Primjene metoda umjetne inteligencije u računalnim igrama

Čulina, Matko

Sveučilište u Rijeci – Odjel za informatiku

Jednopredmetni studij informatike

Matko Čulina

Primjene metoda umjetne inteligencije u računalnim igrama

Završni rad

Mentor: izv. prof. dr. sc. Marina Ivašić-Kos

Rijeka, rujan 2023

Sažetak

Ovaj završni rad istražuje primjenu umjetne inteligencije, posebno strojnog učenja, u razvoju trkaće video igre u Unity okruženju zvane ML-Racing. Igra omogućava strojno učenje agenata putem ML-Agents modula. Rad obuhvaća povijest video igara, razvojne alate poput Unitya i detalje o primjeni ML-Agents platforme.

U radu se naglašava povezanost industrije video igrara i evolucija umjetne inteligencije u igrama. Također pruža se uvid i u metode podržanog učenja i imitacijskog učenja te primjenu GAIL i behavioral cloning tehnika za treniranje agenata. Kroz ovaj rad, stvara se trkaća igra koja omogućava agentima da autonomno voze pomoću strojnog učenja, pružajući praktične primjere primjene ML-Agents platforme u Unity okruženju.

Ključne riječi

Unity, igra, 3d, strojno učenje, ML-Agents, auto, utrkivanje, C#

Sadržaj

Uvod

Video igre kao jedna od većih industrija procijenjena na 348 milijardi dolara 2022. godine [1] s konstantnim rastom, postale su dio najprofitabilnijih industrija na svijetu. Postoji nekoliko značajnih aspekata zašto je to tako. Naime video igre za razliku od drugih proizvoda nisu regionalno ograničene što znači da svi igrači na svijetu mogu igrati iste igre, tome je pridonijela i digitalna distribucija video igara uz pomoć platformi kao što su Playstation Network, Steam, Xbox Live te brojni drugi. Također veliki skok u industriji video igara ostvarile su mobilne igre. Iako popularne u ranijim danima poput „Snake-a“ , mobilne igre su trenutno na vrhuncu popularnosti što se može zahvaliti milijunima novih igrača diljem svijeta koji imaju pristup pametnim mobilnim uređajima, a koji možda nemaju pristup računalu ili drugim igračim konzolama. Također popularnosti pridonose i igre koje su besplatne i dostupne svima, ali generiraju profit pomoću mikrotransakcija [3]. Uz navedene aspekte ne možemo zanemariti ni E-sport element gdje se u pojedinim turnirima u igrama dijele milijuni dolara te se gledanost na Twitch platformi mjeri u stotinama tisuća gledatelja [4].

Povijest video igara seže unatrag nekoliko desetljeća. Iako nisu sve igre koristile umjetnu inteligenciju, ona se pojavljuje rano. Tako da se već 1951 godine u igri Nim pojavljuje umjetna inteligencija koja može pobijediti čovjeka [5], nastavno su se razvile umjetne inteligencije i za igre kao što su šah i dama što su kombinatorne igre. Sljedeći korak su bila je primjena umjetne inteligencije u arkadnim igrama u razdoblju 1970-ih i 1980-ih [6]. Iako popularne, arkadne igre su bile ograničene procesorskom moći i memorijom tako da su igre pratile jednostavne obrasce ponašanja kao npr. Space Invaders [7] gdje se neprijatelji kreću lijevo-desno i prema dolje. Neprijatelji u igrama su bili lako predvidljivi jer su koristili uvijek iste putanje i akcije, dok se povećavanjem težine uglavnom povećavala brzina ili broj neprijatelja, ali ne i njihova inteligencija u izvođenju radnji. Od 2000-ih razvoj umjetne inteligencije raste eksponencijalno uz pomoć razvoja hardware-a, tako da se npr. NPC-ovi (eng. non-playable characters) u igrama poput Grand Theft Auto V ili Skyrim ponašaju kao ljudi te obavljaju svoje zadaće neovisno o igraču. Razvoj tehnologija poput strojnog učenja ili korištenja chatbot modela poput ChatGPT-a omogućit će NPC-ove za koje nije potrebno

prethodno odrediti sve linije teksta u razgovoru nego će umjetna inteligencija imati zadane osobine te svoju svrhu i cilj dok će se ostalo generirati u hodu [8]. To će omogućiti veliku raznolikost igara, te otvorit opciju ponovnog igranja iste igre, jer se razgovori neće ponavljati.

Razvoj trkaćih igara je započeo u razdoblju popularizacije arkadnih igara. Jedna od prvih trkaćih igara s umjetnom inteligencijom je Pole Position iz 1982. godine [9] gdje se protivnici oslanjaju na jednostavne heuristike. Veliki skok u razvoju umjetne inteligencije u trkaćim igrama se dogodio 2000-ih gdje su se razvili protivnici koji voze slično stvarnim vozačima te reagiraju na posebne prometne uvjete poput kiše i snijega.

Strojno učenje omogućava razvoj igara koje će se sve više približiti stvarnom svijetu u svojim simulacijama. Ono se bavi razvojem algoritama i modela koji omogućavaju računalima da uče iz podataka i donose predviđanja ili donose odluke bez izričitog programiranja. Strojno učenje ima brojne praktične primjene uključujući prepoznavanje uzoraka u medicinskim nalazima, preporučivanje proizvoda na temelju prethodnih kupnji, analizu teksta, jezika, matematičkih problema i mnoge druge. Strojno učenje možemo podijeliti u nekoliko vrsta kao što je nadzirano učenje koje se često koristi za rješavanje problema gdje je potrebno predvidjeti izlaz na temelju ulaza i označenih podataka. Također poznajemo i nenadzirano učenje gdje modeli nemaju označene podatke. U ovom slučaju za nas je bitno podržano i imitacijsko učenje. Pri podržanom učenju agent pokušava naučiti donositi odluku koja će maksimizirati njegovu nagradu prilikom interakcije s okolinom dok se pri imitacijskom učenju agent pokušava naučiti tako da oponaša ponašanje iz demonstracija koje mu zada korisnik.

Razvojni alati

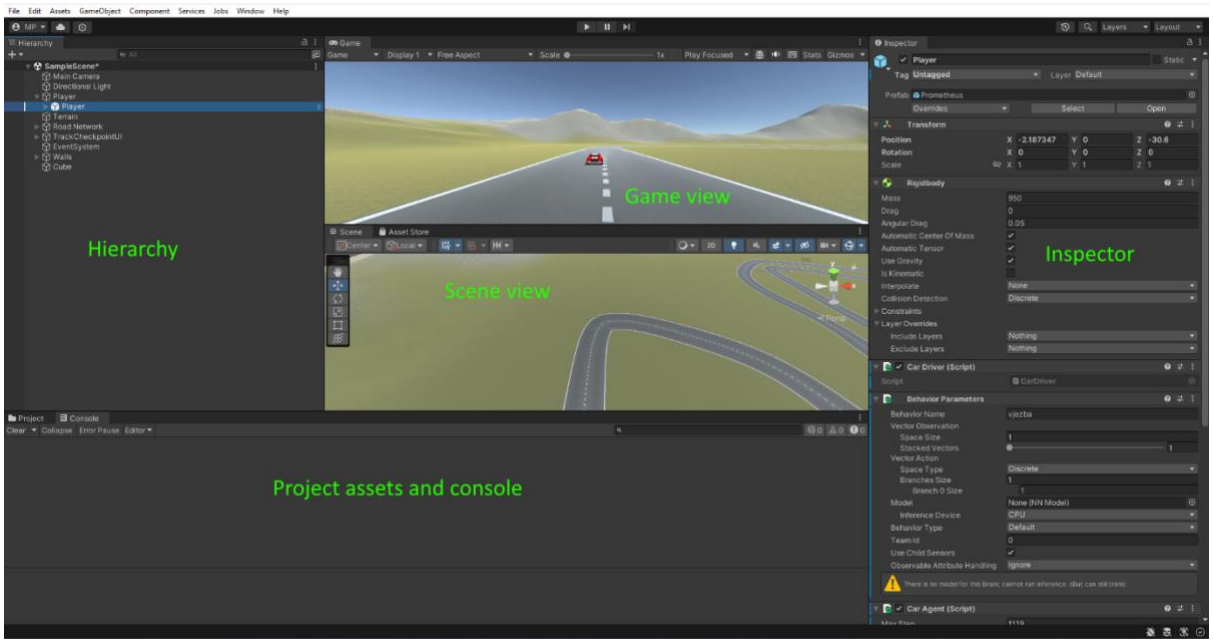
Za izradu igre ML-Racing korišteni su razni alati.

Unity [10] je popularna razvojna platforma za izradu interaktivnih 2D i 3D aplikacija. Prednosti Unity platforme su to što se može koristiti za veliki opseg zadataka poput razvoja igara, medicinskih simulacija, simulacija virtualne i proširene stvarnosti, te se čak primjenjuje i u arhitekturi. Unity podržava razvoj igara istovremeno za više platformi poput PC, Playstation, Android itd. Također

velika prednost Unitya naspram drugih razvojnih alata je Unity Asset Store gdje se mogu kupovati i prodavati razni resursi potrebni za razvoj aplikacija. Unity koristi C# programski jezik, iako podržava i druge jezike poput JavaScripta i Boo-a, C# je ipak najpopularniji.

Sučelje alata Unity

U sljedećem poglavlju bit će prikazano i objašnjeno sučelje alata Unity, Slika 1.



Slika 1. Prikaz grafičkog sučelja alata Unity

Hierarchy je prozor koji prikazuje hijerarhijsku strukturu svih objekata u nekoj sceni. Objekti su prikazani kao ikone, a svaki ima svoje ime. U tom panelu možemo grupirati objekte u odnose roditelj-dijete, dodavati nove objekte kao i aktivirati i deaktivirati ih.

Game view je jedan od glavnih prozora koji omogućuje pregledavanje i interakciju s igrom koja se razvija. On omogućava pregled igre u stvarnom vremenu te jednostavno testiranje funkcionalnosti. Također vrlo je koristan i za testiranje različitih rezolucija i omjera ekrana te može pružiti i detaljnu statistiku poput broja sličica po sekundi, količine korištene memorije, broja renderiranih poligona i drugih statistika korisnih za razvoj igre.

Scene view je također jedan od glavnih prozora te na omogućava uređivanje i vizualizaciju scene. On nam omogućava uređivanje i postavljanje objekata iz različitih perspektiva i kutova.

Inspector je prozor koji koristimo za pregled, izmjenu i dodavanje raznih komponenti i svojstava za objekte u sceni. U njemu možemo podesiti svojstva objekta poput veličine, orijentacije i pozicije. Dodati komponente poput skripti, animacija, efekata kao i elemente poput collidera i rigidbody-a.

Konzola je prozor koji prikazuje greške, upozorenja i poruke koje se generiraju pri izvođenju igre u Unity-u. Poruke su označene bijelom, upozorenja žutom a greške crvenom bojom. Konzola je vrlo korisna zbog korištenja Debug.Log() funkcije pomoću koje ispituje izvršava li se kod u skladu s našim željama ili gdje je kod zastao ako se ne izvrši do kraja.

Project assets je prozor koji omogućuje pregled i organizaciju svih resursa u projektu. U njemu se mogu naći skripte, modeli, zvukovi, materijali, teksture, prefabovi.

ML-Agents

ML-Agents (Machine Learning Agents) [11] je platforma otvorenog koda namijenjena razvoju i treniranju inteligentnih agenata u igrama, a često se koristi zajedno s Pythonom. Razvijena je za Unity te uz pomoć Python API-a omogućava interakciju s agentima i okolinama i izvan Unitya. Samo treniranje se provodi izvan Unitya, u Pythonu. ML-Agenti koriste PyTorch biblioteku za duboko učenje koju je razvio FAIR (Facebook Ai Research lab). Razlikujemo više vrsti učenja koje se koriste za treniranje agenata, ali u kontekstu primjene u igri najzanimljivije su podržano učenje i imitacijsko učenje [12].

Podržano učenje

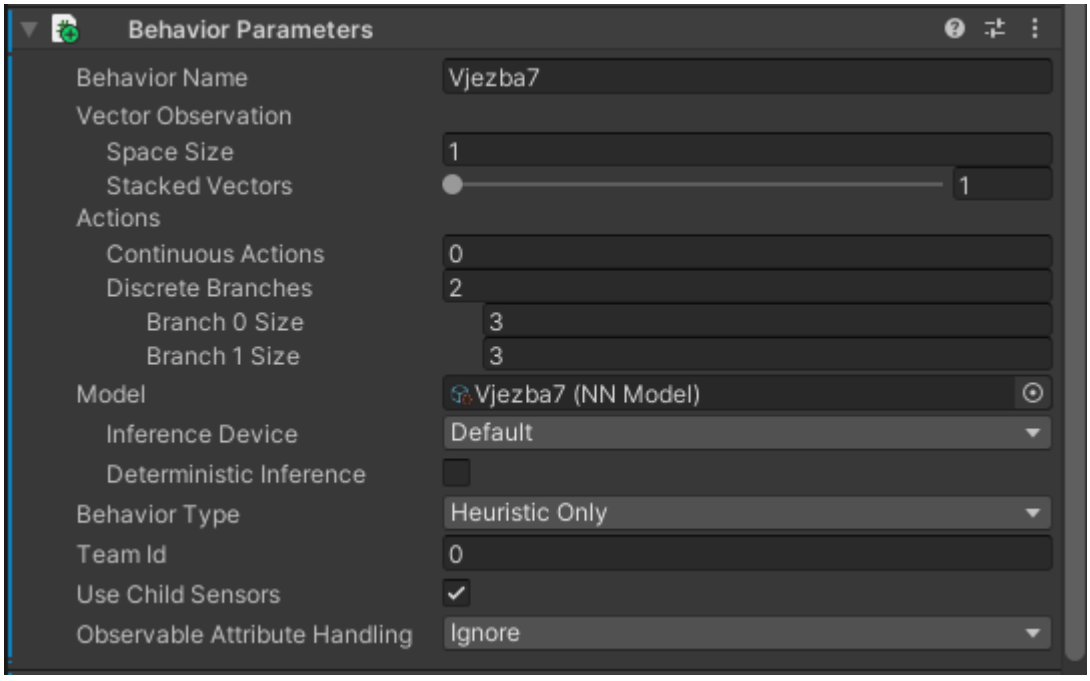
Podržano učenje je metoda strojnog učenja koja se bavi učenjem na temelju podataka koje agent ostvaruje s okolinom. U podržanom učenju koriste se nagrade i kazne kojima se nagrađuje, odnosno kažnjava agent ovisno o tome je li postigao željeni cilj ili mu se približio ili nije. Nagrade se koriste da bi se agentu dala povratna informacija jesu li njegove akcije bile korisne. Agent donosi odluke koje će maksimizirati ukupnu nagradu. Ovakav pristup može biti spor i dugotrajan.

Imitacijsko učenje

Imitacijsko učenje je metoda koja se koristi za treniranje uz pomoć prethodno pripremljenih demonstracija. Ova metoda je korisna kada je dostupan skup podataka koji se može koristiti kao primjer željenih akcija.

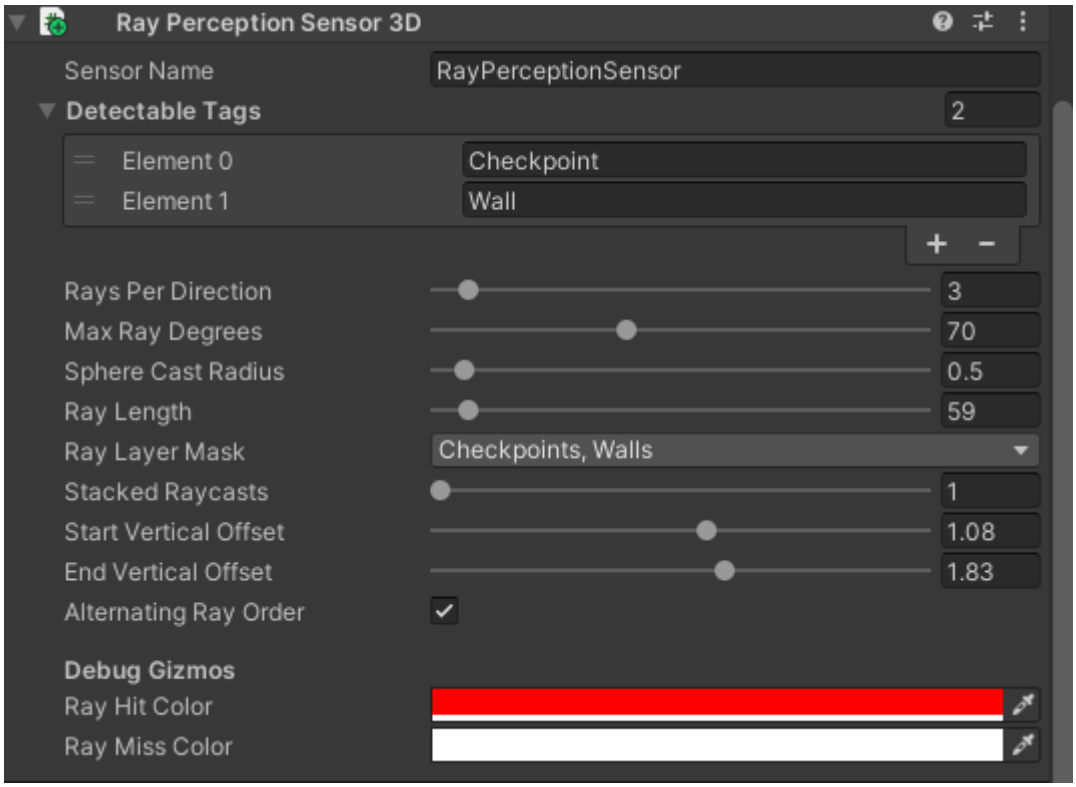
Korištenje ML-Agents modula

Da bismo koristili ML-Agents modul u Unityu potrebno je instalirati Python na računalu [13]. U ovom slučaju koristi se Anaconda koja pruža veliki broj opcija za računanje i analizu podataka. Potrebno je aktivirati novo okruženje u Anaconda Navigatoru te zatim pristupiti tom okruženju i klonirati ML-Agente na željenu lokaciju pomoću naredbene linije koju se unosi u Anaconda Prompt odnosno sučelje naredbenog retka. Nakon što smo klonirali datoteku od poslužitelja možemo unutar Unitya instalirati Unity paket za ML-Agente. Unutar upravitelja paketa odaberemo package.json iz prethodno klonirane datoteke ML-Agents. Ovim načinom smo sigurni da koristimo najnoviju verziju sučelja. Zatim instaliramo pytorch i ML-Agente iz naredbenog retka. Sada nam se u inspector prozoru pri dodavanju nove komponente pojavljuju i funkcije ML-Agenata kao Behavior parameters (Slika 2.), Decision requester, Ray Perception Sensor 3D i ostali.



Slika 2. Prikaz Behavour Parameters komponente

Behavior Parameters je ključni dio konfiguracije strojnog učenja. U njemu možemo postaviti broje korisne opcije kao broj agenata, broj akcija koje agent može koristiti, kao i model mozga i opcija za korištenje samo unosa igrača ili samo korištenje umjetne inteligencije i mozga. Također jako bitan dio je i Ray Perception Sensor 3D (Slika 3.) koji nam omogućava jednostavno projiciranje sfera pomoću kojih možemo odrediti objekte koje one dotiču. Također možemo koristiti i Demonstration Recorder pomoću kojeg možemo snimiti svoje akcije te ih koristiti u imitacijskom učenju agenata.



Slika 3. Prikaz Ray Perception Sensor 3D komponente

Demonstracija korištenja ML-agenata u trkaćoj igri

Ideja ovog rada bila je prikazati korištenje ML-Agent apija u Unity alatu. Iz tog razloga je odabrana jednostavna demonstracijska trkaća igra u kojoj igrač ili agent treba voziti po stazi i doći do cilja u što kraćem vremenu i sa što manje sudara. U igri se nalaze dvije kružne staze stvorene radi učenja lijevog i desnog

zavoja te jedna konvencionalna staza. U ovom slučaju koristi se isti model vozila za igrača i umjetnu inteligenciju, a model i njegova svojstva su zadani prilikom izrade igre. Također igra sadrži modul ML agenta koji je treniran podržanim i imitacijskim učenjem.

Izrada demonstracijske računalne igre

Izrada igre je prilično jednostavna, uključuje teren na kojem je staza, te izradu staza i igrača. Prvo je bilo potrebno napraviti teren na kojem će se igra odvijati. Teren se napravi s opcijom Desni klik>3D Object>Terrain. Za dodatne opcije uređivanja terena je korišten dodatak Terrain Sample Asset Pack. Nakon stvaranja terena koristeći inspector dodatno uređujemo oblik terena pomoću kistova za izdizanje i spuštanje terena kao i teksturu uz pomoć kista za bojanje određenom teksturom.

Sljedeći korak je bio izrada staze po kojoj će se voziti. Staze se mogu izraditi slaganjima i uređivanjima više raznih elemenata, ali u ovom slučaju je izrađena pomoću alata EasyRoads3D koji je besplatno dostupan u Unity dućanu. Prvo se stvori Road Network te se zatim u Inspectoru koristi opcija AddRoadObject kojom se iscrtavaju staze na terenu. U Inspectoru se određuje širina staze i podešavaju se dodatne opcije koje nam za ovaj primjer nisu bile potrebne.

U igri se napravljene dvije kružne staze koje su namijenjene za treniranje lijevih odnosno desnih zavoja, te jedna konvencionalna staza.

Za model igrača korišten je Unity asset iz dućana pod nazivom Prometeo Car controller. Iako uz model vozila dolazi i skripta za kontroliranje vozila, nju nismo koristili u ovom slučaju obzirom da je skripta kompleksnija i nije u potpunosti prikladna za strojno učenje. Model igrača smo dodali u izbornik te smo ga odvukli po želji na stazu.

Problem izrade staze i checkpointa

S obzirom da u igri koristimo strojno učenje prvi problem pri izradi je bio kako organizirati stazu u podatke koje umjetna inteligencija (nadalje UI) može pročitati. Kao najbolja opcija pokazala se izrada checkpointa i zidova koje UI može pročitati.

Checkpoints

Za izradu checkpointa koristimo novi 3D objekt odnosno kocku kojoj uklanjamo MeshRenderer da bi postala prozirna odnosno prazna. Također U Box Collideru

koristimo opciju Is Trigger što znači da objekt više neće fizički odbijati druge objekte kada dođe do kolizije nego će generirati događaje koji se mogu detektirati i obrađivati u skriptama. Kada smo zadovoljni s oblikom checkpointa koji mora obuhvatiti cijelu širinu staze, možemo stvoriti novu skriptu koju vežemo za taj objekt. Tada možemo preimenovati kocku u CheckpointSingle i stvoriti prefab tako da odvučemo kocku iz hijerarhije u assete. Prefab predstavlja unaprijed definirani objekt ili skup objekata koji se može koristiti više puta prilikom izrade igre, što je vrlo korisno jer ova igra zahtjeva veliki broj checkpointa. On omogućava lako upravljanje s obzirom da se izmjena na prefabu odražava u svim instancama tog prefab objekta. Zatim uređujemo skriptu JedanCheckpoint (Slika 4.).

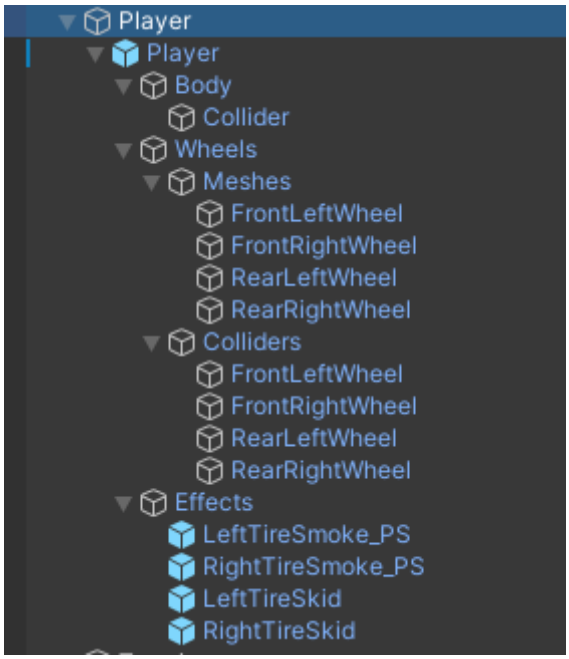
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class JedanCheckpoint : MonoBehaviour {

    private MeshRenderer meshRenderer;//za prikaz checkpointa
    private TrakaCheckpointova trakaCheckpointova; //pohrana za roditeljsku komponentu

    private void Awake() {
        meshRenderer = GetComponent<MeshRenderer>();
    }
    private void Start() {//Sakrij checkpoint na poetku igre
        Sakrij();
    }
    private void OnTriggerEnter(Collider drugi) {
        if (drugi.TryGetComponent<Player>(out Player player)) {//ako pri sudaru objekt koristi player skriptu
            trakaCheckpointova.AutomobilKrozCheckpoint(this, drugi.transform);//oznacavamo prolazak kroz checkpoint
        }
    }
    public void PostaviTrakuCheckpointova(TrakaCheckpointova trakaCheckpointova) {//povezujemo checkpint s trakom
        this.trakaCheckpointova = trakaCheckpointova;
    }
    public void Prikazi() {//prikazi checkpoint
        meshRenderer.enabled = true;
    }
    public void Sakrij() {
        meshRenderer.enabled = false;//sakrij  checkpoint
    }
}
```

Slika 4. Prikaz skripte JedanCheckpoint

Prvo definiramo klasu JedanCheckpoint te definiramo varijable trakaCheckpointova koja služi za pohranu reference na komponentu TrakaCheckpointova odnosno za obavijesti kada igrač prođe kroz checkpoint i meshRenderer. Varijablu meshRenderer koristimo da bismo sakrili ili prikazali checkpoint po pogrešnom odnosno ispravnom prolasku kroz checkpoint. Na kraju koda se nalaze i funkcije Prikazi i Sakrij koje koristimo u tom slučaju.

OnTriggerEnter metoda se poziva kada bilo koji collider uđe u zonu detekcije ovog objekta. Informacije o collideru spremaju se u varijablu drugi tipa Collider.

Zatim provjeravamo je li collider koji je ušao u detekciju sadrži komponentu Player. Iako se na vozilo mogao dodati jednostavni box collider, u ovom slučaju se koristi unaprijed određeni detaljniji collider uključen u model vozila koji koristimo. Ovaj model vozila sadrži i collidere za kotače koji se mogu koristiti u raznim slučajevima kao za bolju simulaciju i precizniju detekciju, ali ih u ovom slučaju ne koristimo. S obzirom da za detekciju koristimo collider vozila a ne samo vozilo, a model vozila je takav da se collider nalazi kao dijete vozilu (Slika 5.).

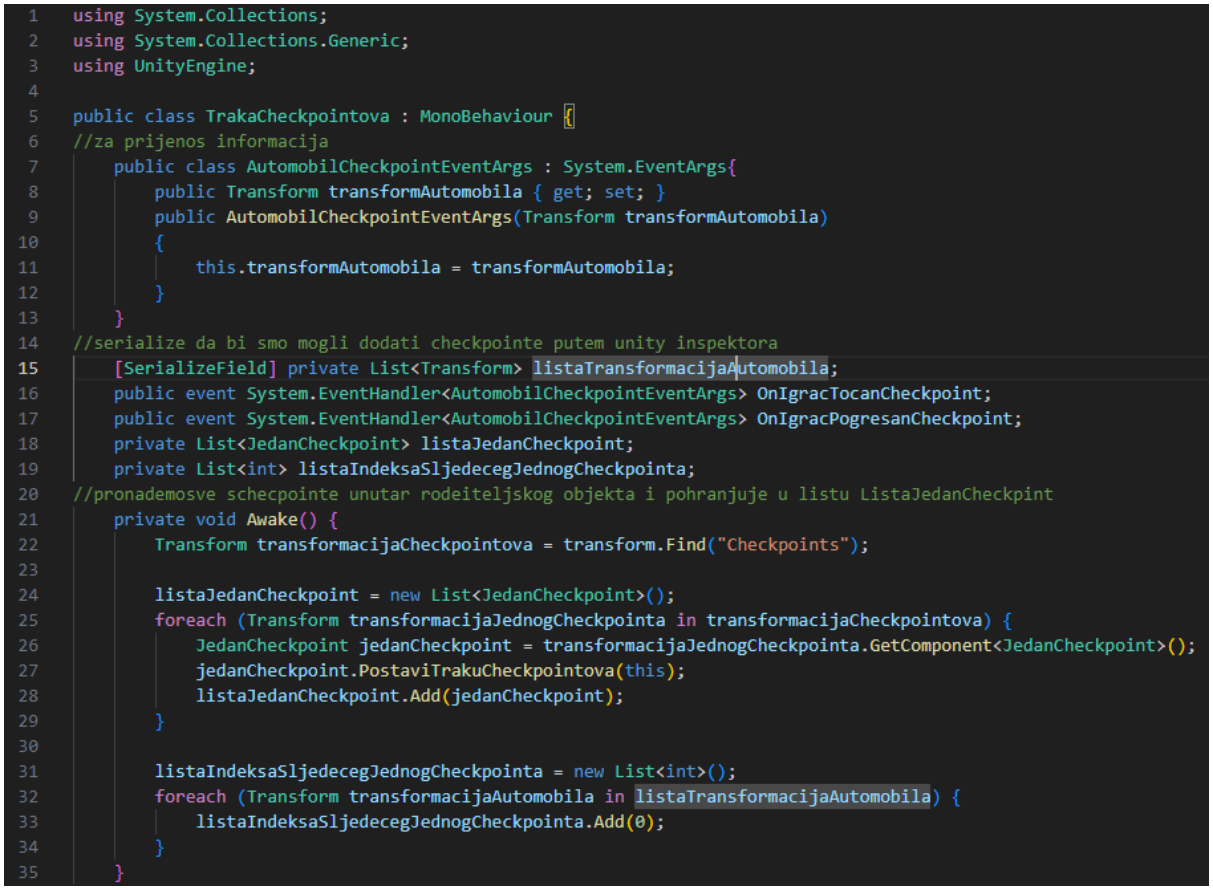


Slika 5. Prikaz strukture modela vozila

Player skriptu vežemo na collider. Ova skripta može biti i prazna jer je koristimo samo za funkciju TryGetComponent. Ako je skripta Player pronađena kao komponenta pozivamo AutomobilKrozCheckpoint metodu trakaCheckpointova komponente i prosljeđujemo referencu ovog checkpointa i transforma koji je ušao u checkpoint. PostaviTrakuCheckpointova koristimo za dodavanje checkpointa.

Možemo dodati više checkpointa tako da ih dupliciramo i odvlačimo po stazi gdje želimo. Više checkpointa organiziramo u prazan objekt Checkpoints.

Zatim nam je potrebna opcija da registriramo više checkpointa, a ne samo jedan. Za to nam je potrebna skripta TrakaCheckpointova (Slika 6. i 7.) koju vežemo na road element.

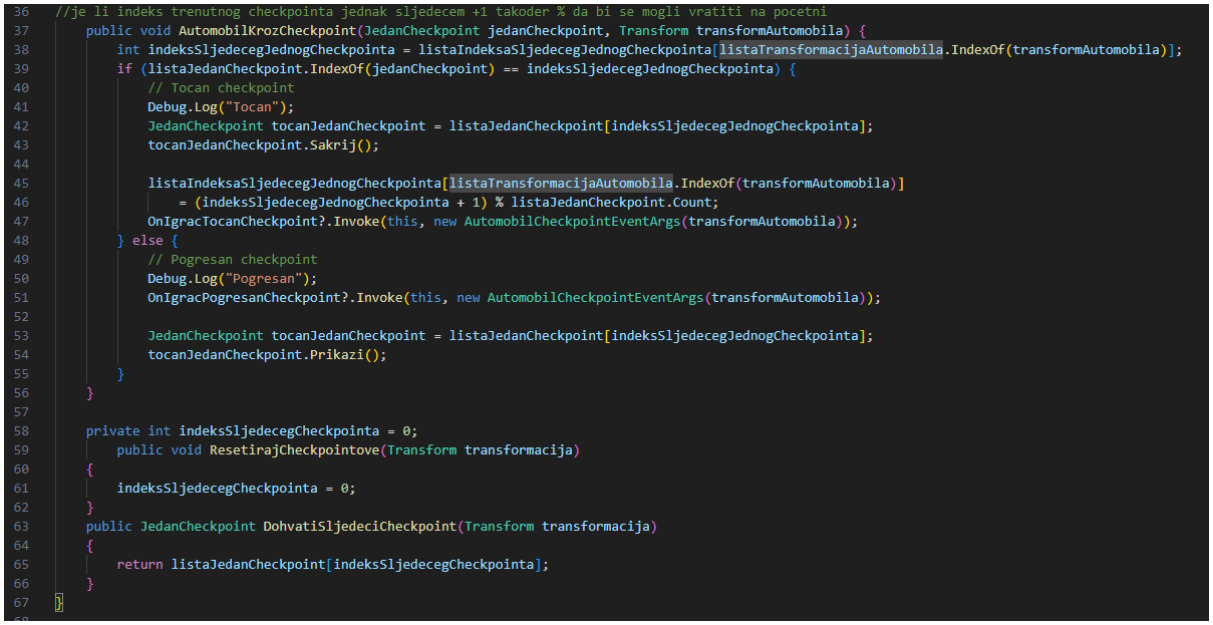


```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class TrakaCheckpointova : MonoBehaviour {
//za prijenos informacija
    public class AutomobilCheckpointEventArgs : System.EventArgs{
        public Transform transformAutomobila { get; set; }
        public AutomobilCheckpointEventArgs(Transform transformAutomobila)
        {
            this.transformAutomobila = transformAutomobila;
        }
    }
//serialize da bi smo mogli dodati checkpointe putem unity inspektora
    [SerializeField] private List<Transform> listaTransformacijaAutomobila;
    public event System.EventHandler<AutomobilCheckpointEventArgs> OnIgracTocanCheckpoint;
    public event System.EventHandler<AutomobilCheckpointEventArgs> OnIgracPogresanCheckpoint;
    private List<JedanCheckpoint> listaJedanCheckpoint;
    private List<int> listaIndeksaSljedecegJednogCheckpointa;
//pronademosve schecpointe unutar rodeiteljskog objekta i pohranjuje u listu ListaJedanCheckpint
    private void Awake() {
        Transform transformacijaCheckpointova = transform.Find("Checkpoints");

        listaJedanCheckpoint = new List<JedanCheckpoint>();
        foreach (Transform transformacijaJednogCheckpointa in transformacijaCheckpointova) {
            JedanCheckpoint jedanCheckpoint = transformacijaJednogCheckpointa.GetComponent<JedanCheckpoint>();
            jedanCheckpoint.PostaviTrakuCheckpointova(this);
            listaJedanCheckpoint.Add(jedanCheckpoint);
        }

        listaIndeksaSljedecegJednogCheckpointa = new List<int>();
        foreach (Transform transformacijaAutomobila in listaTransformacijaAutomobila) {
            listaIndeksaSljedecegJednogCheckpointa.Add(0);
        }
    }
```

Slika 6. Prikaz skripte TrakaCheckpointova

TrakaCheckpointova

Definira se klasa AutomobilCheckpointEventArgs koja služi za prijenos informacija o checkpoint događajima. Zatim definiramo OnIgracTocanCheckpoint i OnIgracPogresanCheckpoint kao događaje za obradu interakcije s checkpointom. U funkciji Awake pronalazimo podređene objekte s imenom „Checkpoints“ te prolazimo kroz petlju gdje postavljamo reference i dodajemo checkpointe na listu.



```
//je li indeks trenutnog checkpointa jednak sljedecem +1 takoder % da bi se mogli vratiti na pocetni
    public void AutomobilKrozCheckpoint(JedanCheckpoint jedanCheckpoint, Transform transformAutomobila) {
        int indeksSljedecegJednogCheckpointa = listaIndeksaSljedecegJednogCheckpointa[listaTransformacijaAutomobila.IndexOf(transformAutomobila)];
        if (listaJedanCheckpoint.IndexOf(jedanCheckpoint) == indeksSljedecegJednogCheckpointa) {
            // Tocan checkpoint
            Debug.Log("Tocan");
            JedanCheckpoint tocanJedanCheckpoint = listaJedanCheckpoint[indeksSljedecegJednogCheckpointa];
            tocanJedanCheckpoint.Sakrij();

            listaIndeksaSljedecegJednogCheckpointa[listaTransformacijaAutomobila.IndexOf(transformAutomobila)]
                = (indeksSljedecegJednogCheckpointa + 1) % listaJedanCheckpoint.Count;
            OnIgracTocanCheckpoint?.Invoke(this, new AutomobilCheckpointEventArgs(transformAutomobila));
        } else {
            // Pogresan checkpoint
            Debug.Log("Pogresan");
            OnIgracPogresanCheckpoint?.Invoke(this, new AutomobilCheckpointEventArgs(transformAutomobila));

            JedanCheckpoint tocanJedanCheckpoint = listaJedanCheckpoint[indeksSljedecegJednogCheckpointa];
            tocanJedanCheckpoint.Prikazi();
        }
    }

    private int indeksSljedecegCheckpointa = 0;
        public void ResetirajCheckpointove(Transform transformacija)
    {
        indeksSljedecegCheckpointa = 0;
    }
    public JedanCheckpoint DohvatiSljedeciCheckpoint(Transform transformacija)
    {
        return listaJedanCheckpoint[indeksSljedecegCheckpointa];
    }
}
```

Slika 7. Prikaz skripte TrakaCheckpointova nastavak

U funkciji AutomobilKrozCheckpoint provjeravamo je li uneseni checkpoint odgovara sljedećem i ako da onda ažuriramo sljedeći checkpoint na checkpoint+1, a ako je to zadnji checkpoint onda se vraćamo na prvi checkpoint. Zatim pozivamo događaj za točan checkpoint. Ako uneseni checkpoint ne odgovara sljedećem onda pozivamo događaj za krivi checkpoint.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class VozacAutomobila : MonoBehaviour
{
    private float brzinaKretanja = 50.0f;
    private float brzinaOkretanja = 50.0f;
    // Update jednom po frejmu
    void Update()
    {
        float kolicinaNaprijed = Input.GetAxis("Vertical");
        float kolicinaOkretanja = Input.GetAxis("Horizontal");

        // je li se vozi prije nego se okrece
        if (Mathf.Abs(kolicinaNaprijed) > 0.01f)
        {
            PostaviUlaze(kolicinaNaprijed, kolicinaOkretanja);
        }
        else
        {
            // ako auto miruje ne dopusti okretanje
            PostaviUlaze(kolicinaNaprijed, 0f);
        }
    }
    public void PostaviUlaze(float kolicinaNaprijed, float kolicinaOkretanja)
    {
        transform.Translate(Vector3.forward * kolicinaNaprijed * brzinaKretanja * Time.deltaTime);
        transform.Rotate(Vector3.up, kolicinaOkretanja * brzinaOkretanja * Time.deltaTime);
    }
    public void PotpunoZaustavi()
    {
        PostaviUlaze(0f, 0f);
    }
}
```

Slika 8. Prikaz skripte VozacAutomobila

Kontrola vozila

Pod kontrolom vozila podrazumijevamo sposobnost igrača da upravlja kretanjem vozila unutar igre. Bitni elementi kontrole vozila su kretanje i okretanje koje možemo odrediti u skripti. Također je bitno postaviti mogućnost da igrač kroz različite ulaze kontrolira vozilo.

VozacAutomobila

Za kontrolu kretanja vozila koristimo skriptu VozacAutomobila (Slika 8.) koju vežemo za model vozila. Da smo skriptu preimenovali u Player i koristili na collideru ustvari bi vozili samo collider. Zato koristimo skriptu VozacAutomobila na vozilu i skriptu Player na collideru. U skripti se prvo definira brzina kretanja i okretanja po želji. Metoda Update se poziva na svakom frejmu te prikuplja ulazne podatke. Zatim u funkcji void Update() postavljamo ulazne podatke na

temelju osi. PostaviUlaze metoda postavlja ulazne podatke za kretanje vozila. Pomicanje se obavlja pomoću transform.Translate i transform.Rotate funkcije. Potpuno zaustavi postavlja ulazne podatke na 0 tako da bi se vozilo potpuno zaustavilo po potrebi.

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

public class AgentVozacAutomobila : Agent {//nasljeduje iz ML_agenta
//trakacheckpointa i spawn pozicija
    [SerializeField] private TrakaCheckpointova trakaCheckpointova;
    [SerializeField] private Transform pozicijaSpawnanja;
    private VozacAutomobila vozacAutomobila;

    private void Awake() {
        vozacAutomobila = GetComponent<VozacAutomobila>();
    }

    private void Start()
    {
        trakaCheckpointova.OnIgracTocanCheckpoint += TrakaCheckpointova_OnIgracTocanCheckpoint;
        trakaCheckpointova.OnIgracPogresanCheckpoint += TrakaCheckpointova_OnIgracPogresanCheckpoint;
    }

    private void TrakaCheckpointova_OnIgracPogresanCheckpoint(object sender, EventArgs e)
    {
        if (e != null && (object)sender == (object)trakaCheckpointova)
        {
            AddReward(-1f);
        }
    }

    private void TrakaCheckpointova_OnIgracTocanCheckpoint(object sender, EventArgs e)
    {
        if (e != null && (object)sender == (object)trakaCheckpointova)
        {
            AddReward(1f);
        }
    }
```

Slika 9. Prikaz skripte AgentVozacAutomobila (1/3)

AgentVozacAutomobila

Skripta AgentVozacAutomobila (Slika 9. 10. i 11.) predstavlja agenta koji se koristi za treniranje pomoću ML-Agents framework-a. trakaCheckpointova je referenca na komponentu TrakaChecpointova koja se koristi za praćenje checkpointa. PozicijaSpawnanja označava mjesto gdje će se vozilo stovriti na početku svake epizode. Metoda Start postavlja slušatelja događaja iz komponente TrakaCheckpointova. Sljedeće metode

TrakaCheckpointova_OnIgracPogresanCheckpoint i TrakaCheckpointova_OnIgracTocanCheckpoint definiraju nagrade za agenta. Ako je agent prošao kroz ispravan checkpoint dodaje mu se pozitivna nagrada, a ako je prošao kroz krivi dodaje se negativna nagrada. U ovom slučaju za checkpointe koristimo vrijednost nagrade od 1 što predstavlja veliku nagradu ili kaznu. Razlika između nagrada i kazni je vrlo bitna jer pomoću nje agenti oblikuju svoje ponašanje. Agenti uvijek pokušavaju maksimizirati ukupnu nagradu, iako u posebnim slučajevima agenti mogu izbjegavati rizik i odabrati sigurne akcije koje su pojedinim primjenama nepoželjne.

```
40  //random spawn uz vektor i da je usmjeren prema dobrom checkpointu
41      public override void OnEpisodeBegin() {
42          transform.position = pozicijaSpawnanja.position + new Vector3(UnityEngine.Random.Range(-5f, +5f), 0, UnityEngine.Random.Range(-5f, +5f));
43          transform.forward = pozicijaSpawnanja.forward;
44          trakaCheckpointova.ResetirajCheckpointove(transform);
45          vozacAutomobila.PotpunoZaustavi();
46      }
47  //promatramo kut izmedu smjera vozila i checkpointa
48      public override void CollectObservations(VectorSensor senzor) {
49          Vector3 smjerCheckpointa = trakaCheckpointova.DohvatiSljedeciCheckpoint(transform).transform.forward;
50          float usmjerenostDot = Vector3.Dot(transform.forward, smjerCheckpointa);
51          senzor.AddObservation(usmjerenostDot);
52      }
53  //kako reagira na akcije
54      public override void OnActionReceived(ActionBuffers akcije) {
55          float kolicinaNaprijed = 0f;
56          float kolicinaOkretanja = 0f;
57
58          switch (akcije.DiscreteActions[0]) {
59              case 0: kolicinaNaprijed =  0f; break;
60              case 1: kolicinaNaprijed = +1f; break;
61              case 2: kolicinaNaprijed = -1f; break;
62          }
63          switch (akcije.DiscreteActions[1]) {
64              case 0: kolicinaOkretanja =  0f; break;
65              case 1: kolicinaOkretanja = +1f; break;
66              case 2: kolicinaOkretanja = -1f; break;
67          }
68  //salje ulaz
69          vozacAutomobila.PostaviUlaze(kolicinaNaprijed, kolicinaOkretanja);
70      }
71
72      public override void Heuristic(in ActionBuffers akcijeOut) {
73          int akcijaNaprijed = 0;
74          if (Input.GetKey(KeyCode.UpArrow)) akcijaNaprijed = 1;
75          if (Input.GetKey(KeyCode.DownArrow)) akcijaNaprijed = 2;
76
77          int akcijaOkretanja = 0;
78          if (Input.GetKey(KeyCode.RightArrow)) akcijaOkretanja = 1;
79          if (Input.GetKey(KeyCode.LeftArrow)) akcijaOkretanja = 2;
80
81          ActionSegment<int> diskretneAkcije = akcijeOut.DiscreteActions;
82          diskretneAkcije[0] = akcijaNaprijed;
83          diskretneAkcije[1] = akcijaOkretanja;
84      }
```

Slika 10. Prikaz skripte AgentVozacAutomobila (2/3)

OnEpisodeBegin postavlja mjesto i rotaciju automobila na terenu te poništava sve checkpointe na stazi. CollectObservations definira kako će se skupljati podaci iz okoline. Za vožnju vozila promatramo kut između smjera vozila i sljedećeg checkpointa. OnActionReceived pomaže pri reagiranju na akcije. Ovisno o ulazu poziva se PostaviUlaze da bi se vozilo kretalo. Heuristic metodu koristimo po savjetu iz dokumentacije, naime Heuristic nam omogućava da upravljamo vozilom bez korištenja umjetne inteligencije.

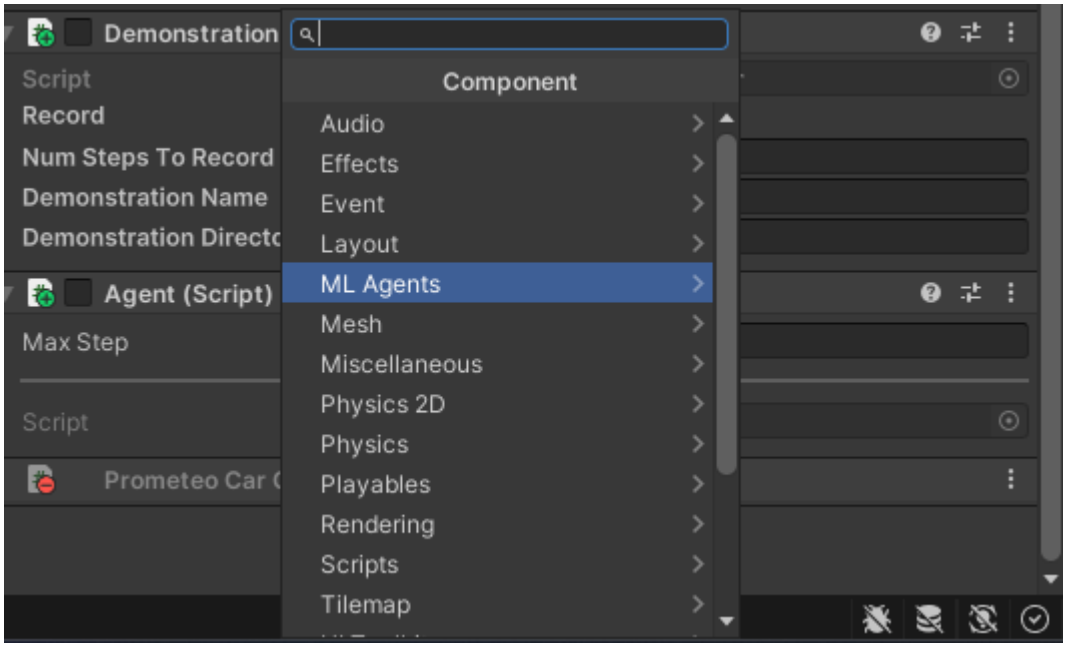
```
85  //ako se sudari sa zidom veca negativna nagrada
86      private void OnCollisionEnter(Collision kolizija) {
87          if (kolizija.gameObject.TryGetComponent<Wall>(out Wall wall)) {
88              // Udario u zid
89              AddReward(-0.5f);
90              //KrajEpizode();
91          }
92      }
93  //ako ostaje sudaren sa zidom manja negativna nagrada
94      private void OnCollisionStay(Collision kolizija) {
95          if (kolizija.gameObject.TryGetComponent<Wall>(out Wall wall)) {
96              // Udario u zid
97              AddReward(-0.1f);
98          }
99
100     }
101
102 }
```

Slika 11. Prikaz skripte AgentVozacAutomobila (3/3)

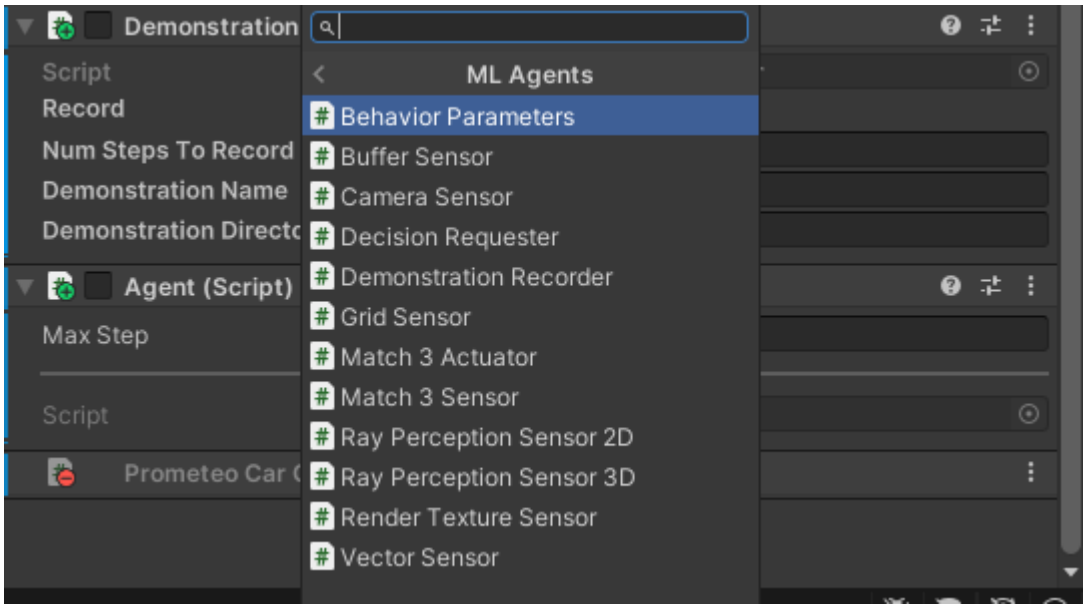
Također dodajemo I OnCollisionEnter i OnCollisionStay gdje dodajemo negativne nagrade agentu ako dodirne zid ili ako duže vremena dodiruje zid. Agentu dodajemo negativnu nagradu od 0.5 u slučaju da udari zid odnosno izađe sa staze. Također u slučaju da agent udari zid, ne želimo da agent dobije negativnu nagradu i nastavi voziti po zidu odnosno rubu staze pa dodajemo i negativnu nagradu za zadržavanje agenta u zidu od 0.1. Ovo omogućava da agent prepozna da u slučaju udara u zid što prije izađe iz zone detekcije. U slučaju da ne koristimo ove dvije funkcije UI ima tendenciju ostanka na istom mjestu jer ne želi dobiti negativne nagrade.

ML-Agents korištenje

Da bismo koristili ML-Agente potrebno je prvo dodati komponente na željene objekte u inspektoru (Slika 12. i 13.).



Slika 12. Prikaz odabira komponente u inspektoru



Slika 13. Prikaz opcija prilikom odabira ML-Agents komponente

Nakon što konfiguriramo komponente po želji, možemo pristupiti Python konzoli gdje upisujemo komandu mlagents-learn --run-id=test (Slika 14.) , zatim nam konzola javlja da sluša akcije na određenom portu i da smo spremni započeti učenje. Učenje započinjemo pritiskom gumba play u Unity editoru.

Slika 14. Prikaz korištenja ML-Agents modula u Anacondi

U ovom slučaju ovu komandu ćemo koristiti pet puta s različitim –run-id parametrom. Time stvaramo pet različitih mozgova za agente. Svaki mozak smo trenirali sat vremena te možemo primijetiti da je u dva slučaja agent uspio doći do trećeg checkpointa, dok 3 slučaja nikad nije uspio doći do prvog checkpointa. Ako nikad ne dođe do checkpointa, ne može ni znati da mu taj objekt daje nagradu. Prilikom ovako kratkog učenja agent niti jedan agent nije uspio završiti stazu. Možemo primijetiti da prilikom podržanog učenja agent uči vrlo sporo te da će teško doći do prvog checkpointa. Zbog toga je potrebno trenirati agenta iznimno dugo odnosno potrebno mu je nekoliko stotina tisuća koraka za učenje. Zato možemo koristiti imitacijsko učenje.

Imitacijsko učenje možemo podesiti u konfiguracijskoj datoteci koju smo definirali. U ovom slučaju to je vjezba8.yaml. Za imitacijsko učenje potrebna je demo datoteka koju možemo snimiti pomoću prethodno spomenute komponente Demonstration Recorder u inspektoru. Unutar yaml datoteke možemo dodati opcije za GAIL (Generative Adversarial Imitation Learning) i behavioral cloning [14].

GAIL je pristup koji nagrađuje agenta za ponašanje slično demonstracijama. Može se koristiti i bez nagrada iz okoline, te je pogodan za scenarije gdje nema signala nagrade iz okoline.

Behavioral Cloning (BC) trenira agenta da oponaša akcije prikazane u demonstraciji. Ovakav pristup najbolje funkcionira kada imamo demonstraciju dostupnu za

svako moguće stanje koje agent može doživjeti. BC nikada ne može biti bolji od demonstracije, zato se često koristi zajedno s GAIL i extrinsic.

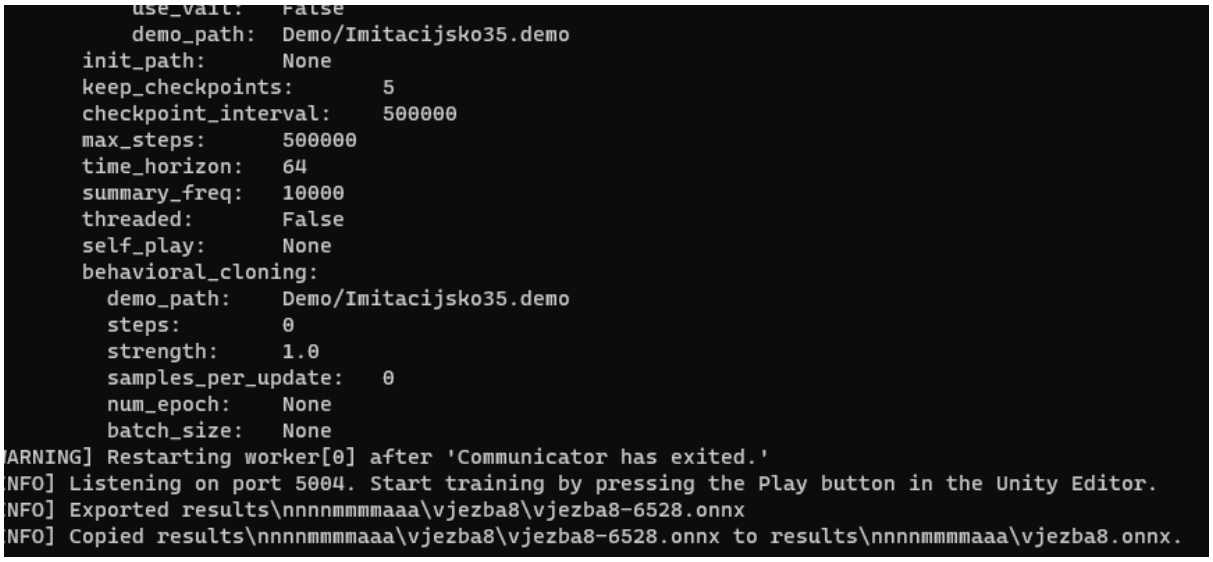
S obzirom na to, na početku postavimo jačinu GAIL i Behavioral Cloning na 1 te extrinsic na 0.1 (Slika 15.) Time smo agenta postavili da uči iz demonstracija, a što manje pomoću extrinsic nagrada. S prethodno navedenom komandom kao i u podržanom učenju možemo pokrenuti nova učenja, te ponovno koristimo pet agenata koji uče u razdobljima od sat vremena. Ovaj put primjećujemo da u zadanom razdoblju svih pet agenata uspješno prođe kroz prvi zavoj, od čega dva agenta uspješno završe stazu. Iako su dva agenta uspjela završiti stazu, tijekom vožnje koriste puno bespotrebnih unosa smjera što se odražava trzanjem vozila odnosno malim skretanjima vozila u raznim smjerovima. S obzirom na ML-Agents dokumentaciju ovaj problem bi mogli riješiti nastavkom treniranja agenata kojima smo zadovoljni do željene glatkoće vožnje.

Kada smo zadovoljni s znanjem agenta možemo smanjiti vrijednosti Gail i BC opcije te povisiti vrijednost strength opcije unutar extrinsic da bismo omogućili UI da postane i bolji od demonstracije. Potom možemo ponovno s komandom pokrenuti učenje agenta. Ovaj put prilikom učenja UI pokušava maksimizirati svoju razliku nagrada, odnosno pokušava što brže proći kroz checkpointe da bi ostvario što veći broj nagrada u zadanom vremenskom intervalu.

```yaml
behaviors:
  vjezba8:
    trainer_type: ppo
    hyperparameters:
      batch_size: 256
      buffer_size: 10240
      learning_rate: 0.0003
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear

    network_settings:
      normalize: false
      memory_size: 128
      hidden_units: 128
      num_layers: 2

    reward_signals:
      extrinsic:
        strength: 0.1
        gamma: 0.99
      gail:
        strength: 1.0
        gamma: 0.99
        demo_path: Demo/Imitacijsko35.demo
      behavioral_cloning:
        strength: 1.0
        demo_path: Demo/Imitacijsko35.demo
    max_steps: 500000
    time_horizon: 64
    summary_freq: 10000

```

Slika 15. Prikaz konfiguracijske datoteke vjezba8.yaml

U ovom slučaju prilikom treniranja jednog agenta u razdoblju od sat vremena ne vidimo značajan pomak u sposobnosti agenta da kontrolira vozilo što odražava nedovoljnu istreniranost agenta u prethodnom postupku. Da imamo dovoljno vremena istrenirati agenta mogli bi smo primijetiti promjenu u vožnji agenta na način da će agent pokušati maksimizirati nagradu na uštrb praćenja demonstracija. Odnosno agent će pokušati odrediti svoje akcije tako da vozi što brže i preciznije, a može postati bolji i od demonstracijske datoteke tj. pravog igrača. Ovim postupkom možemo generirati agente za različite razine vožnje od početnika do profesionalca ovisno koliko ih treniramo.



Slika 16. Prikaz izlaza konzole nakon treniranja

Ovakav postupak producira .onnx datoteku (Slika 16.) koja je ustvari mozak agenta. Što više treniramo datoteku to će se agent bolje ponašati u svom okruženju. Mozak agenta možemo dodati unutar Behavior parameters komponente player objekta. S time smo efektivno stvorili UI.

Dovršetak igre

Igru dovršavamo dodavanjem izbornika. Za korištenje izbornika potrebno je definirati i druge scene koje ćemo koristiti unutar igre. S obzirom da imamo pripremljenu scenu, možemo je duplicirati dva puta. Originalna scena će nam koristiti za igranje te ćemo u njoj isključiti skriptu AgentVozacAutomobila i uključiti VozacAutomobila. Druge dvije scene će nam koristiti za demonstraciju

podržanog i imitacijskog učenja te ćemo kod njih isključiti skriptu VozacAutomobila, a uključiti AgentVozacAutomobila. Player objektima dodajemo model mozga koji želimo, a koji smo generirali prethodnim postupcima. Na posljetku izrađujemo glavni izbornik pomoću opcija Canvas i Buttons. Akcije tipki unutar izbornika postavljamo pomoći inspektora i Main Menu skripte.

Poteškoće pri izradi

Prilikom izrade ove igre naišli smo na brojne poteškoće. Prva je bila izrada ceste. S obzirom da nismo htjeli plaćati za assete, koristili smo besplatnu verziju EasyRoads3D koja omogućava izmjenu staza samo u jednoj sceni. Ukoliko želimo mijenjati stazu za druge scene potrebna je nadoplata. Također problem je nastao i pri izradi zidova koje UI treba izbjegavati. Dodavanje zidova je bilo dugotrajno i zahtjevno s obzirom da se svaki zid dodavao zasebno. Uslijedili su i problemi s postavljanjem ML-Agents modula. Naime modul zahtjeva stariju verziju Pytorcha nego što je trenutna, također modul ML-Agents koji je dostupan u Unity packet manageru nije najnoviji dostupni nego smo ga morali instalirati manualno. Prilikom treniranja nastao je problem s obzirom na dimenziju vremena potrebnu za pravilno treniranje. Treniranje je bilo dugotrajno i zahtijevalo je veliku potrošnju računalnih resursa, a rezultati nisu blizu početnih očekivanja.

Zaključak

Cilj ovog rada bio je istražiti primjenu ML-Agents modula u kontekstu razvoja računalne igre. U radu smo opisali postavljanje i korištenje ML-Agents modula u sučelju Unitya. Također opisali smo i skripte koje su potrebne da bi ML-Agents modul funkcionirao ispravno i da bi igra bila funkcionalna, a naveli smo i probleme na koje smo naišli prilikom izrade igre.

Iako smo naišli na probleme pri izradi, bilo je zanimljivo naučiti više o teoriji i primjeni ML-Agents modula. S obzirom na napredak tehnologije umjetne inteligencije, za očekivati je da će se vrijeme treniranja agenata skraćivati. Brojne nove tehnologije će nam to omogućiti poput CUDA platforme [15] i razvijanja procesora isključivo za primjene u umjetnoj inteligenciji [16]. UI je evoluirao tijekom godina te je za očekivati da će i nastaviti evoluirati. Strojno učenje omogućava igrama da postanu sve dinamičnije i realističnije i samo je pitanje na koje sve druge načine nam može koristiti.

Popis slika

Reference

[1]https://www.statista.com/topics/868/video-games/#topicOverview [12.09.2023]

[2] https://www.ucumberlands.edu/blog/future-gaming-industry [12.09.2023]

[3] https://en.wikipedia.org/wiki/Microtransaction [14.09.2023]

[4] https://twitchtracker.com/statistics [14.09.2023]

[5] https://en.wikipedia.org/wiki/Nimrod_(computer) [14.09.2023]

[6] https://en.wikipedia.org/wiki/Arcade_game [14.09.2023]

[7] https://en.wikipedia.org/wiki/Space_Invaders [14.09.2023]

[8] https://www.forbes.com/sites/paultassi/2023/02/16/chatgpt-is-coming-to-video-games-god-help-us-all/?sh=12d140f41d48 [14.09.2023]

[9] https://en.wikipedia.org/wiki/Pole_Position [14.09.2023]

[10] https://docs.unity.com/ [15.09.2023]

[11] https://github.com/Unity-Technologies/ml-agents [15.09.2023]

[12] Artificial Intelligence in Games, Paul Roberts, CRC Press, prvo izdanje 2023

[13] https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Installation.md [15.09.2023]

[14] https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/ [15.09.2023]

[15] https://en.wikipedia.org/wiki/CUDA [15.09.2023]

[16] https://www.nvidia.com/en-us/data-center/l4/ [15.09.2023]