBest.ToString();
                    }

                    int secondsBest = (int)tempTime % 60;
                    string secondBestString;
                    secondBestString = secondsBest.ToString();
                    if ( secondsBest < 10)
                    {
                        secondBestString = "0" +
secondsBest.ToString();
                    }

                    int fractionBest = (int)(tempTime * 100) %
100;
                    bestTimer.text ="Best lap: " +
minutesBestString + ":" + secondBestString + ":" +
fractionBest.ToString();



                }
                startTime = timeCar;
                print(startTime);
```

```
            }

             }
```

In momentul apăsarii tastei „P” apare meniul de pauză, iar pe tasta „R” putem reseta mașina în caz că jucătorul s-a blocat.

```
      if( Input.GetKey(KeyCode.P))
         {
             Time.timeScale = 0;
             pauseScreen.SetActive(true);
             //oprirea timpului jocului si activarea meniului de
pauza

         }

         if ( Input.GetKeyDown(KeyCode.R))
         {
             player.transform.position = new Vector3(58.60001f,
2.21f, -73.5f);
             playerRigid.velocity = Vector3.zero;
             player.transform.rotation = Quaternion.Euler(0f, 0f,
0f);
              }
```

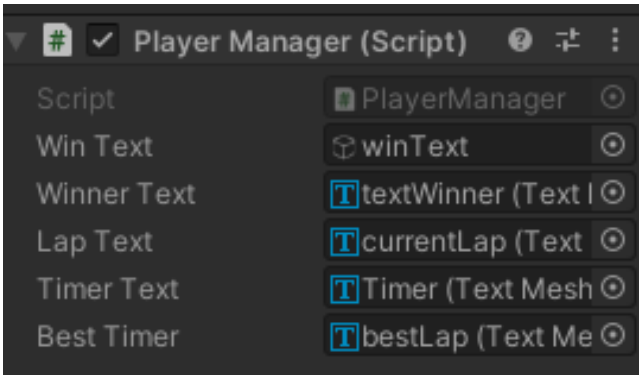


Figura 3.16 – Scriptul jucătorului ce primeste referinte catre obiectele de tip text

Figura 3.17 – Imagine din timpul cursei

3.1.4 Scena „player vs player”

Această scena este similară cu „Player vs AI”, având doar câteva diferențe în logica jocului Fiecare mașină a celor doi jucători are atașat un script de tipul „Player Manager”, astfel jucătorul care trece primul linia de final dupa 3 ture câștigă jocul.

Jucătorul al doilea are un script puțin modificat „CarController2”, diferența fiind faptul că acesta își controlează vehiculul prin intermediul tastelor I/J/K/L în loc de W/A/S/D.Pentru a îi putea diferenția axele de mișcare, am creat unele noi din Edit > Project Settings > Input Manager > Axis.

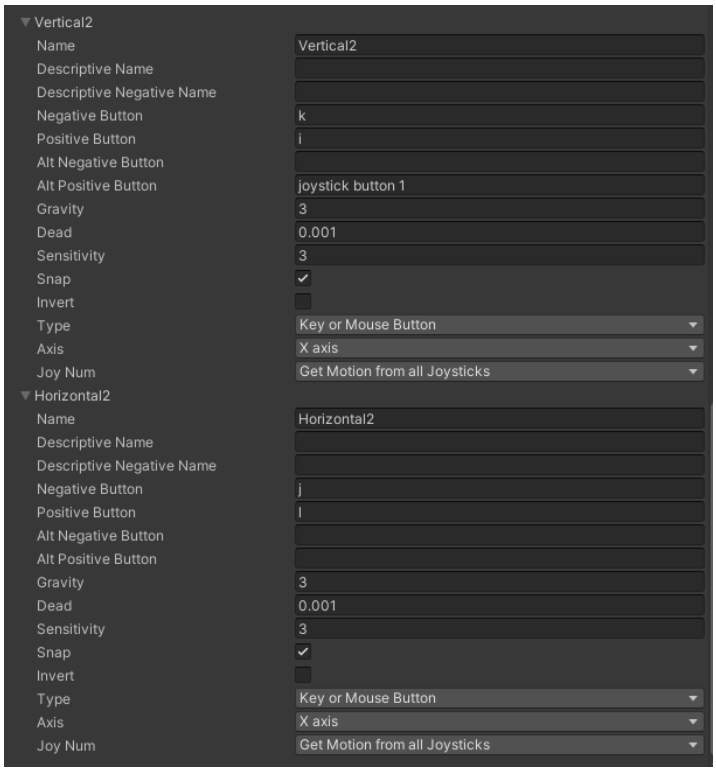


Figura 3.18 – Crearea de noi axe in “Input Manager”

De asemenea jucatorul 2 poate să își reseteze poziția mașinii apăsând tasta „O”

```
if (Input.GetKeyDown(KeyCode.O))
        {
            player2.transform.position = new Vector3(58.60001f,
2.21f, -73.5f);
            playerRigid2.velocity = Vector3.zero;
            player2.transform.rotation = Quaternion.Euler(0f, 0f,
0f);
        }
```

O altă modificare ar fi modul ecranului „split screen”, creat prin utilizarea a 2 camere, fiecare dintre ele avand width 0.5, iar una din ele având coordonataX 0.5.De asemenea, fiecare jucător are propria sa interfață.

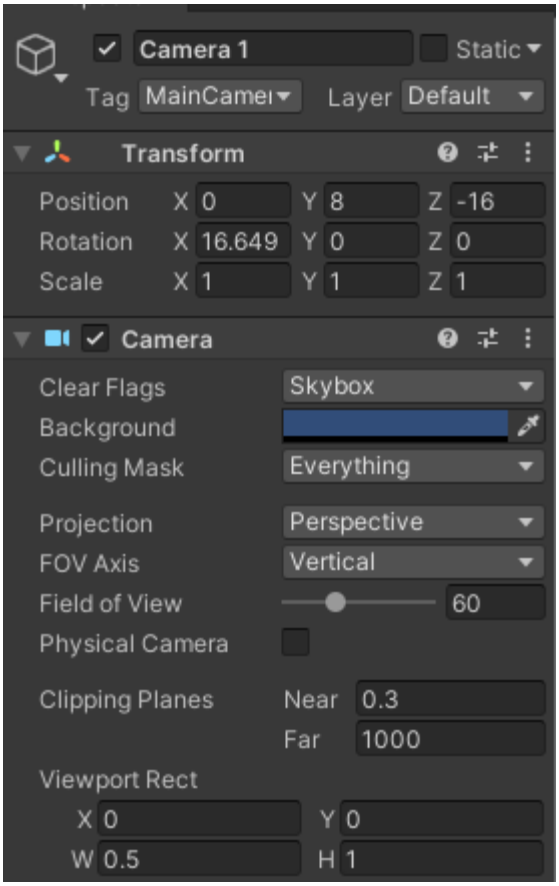


Figura 3.19 – Camera jucătorului 1

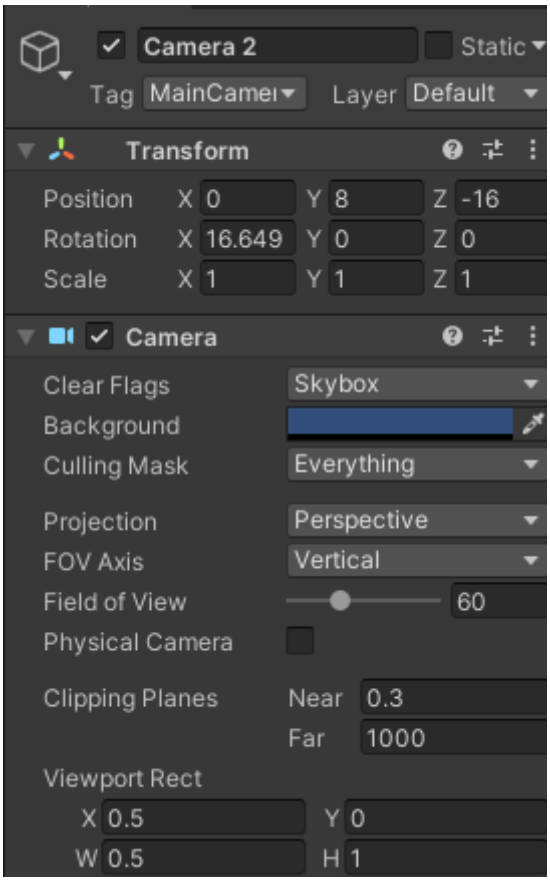


Figura 3.20 – Camera jucătorului 2

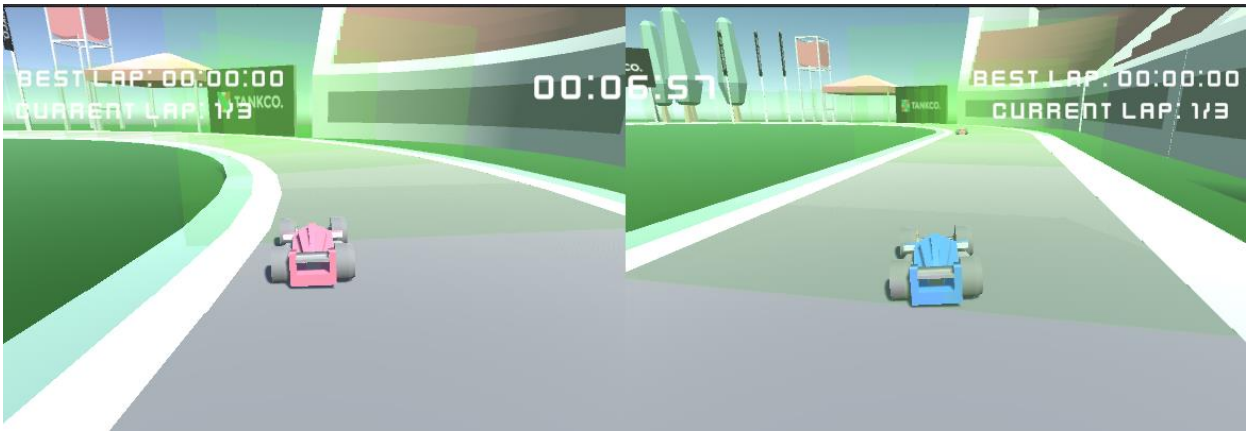


Figura 3.21 - Un screenshot din timpul cursei

3.1.5 Scena ”Watch AI”

În această scenă putem urmări agentul cum se antrenează.Acesta trebuie sa fie setat pe ”Behaviour Mode”: Interference Only și să aibe atribuit un ”creier” în câmpul model.Mai multe detalii despre agent pot fi citite în subcapitolul 3.2

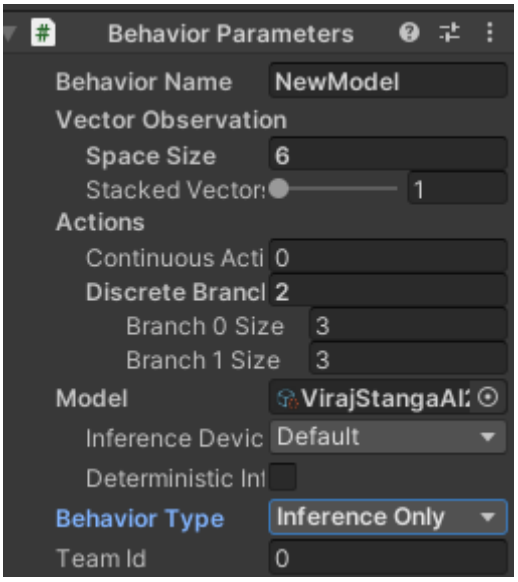


Figura 3.22 – Setările necesare agentului

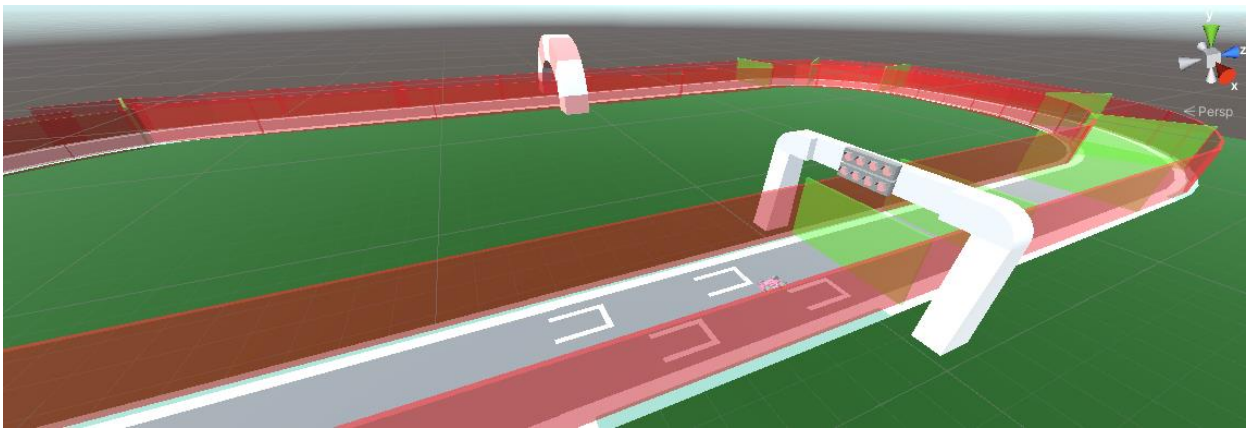


Figura 3.23 – Screenshot din timpul antrenarii agentului

3.1.6 Pause Menu

Meniul pentru pauză poate fi accesat în timpul jocului prin apăsarea tastei „P”.Acesta este creat pe un canvas într-un obiect nou numit „PauseMenu”.Acesta conține o imagine mai mare ușor transparentă ce acoperă canvasul, precum și o casetă mică neagra pe care se află meniul propriu zis.În aceasta sunt prezente și 3 butoane cu următoarele funcții: întoarcerea la meniu principal, resetarea jocului, sau reluarea lui.

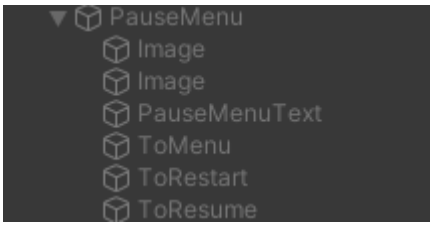


Figura 3.24 – Ierarhia obiectelor din meniul de pauză

Bucata de script pentru punerea pauzei și activarea meniului de pauză:

```
if(Input.GetKey(KeyCode.P))
        {
            Time.timeScale = 0;
            pauseScreen.SetActive(true);
            //oprirea timpului jocului si activarea meniului de
pauza

        }
```

Scriptul „PauseMenuScript” conține funcțiile butoanelor astfel:

```
public void toResume()
        {
            Time.timeScale = 1; //pornim iar timpul
            pauseScreen.SetActive(false);
        // dezactivam meniul de pauza

        }

public void toRestart()
    {


SceneManager.LoadScene(SceneManager.GetActiveScene().name);
        //restartam scena

    }

public void toMenu()
    {
        pauseScreen.SetActive(false);
        SceneManager.LoadScene("MainMenu");
        //incarcam scena meniului principal

    }
```

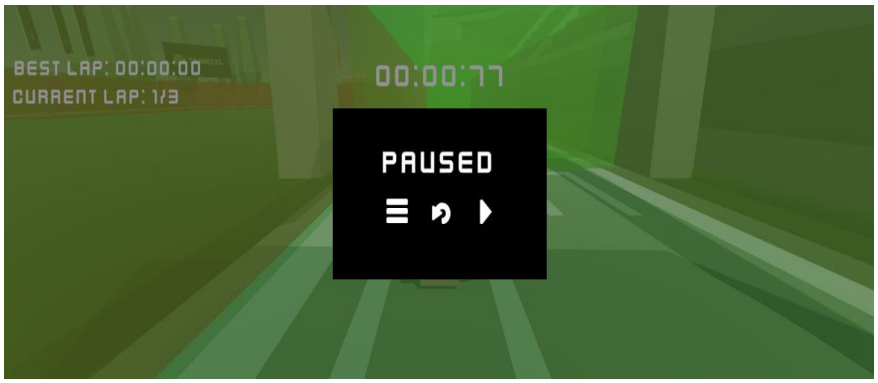


Figura 3.25 – Meniu de pauză

3.2 Training Agent si metode utilizate

3.2.1 Agent

3.2.1.1 Componente

Ca și model, Agentul folosește același tip de masină ca și utilizatorul.Agentul are atașate componente specifice precum: „Rigidbody”,”Box Collider”,”Behaviour Parameters”, „Demonstration Recorder”, „Ray Perception Sensor”, „Decision Requester” si două scripturi, „Training” si „CarControllerAI”. Componenta „Behaviour Parameters” controlează parametrii pe care îi folosește agentul nostru:

-„Behaviour Name” conține numele modelului nostru

-„Actions” conține informații despre acțiunile agentului:agentul nostru se mișca folosind acțiuni discrete, pe ramura 0 are mărimea de 3 (aceasta se poate mișca în fața / în spate / sau stă pe loc, respectiv valorile 1 / 0 / -1), iar pe ramura 1 are tot mărimea 3 (acesta poate vira la dreapta / stânga / sau poate să nu vireze deloc, respectiv valorile 1 / -1 / 0)

-„Vector Observations” conține informații referitoare la observațiile colectate de către agent.Space size are valoarea 6 deoarece noi colectăm două repere de coordonate Vector3, având suma 6 (3 + 3).

-în câmpul „Model” se poate încărca un creier cu rețea neuronală.

-„Behaviour type” putem alege comportamentul („Default” pentru antrenare, „Heuristic Only” pentru controlarea agentului cu butoanele de la tastatură si „Interference Only” pentru a vedea cum se mișcă singur, dacă are încărcat un creier).

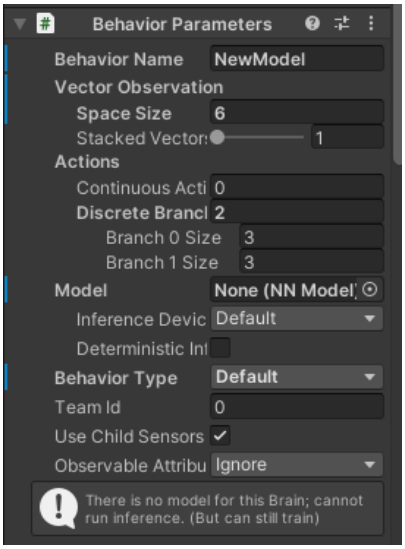


Figura 3.26 – Componenta Behaviour Parameters

Componenta „Decision Requester” are două câmpuri.”Decision Period” care îi setează la câți pași să ia o decizie, și „Take Actions Between” care îi permite sau nu să facă decizii și între perioade(fără efect dacă decision period este setat la 1).

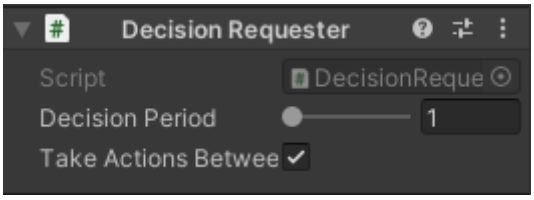


Figura 3.27 – Componenta Decision Requester

Componenta „Demonstration Recorder” permite înregistrarea de demonstraîii pentru Imitation Learning.Pentru a putea folosi această componentă, agentul trebuie să fie setat pe modul „Interference Only”. Aceasta componentă are câmpurile „Num Steps to Record” care ține cont de câți pași vor fi înregistrați pe demonstrație, „Demonstration Name” care setează numele demonstrației, „Record” care verifică dacă se va inregistra sau nu și „Demonstration Record” care setează unde va fi salvată demonstrația.

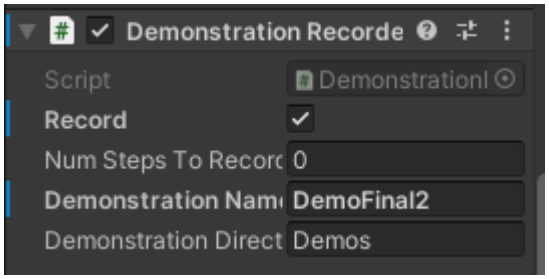


Figura 3.28 – Componenta Demonstration Recorder

Componenta “Ray Perception Sensor 3D” care controlează informațiile adiționale pentru agentul nostru:

-“Sensor Name” deține numele senzorului.

-“Detectable Tags” reține ce tip de elemente vor fi detectate de senzor, în cazul nostru: Wall/CheckPoint.

-“Rays per direction” care setează numărul de raze (acestea vin în formă de con, de ex 3 inseamnă 7 raze, 3 pe dreapta, 3 pe stânga si una în centru)

-“Ray Layer Mask” care verifică elementele făcând parte din care mască vor fi detectate

-“Start Vertical Offset” si “End Vertical Offset” care ajustează înalțimea punctului de începere / final al razelor.

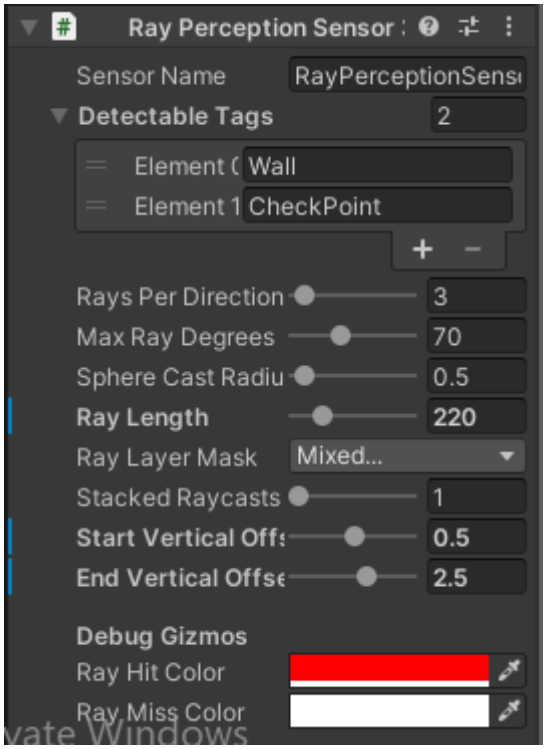


Figura 3.29 – Componenta Ray Perception Sensor

3.2.1.2 Script Agent

Agentul are atașate două scripturi, „CarControllerAI” și „Training”.

Scriptul „CarControllerAI” se ocupă de partea de deplasare, fiind o variantă ușor modificată a scriptului „CarControllerFinal”.Modificarea principală o reprezintă faptul că funcția Move() a fost parametrizată, pentru a îi oferi agentului posibilitatea de a o accesa.

```
public void Move(float forwardAmount, float turnAmount)
    {
        //Movement car
        LFrontColiider.motorTorque = forwardAmount * maxMotorPower
* 10;
        RFrontCollider.motorTorque = forwardAmount * maxMotorPower
* 10;
        float aditionalMotorPower = forwardAmount * 4000;
        rigidBodyCar.AddForce(transform.forward *
aditionalMotorPower);

        //Steering
        float currentTurnAngle = maxSteeringAngle * turnAmount;
        LFrontColiider.steerAngle = currentTurnAngle;
        RFrontCollider.steerAngle = currentTurnAngle;
    }
```

Scriptul „Training” are în componența sa partea logică de gândire a agentului.În funcția de Awake() inițializăm „Rigidbody”, iar într-un array de Vector3, checkpointPositions ținem minte coordonatele checkpointurilor.

```
private void Awake()
    {
        //initializam rigidbody
        agentRigidBody = GetComponent<Rigidbody>();


        //creem un array de Vector3
        checkpointPositions = new Vector3[100];
        //punem in checkpointsTransform obiecte de tip CheckPoints
Transform
        Transform checkpointsTransform =
manager.transform.Find("CheckPoints");
        init = 0;
        foreach(Transform checkpointSingle in
checkpointsTransform)
        {
            //print(checkpointSingle.name);
            checkpointPositions[init] =
checkpointSingle.transform.position; // array of Vector3
            init++;

        }

        }
```

Funcția OnEpisodeBegin() setează instantele ce țin de agent la începutul episodului.

```
public override void OnEpisodeBegin()
    {
        //punem pozitia de start a masinii la linia de inceput
        transform.position = new Vector3(60.1f, 2.38f, -110.0f);
        transform.rotation = Quaternion.Euler(0f,0f,0f);
        agentRigidBody.velocity = Vector3.zero;
        reward = 0;
        checkp = 0;
        //setam contorul pentru checkpoints la 0
        i = 0;
        //setam contorul pentru numarul de curse la 0
        lapCount = 0;


    }
```

Functia CollectObservations(VectorSensor sensor) colectează informațiile despre mediul de antrenament pentru agent.Funcția colectează în permanența coordonatele agentului și coordonatele următorului checkpoint, ambele sub forma unor seturi de coordinate de tip Vector3.

```
public override void CollectObservations(VectorSensor sensor)
    {

        //coordonatele agentului
        sensor.AddObservation(transform.position);

        sensor.AddObservation(checkpointPositions[i]);
//coordonatele Vector3 ale urmatorului checkpoint
       // print("Urmeaza checkpoint " + i);

    }
```

Funcția void Heuristic(in ActionBuffers actionsOut) se ocupă de deplasarea agentului folosind controalele de la tastatură, pentru modul “Heuristic Only”.

Functia void OnActionReceived(ActionBuffers actions) realizează deplasarea agentului în modul “Default” și “Interference Only”, permițând acestuia să acceseze scriptul “CarControllerAi”

Pentru stabilirea recompenselor si penalizărilor primite de agent a fost utilizată funcția OnTriggerEnter(Collider collision) care verifică cu ce obiect s-a atins agentul (wall / checkpoint).

```
private void OnTriggerEnter(Collider collision)
    {
        //daca agentul se loveste de zid, ii dam un reward negativ
si resetam episodul
        if (collision.TryGetComponent<Wall>(out Wall wall))
        {
            AddReward(-1f);
            EndEpisode();
        }

        if (collision.TryGetComponent<CheckPoint>(out CheckPoint
checkpoint) && checkpoint.pozitie == checkpointPositions[i])
        {

            print("Ai intrat in checkpoint " + i);
            i++;
           // print(checkpointPositions.Length);
            if( i == 14)
            {
                i = 0;
               // AddReward(+1f);
```

```
                AddReward(+0.5f); //bonus reward pentru trecerea
liniei de finish
                lapCount++;
                // EndEpisode();


            }

            if(lapCount == 3)
            {
                //daca agentul a parcurs 3 ture resetam episodul
                EndEpisode();
            }
           // AddReward(+1f);
            AddReward(+0.3f); // un reward de + 0.3 pentru fiecare
checkpoint atins

        }

        //daca agentul se loveste de checkpointul gresit, resetam
episodul si ii dam un reward negativ
        else if (collision.TryGetComponent<CheckPoint>(out
CheckPoint checkpoint2) && checkpoint2.pozitie !=
checkpointPositions[i])
        {
           // print("Wrong Checkpoint");
            AddReward(-1f);
            EndEpisode();
        }
}
```

3.2.2 Mediu de antrenare agent

Partea cea mai complicată pentru antrenarea unui agent cu inteligență artificială este alegerea unui mediu de antrenament potrivit care facilitează un bun proces de învățare.Pentru antrenarea agentului aplicației am ales sa înconjor pista cu pereți, iar pe parcursul acesteia să pun checkpointuri.

Agentul va fi pedepsit atunci când se va lovi de pereți si recompensat atunci când parcurge checkpointurile în ordine.

3.2.2.1 Mediu de antrenare initial

Mediul inițial de invățare avea checkpointuri dese, iar agentul era recompensat cu cate +1

pentru fiecare checkpoint ales si cu -1 pentru lovirea peretelui.O dată ce agentul atinge un perete episodul se încheie.

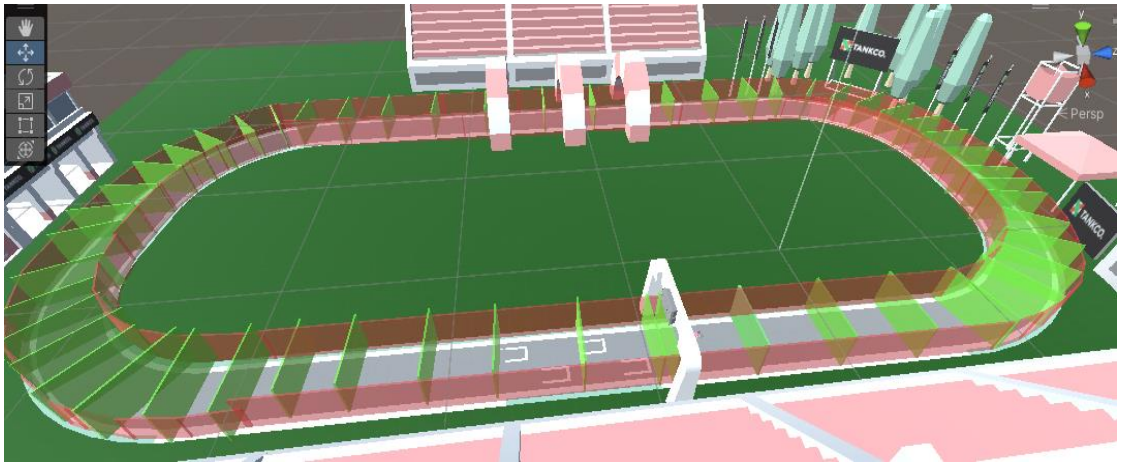


Figura 3.30 – Mediul 1 de antrenament

3.2.2.2 Mediu de antrenare modificat

În urma testelor și antrenamentelor, am modificat mediul de învățare făcând pereții solizi, agentul primind -1 pentru fiecare perete lovit, si -0.1 pentru fiecare secundă în care menține contactul cu zidul.Agentul primea în continuare +1 pentru fiecare checkpoint atins.

3.2.2.3 Mediu de antrenare final

După o altă serie numeroasă de teste, am redus numărul de checkpointuri deoarece acestea erau prea dese și agentul nu reușea să învețe destul.După reducerea numărului lor și a recompensei primite de la +1 la 0.3, agentul a înregistrat progres rapid.Pedeapsa pentru atingerea peretelui ramane -1, peretii nu mai sunt solizi, iar episodul se incheie.

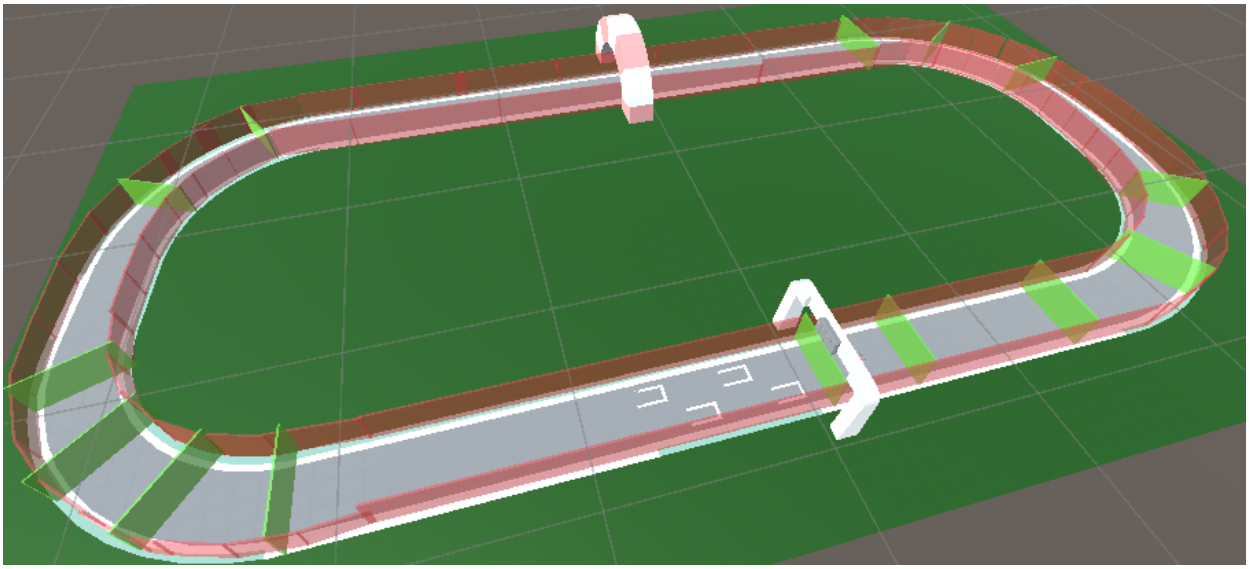


Figura 3.31 – Mediu final de antrenament

3.2.3 Procesul de antrenare agent

Pentru o mai rapidă antrenare, am creat un prefab cu agentul și l-am duplicat, punând toți agenții pe același layer și modificând coliziunile dintre aceștia.În acest mod am reușit să antrenez simultan 12 agenți ce contribuie cu informații la același creier.

Figura 3.32 – Agenti

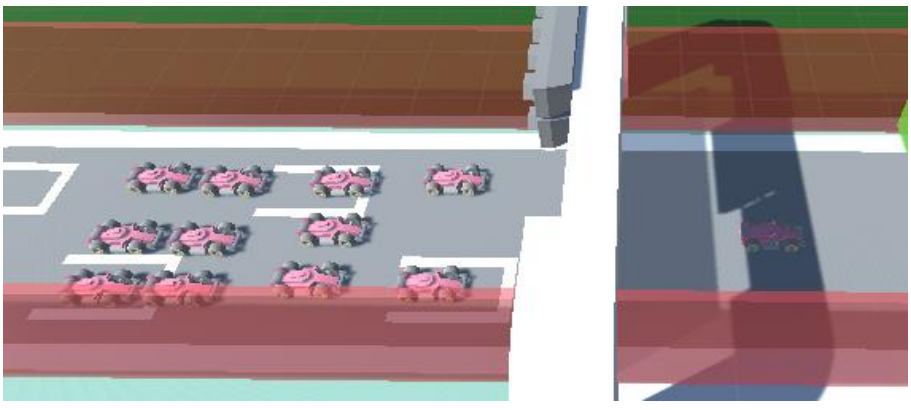


Figura 3.33 – Agenți pe pista

Pentru a începe o rundă de antrenare deschidem un cmd în folderul jocului, apoi rulăm comanda cd venv/Scripts/activate. Următoarea comandă ce trebuie rulată în terminal este mlagents-learn [cale catre fisierul de configuratie] [--initialize-from=”id” (optional)] [--run-id=”id”].Dupa rulareă acestei comenzi, apasăm butonul de play în Unity ,iar agentul începe să se antreneze automat

Pentru ca agentul să se poată antrena, behaviour type trebuie să fie setat pe „Default”.Pentru a opri procesul de invațare putem apăsa ctrl + c în cmd sau butonul de stop din editorul Unity.Rezultatele vor fi salvate în folderul results.

Figura 3.34 – Rularea comenzii de începere a antrenamentului

Rezultatele pot fi vizualizate sub formă de grafice folosind tensorboard, introducând comanda în cmd: tensorboard –logdir results, apoi accesând localhost:6006 în browser.

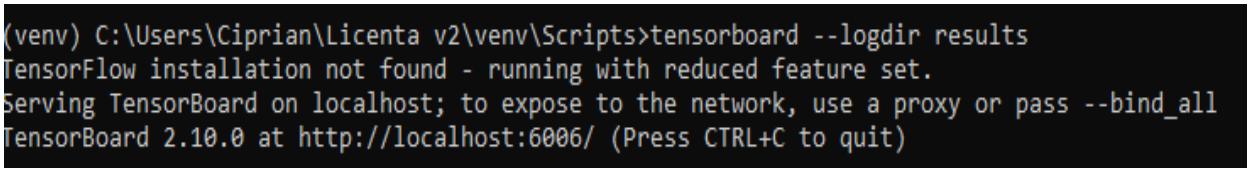


Figura 3.35 – Rularea tensorboard în cmd

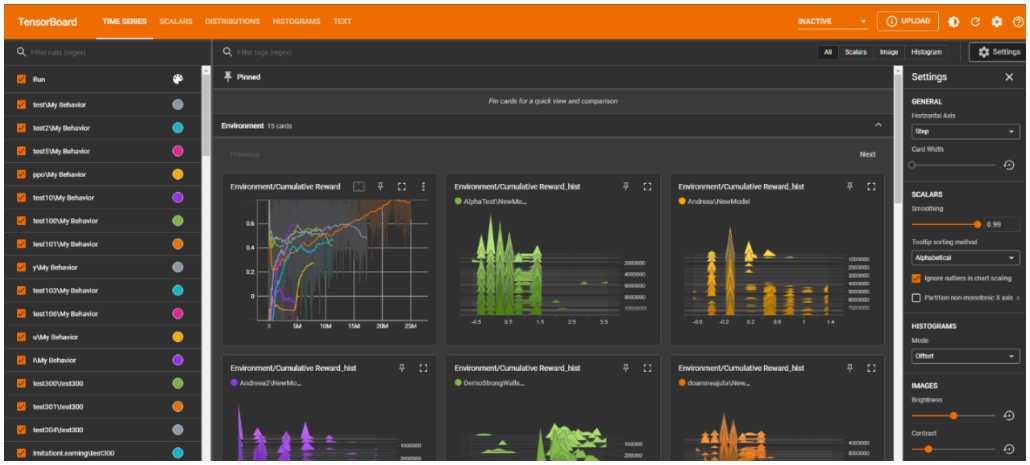


Figura 3.36 – Tensorboard dechis pe localhost:6006

Voi prezenta mai jos câțiva parametrii relevanți din fișierul config.yaml:

-„trainer_type” alege tipul de algorithm folosit dintre ppo/sac/poca.

-„batch_size” se ocupă de numărul de experiențe din fiecare iterație, ar trebui sa fie de 10 ori mai mic decat „buffer_size” pentru ppo

-„buffer_size” setează numărul de experiențe pe care le colectează agentul înainte de a modifica modelul

-„hidden_units” setează numărul de unitați din straturile rețelei neuronale

-„num_layers” alege numărul de straturi din rețeaua neuronală

-„strength” alege numărul cu care se înmulțește recompensa dată

-„demo_path” selectează calea către demonstrație, în cazul utilizării

```yaml
behaviors:
  NewModel:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: constant
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 3
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
      gail:
        gamma: 0.99
        strength: 0.5
        demo_path: Demos/Demo1_5.demo
    behavioral_cloning:
      strength: 0.5
      demo_path: Demos/Demo1_5.demo
    max_steps: 25000000
    time_horizon: 64
    summary_freq: 10000
```

Figura 3.40 – Posibil format fișier config.yaml

3.2.4 Metodele de antrenare folosite

Cea mai mare provocare la încercarea de a antrena un agent o reprezintă găsirea parametrilor potriviți și a unui mediu de învățare propice.În acest capitol voi prezenta cateva idei legate de metodele de învățare, idei observate după sute de ore de antrenament.În imaginea de mai jos este prezentat unul dintre folderele proiectului în care au fost depozitați „creierii agenților”, fiecare folder conținând câte un creier antrenat aproximativ între 1M si 25M de pași.

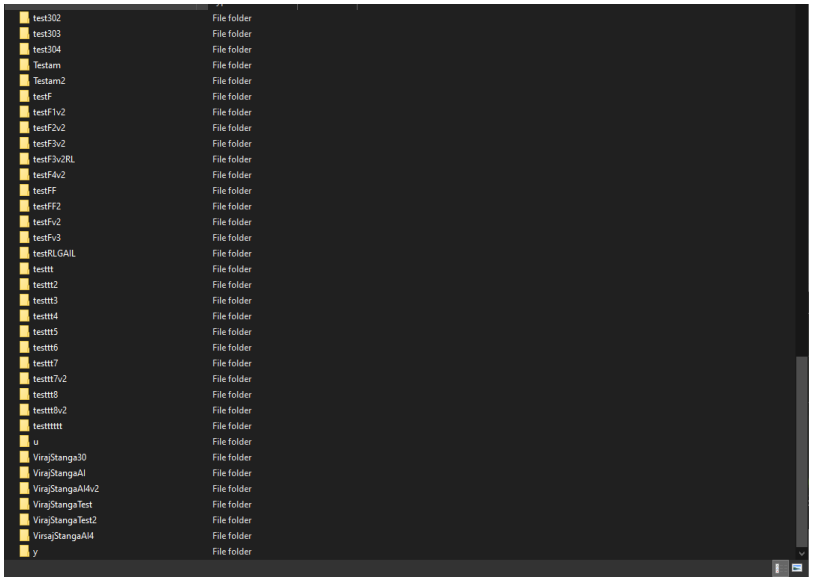


Figura 3.36 – Folderele cu creierii agenților

MLAgents ne pune la dispoziție 3 algoritmi de învățare, RL, GAIL și BC. Deși RL este cel mai puternic algoritm care ne permite creearea de AI mai pricepuți decât oamenii la anumite sarcini, acesta nu este foarte rapid la învățare de unul singur, astfel pentru sarcini mai complicate o preînvățare cu „Imitational Learning” este o abordare corectă.

Tabelele de mai jos vor arăta diferența de evoluție dintre un agent care a fost instruit doar cu RL (culoarea Mov) și altul care a fost instruit cu RL 1.0, GAIL 0.5 și BC 0.5 (culoarea Galben).

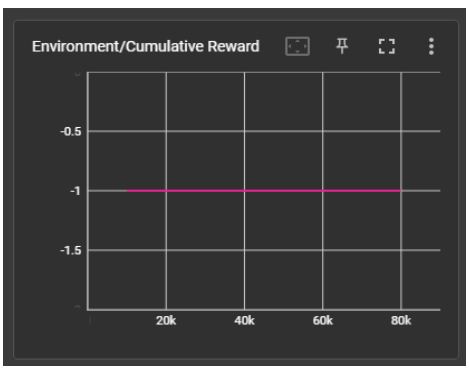


Figura 3.37 – Performanță Agent mov 80k

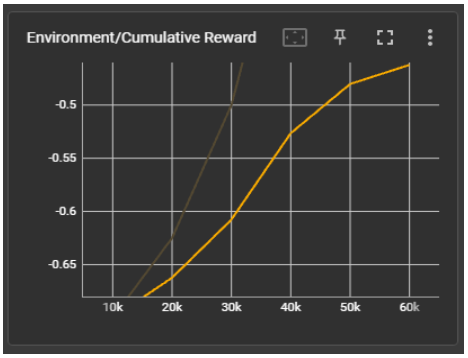


Figura 3.38 – Performanță Agent galben 80k

Figura 3.39 – Performanță Agent mov 160k

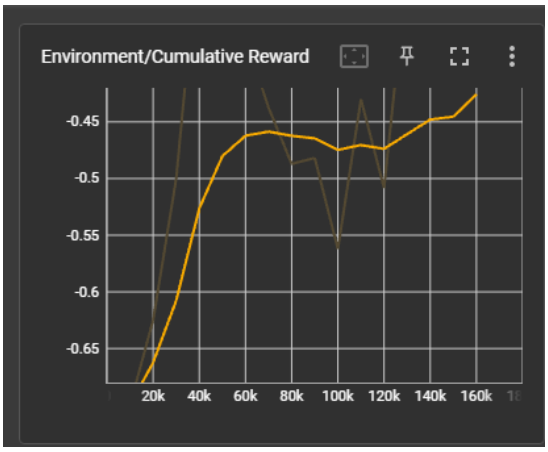


Figura 3.40 – Performanță Agent galben 160k

Figura 3.41 – Performanță Agent mov 200k

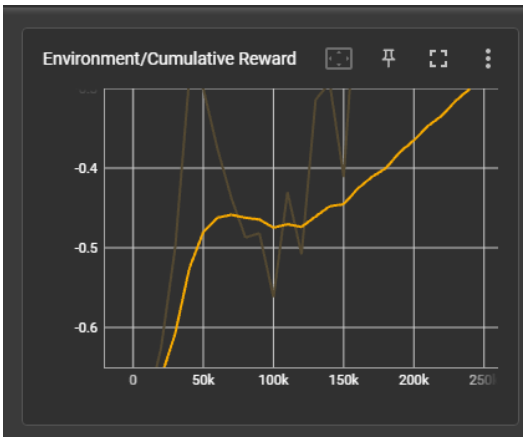


Figura 3.42 – Performanță Agent galben 200k

După cum se poate observa în imagini, în timp ce agentul mov nu a reușit încă să atingă vreun checkpoint, cel galben deja a reușit să ajungă, chiar și până la al treilea checkpoint în același număr de pași. Rezultatele din tabele arată faptul că, într-o fază inițială, învățarea agentului să ne imite demonstrația îi mărește viteza procesului de învățare, urmând ca ulterior să îi scădem puterea metodelor de „Imitation Learning”, dupa ce incepe să se prindă ce are de făcut. Idealul nostru este să îl ajutam să învețe cu GAIL si BC, apoi să îl lăsăm sa se perfecționeze cu RL pentru a deveni supraomenesc.

În graficul de mai jos, este prezentată curba de învățare a unui agent care a învățat prea mult cu BC și GAIL.Acest agent a început să țină prea mult cont de demonstrație și s-a oprit din a mai încerca să se adapteze la mediul înconjurător, rezultatele fiind dezastruoase.

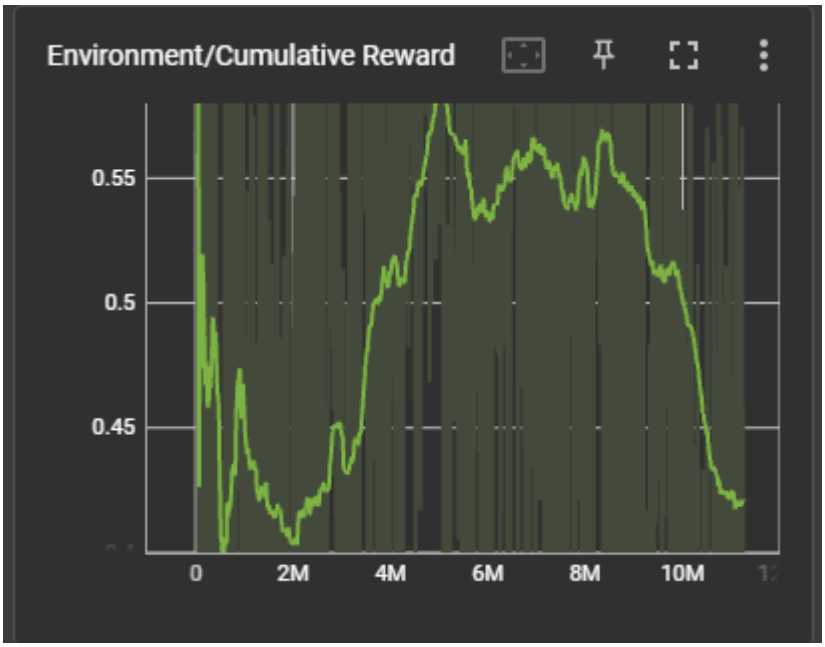


Figura 3.42 – Performanța agentului care a învățat cu prea mult BC și GAIL

O alta practică greșită poate fi schimbatul prea brusc de la o intensitate puternică de „Imitation Learning” și un RL slab, la un RL puternic și „Imitation Learning” slab.

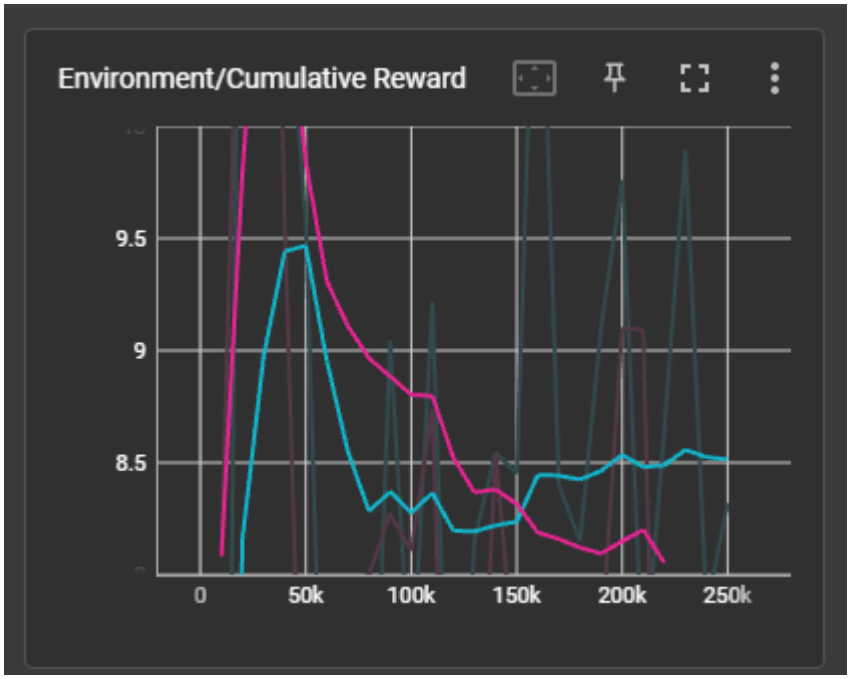


Figura 3.43 – Schimbarea prea rapidă de la Imitation Learning la RL puternic

O altă abordare pentru încercarea măririi performanței agentului, a fost aceea de a îi modifica mediul de invățare astfel încât să fie episoadele mai lungi. Astfel am făcut pereții solizi, nu am mai resetat episodul

atunci când agentul atinge peretele și am modificat sistemul de recompense, astfel încât agentul primește recompensă negativă de -0.1 pentru fiecare secundă în care rămâne lipit de perete. Din păcate, această încercare nu a dus la rezultate favorabile. Dupa mai mult timp de învățare, din cauza recompenselor negative prea mari, agentul începea să nu mai încerce să facă orice tip de mișcare ce ar fi putut să îl ducă în punctele în care lua recompense negative, creând un comportament dubios.

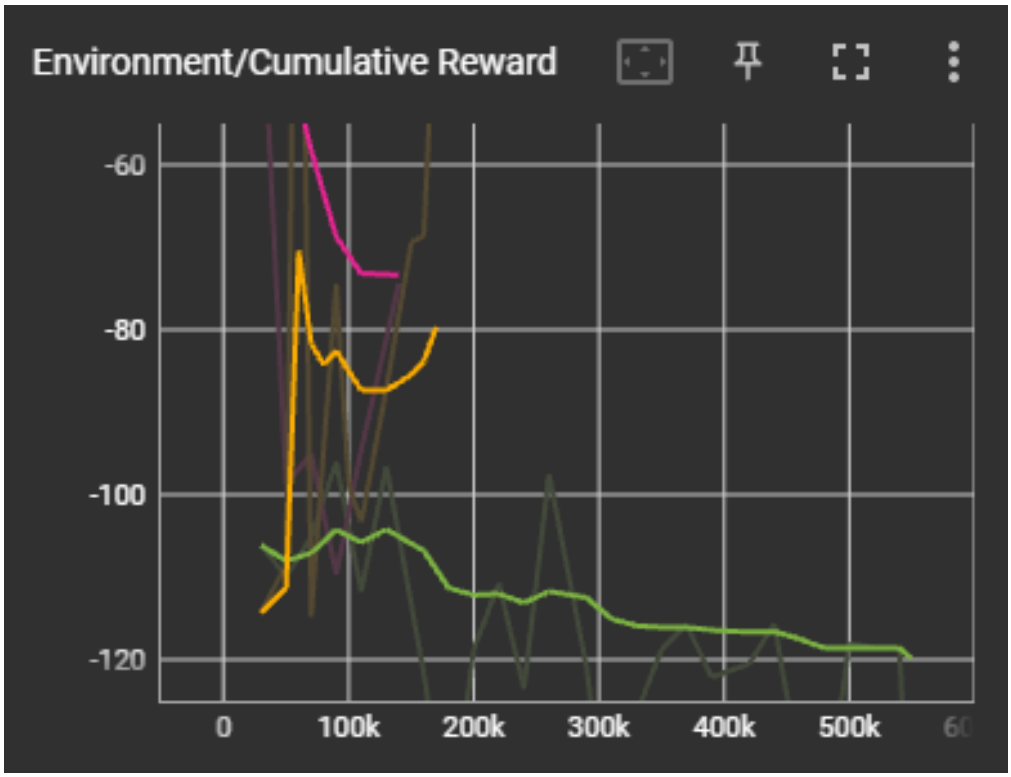


Figura 3.44 – Graficul recompenselor pentru mediul cu pereți solizi

O abordare care a dus la îmbunătățirea performantei a fost reducerea numărului de recompense primite de către agent, astfel agentul a fost forțat să încerce mai multe variante pentru a ajunge la recompense. Aceasta modificare a dus la o drastică mărire a performanței ce poate fi observată în curba de învățare a graficelor și a rezolvat problema stagnării agentului după aproximativ 500 000 de pasi.

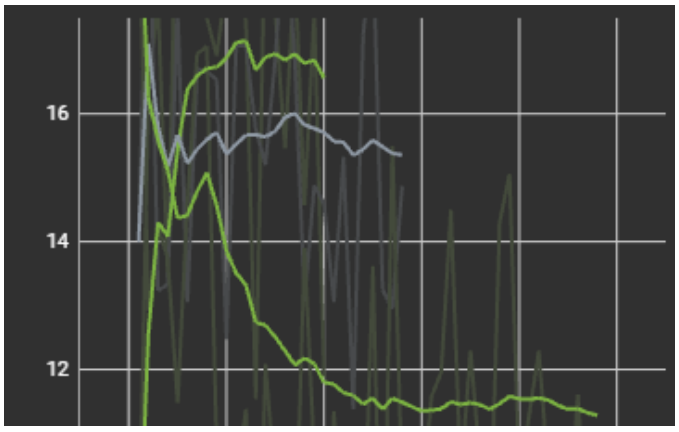


Figura 3.45 – Curba de invățare în mediul cu recompense mari

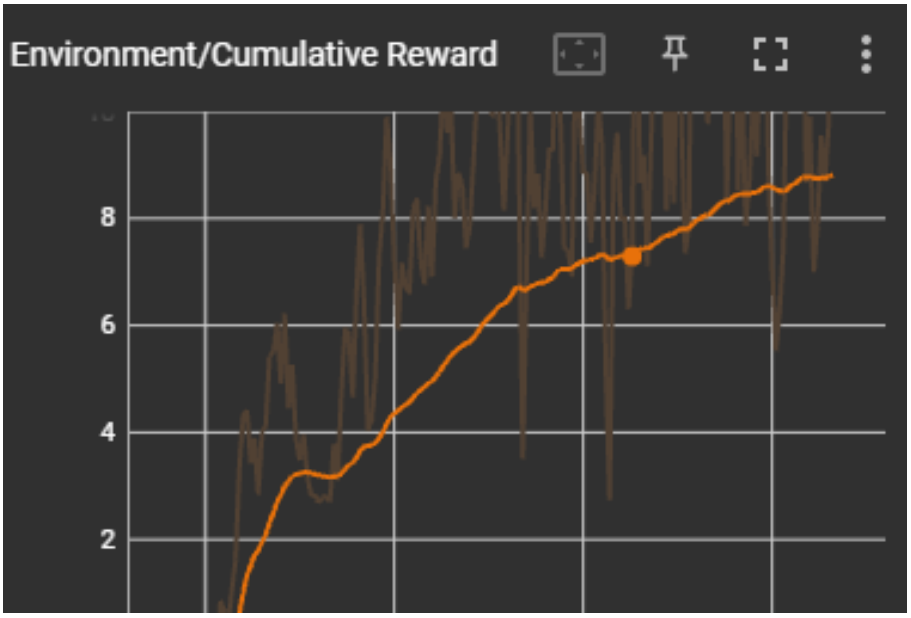


Figura 3.46 – Curba de invatare intr-un mediu cu recompense mai reduse

După sute de încercări, metoda care pare să înceapă să dea rezultate este antrenarea agentului cu un coeficient de Reinforce Learning de 1.0 , GAIL 0.5 și BC 0.5 pentru o perioada de aproximativ 5M de pasi, apoi continuarea trainingului cu Reinforcement Learning de 1.0, GAIL 0.1 și BC 0.1.Următoarele grafice vor arăta faptul că încercarea continuării cu un indice ridicat de GAIL și BC peste 5M de pasi cauzează reducerea performantei agentului.

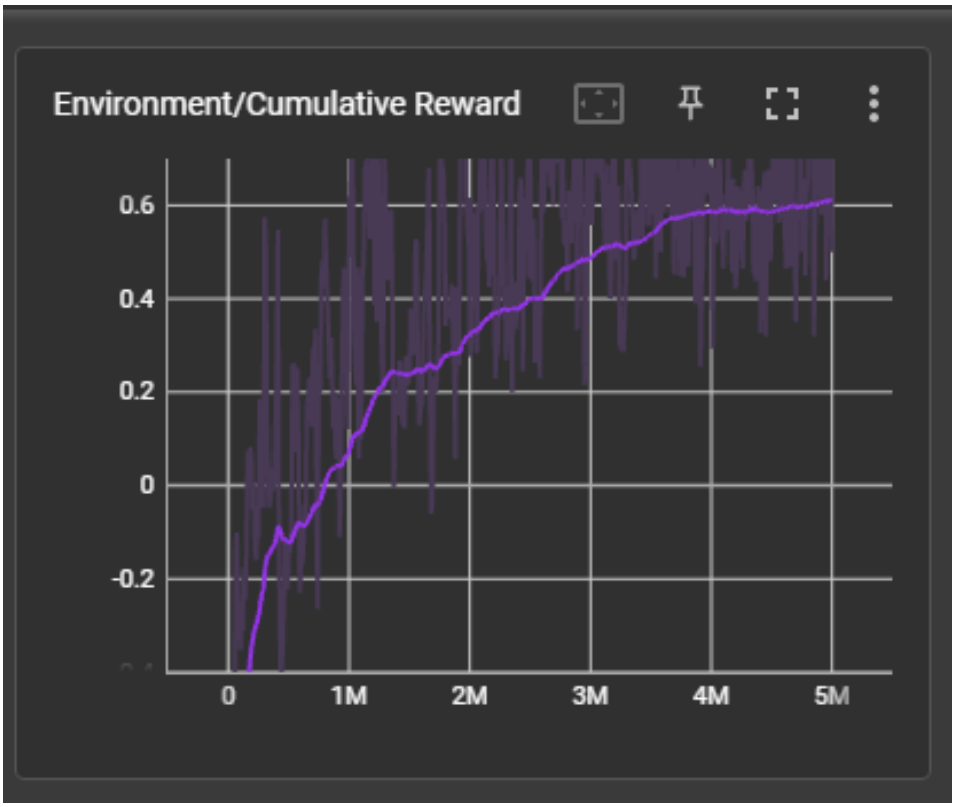
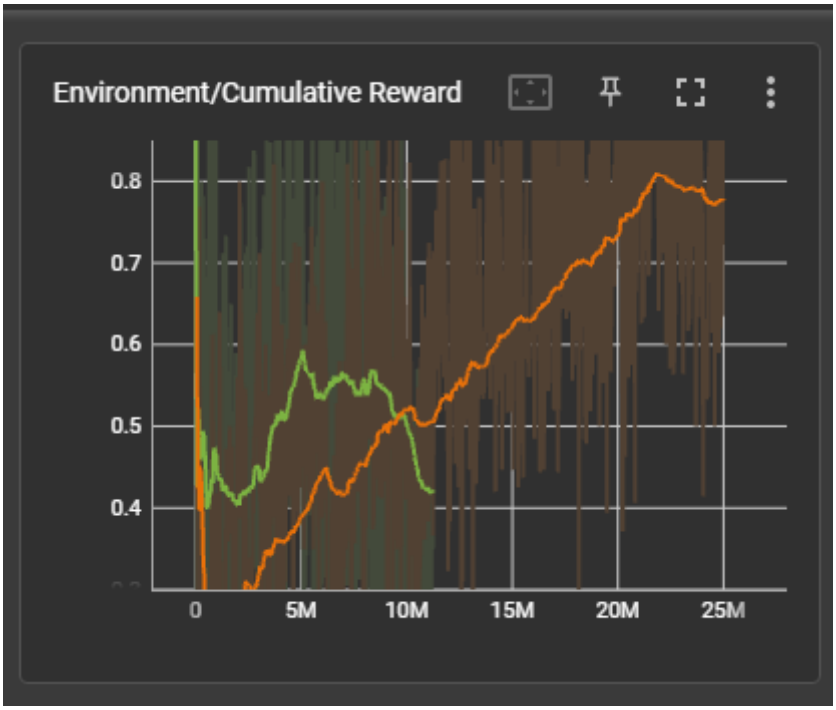


Figura 3.47 – Agent antrenat cu 1.0 RL, 0.5 GAIL, 0.5 BC

Următorul grafic reprezintă continuarea antrenării cu creierul din graficul din figura 3.47. Cu verde este reprezentat agentul cu aceleași setări, iar cu portocaliu agentul cu noile setări.După cum se poate observa, agentul verde deja începea să aibe o performanță din ce în ce mai slaba în timp ce agentul portocaliu învață constant.



Concluzii

Concluzii

Ca o primă părere personală despre librăria MLAgents, aceasta este foarte greu de instalat, apărând frecvent probleme, deoarece doar anumite versiuni ale pachetelor necesare pentru rularea acesteia sunt compatibile între ele.În multe cazuri este necesară încercarea a zeci de versiuni până se poate găsi un setup compatibil. De asemenea, și după instalare, deoarece librăria este „open-source” și încă în dezvoltare, apar numeroase erori, care cel mai frecvent ajung raportate pe forumul de la Unity, deoarece nu există soluție în prezent.Singurul pachet care a reușit să funcționeze pentru mine este pachetul cu versiunea 2.2.1-exp1, o versiune experimentală și nu foarte stabilă.

Pe parcursul dezvoltării aplicației, am aflat informații interesante despre procesul de dezvoltare al unui agent pe bază de inteligență artificială, procesul de creeare al unui joc, cat și despre importanța găsirii mediului de antrenare potrivit pentru învațarea sarcinii de către agent.

Cea mai mare provocare a fost reprezentată de găsirea unui mediu de învățare adecvat pentru agent, aceasta fiind una dintre cele mai mari probleme în domeniul inteligenței artificiale.Găsirea balansului între recompense și penalizări a fost o sarcină grea ce a avut nevoie de multe teste și încercări.

Ca metodă de învățare, pentru cazul discutat în lucrare, cea mai bună metodă a fost reprezentată de aplicarea unei preînvățări agentului, folosind o intensitate ridicată algoritmilor de „Imitational Learning” pentru aproximativ 5 milioane de pași, apoi scăderea intensității acestei metode în favoarea creșterii intensității algoritmului de „Reinforcement Learning”.

O îmbunătățire a aplicației ar putea fi continuarea antrenării agentului, deoarece aceasta este singura cale pentru a îl perfecționa, un agent performant avand nevoie de sute de milioane de pași parcurși în antrenamente.Din pacate, o astfel de ședință de antrenare ar necesita un sistem performant și o perioadă de timp destul de mare, de ordinul săptamânilor sau chiar lunilor.

Bibliografie

Bibliografie

[1] https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Installation.md [Accesat 09.08.2022]

[2] https://www.python.org/ [Accesat 09.08.2022]

[3] https://pytorch.org/ [Accesat 09.08.2022]

[4] https://forum.unity.com/forums/ml-agents.453/ [Accesat 09.08.2022]

[5] https://github.com/Unity-Technologies/ml-agents/releases [Accesat 09.08.2022]

[6] https://docs.unity3d.com/Manual/ [Accesat 12.08.2022]

[7] https://docs.unity3d.com/Manual/WheelColliderTutorial.html [Accesat 12.08.2022]

[8] https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents-Overview.md [Accesat 13.08.2022]

[9] https://docs.unity3d.com/Manual/com.unity.ml-agents.html [Accesat 14.08.2022]

[10] Thomas Simonini - https://towardsdatascience.com/an-introduction-to-unity-ml-agents-6238452fcf4c [Accesat 19.08.2022]

[11] https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md [Accesat 22.08.2022]

[12] https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-Imitation-Learning.md [Accesat 22.08.2022]

[13] https://github.com/Unity-Technologies/ml-agents/blob/release-0.13.1/docs/Learning-Environment-Best-Practices.md [Accesat 23.08.2022]

[14] https://github.com/Unity-Technologies/ml-agents/blob/release-0.13.1/docs/Learning-Environment-Create-New.md#optional-multiple-training-areas-within-the-same-scene [Accesat 25.08.2022]

[15] https://github.com/Unity-Technologies/ml-agents/blob/release-0.13.1/docs/Training-Curriculum-Learning.md [Accesat 25.08.2022]

[16] https://werplay.medium.com/unitys-ml-agents-and-effective-ways-of-training-3-3-e8e519af5ab7 [Accesat 25.08.2022]

[17] https://www.kenney.nl/assets/racing-kit [Accesat pe 22.08.2022]