Primjena podržanog učenja u računalnoj igri utrke

Karadža, Juraj

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 750

PRIMJENA PODRŽANOG UČENJA U RAČUNALNOJ IGRI UTRKE

Juraj Karadža

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 750

PRIMJENA PODRŽANOG UČENJA U RAČUNALNOJ IGRI UTRKE

Juraj Karadža

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 30. rujna 2024.

DIPLOMSKI ZADATAK br. 750

Pristupnik:	**Juraj Karadža (0036519620)**
Studij:	Računarstvo
Profil:	Znanost o mrežama
Mentor:	prof. dr. sc. Igor Sunday Pandžić

Zadatak: **Primjena podržanog učenja u računalnoj igri utrke**

Opis zadatka:

Podržano učenje je vrsta strojnog učenja u kojoj algoritam postupno uči kroz niz pokušaja i pogrešaka, pri čemu dobiva pozitivne signale (nagradu) za željena ponašanja, a negativne signale (kaznu) za neželjena ponašanja. Algoritam nastoji maksimizirati ukupnu nagradu i tako postupno uči željeno ponašanje. Ovakvo učenje se često upotrebljava za autonomne agente u računalnim igrama. Utrke su popularna vrsta računalnih igara u kojima igrač upravlja automobilom u simulaciji utrke koja može sadržavati i prepreke i druge elemente. Suigrači pritom mogu biti autonomni agenti ili drugi stvarni igrači. Cilj ovog zadatka je osmisliti sustav u kojem korisnik oblikuje razine igre uključujući staze za utrku, eventualne prepreke, nagrade i druge elemente, a algoritam podržanog učenja igra tako postavljenu igru nastojeći skupiti što više bodova. Vaš zadatak je osmisliti, implementirati i testirati ovakav sustav, uključujući i mogućnost utrke više agenata.

Rok za predaju rada: 14. veljače 2025.

Zahvaljujem se Prof. Dr. Sc. Igoru Pandžiću na mentorstvu i strpljenju. Također, zahvaljujem se svojim roditeljima, Vjekoslavu i Ivanki, na podršci i razumijevanju tijekom cijelog mog obrazovanja.

Posebno hvala i mojoj Dori, što je ovaj period mog života ispunila lijepim uspomenama.

Sadržaj

1. Uvod

Razvoj strojnog učenja donio je revoluciju u mnogim područjima primjene, od medicinske dijagnostike i autonomnih vozila do analize podataka i računalnih igara. Jedno od područja strojnog učenja je podržano učenje (engl. Reinforcement Learning, RL), koje omogućuje agentima da uče optimalna ponašanja kroz interakciju s okolinom. Ključna značajka podržanog učenja je sposobnost algoritma da samostalno istražuje okruženje, donosi odluke na temelju povratnih informacija te se prilagođava kako bi maksimizirao ukupnu nagradu. Ova metoda pronalazi široku primjenu u situacijama gdje je potrebno donositi kompleksne odluke u dinamičnim okruženjima, uključujući računalne igre.

Računalne igre predstavljaju savršen okvir za istraživanje podržanog učenja zbog svoje strukture i mogućnosti simulacije raznih scenarija. Utrke su posebno popularan žanr igara koji pruža izazovno okruženje za razvoj i testiranje autonomnih agenata. Utrke uključuju različite elemente poput brzine, strategije, reakcije na prepreke te interakcije s drugim sudionicima, što ih čini idealnim poligonom za primjenu podržanog učenja. U takvim scenarijima, agenti mogu učiti kako upravljati vozilom, izbjegavati prepreke, koristiti optimalne putanje i donositi odluke u realnom vremenu.

Cilj ovog diplomskog rada je osmisliti sustav koji korisnicima omogućuje kreiranje vlastitih razina igre, uključujući definiranje staza za utrku i postavljanje prepreka. Nakon toga, algoritam podržanog učenja igra tako dizajniranu igru s ciljem postizanja što boljih rezultata, odnosno prikupljanja maksimalnog broja (internih) bodova. Sustav također uključuje mogućnost utrke više agenata, omogućujući simulaciju kompetitivnih scenarija u kojima igrač sudjeluje upravljajući vlastito vozilo.

Implementacija ovog sustava obuhvaća nekoliko ključnih koraka. Prvi korak je razvoj osnovne strukture igre utrke automobila. Ovaj korak uključuje izradu automobila i

staza koje će služiti kao natjecateljska okruženja, uključivanje prepreka koje povećavaju izazovnost igre te osmišljavanje vizualnog identiteta same igre kako bi bila privlačna i intuitivna za korisnike.

Drugi korak je prilagodba sustava za podržano učenje. Ovdje je ključno omogućiti umjetnoj inteligenciji da upravlja vozilima u utrci. To uključuje implementaciju mehanizama nagrađivanja za poželjne akcije, poput odabira optimalnih putanja ili izbjegavanja prepreka, te sustava kažnjavanja za pogreške, poput sudara ili nepravilnih putanja. Na kraju ovog koraka, cilj je imati potpuno istrenirane i funkcionalne AI agente koji se znaju utrkivati na našim stazama.

I treće, potrebno je razviti intuitivno korisničko sučelje koje omogućuje igraču kreiranje razina igre i podešavanje postavki same igre. To će uključivati odabir staze, količinu prepreka i broj neprijateljskih automobila.

Ova tri koraka čine temelj strukture ovog rada. Svaki korak će biti detaljno razrađen u zasebnim poglavljima, počevši od tehničkih aspekata razvoja igre, preko implementacije podržanog učenja, pa sve do kreiranja korisničkog sučelja. Završno poglavlje rada će predstaviti finalni produkt pod nazivom "Desert Racers".

2. Razvoj Računalnih Igara

Razvoj računalne igre složen je zadatak koji uključuje tehničke i kreativne izazove. Na tehničkoj razini, potrebno je implementirati sustav za vizualizaciju grafike, obradu vanjskih unosa (bilo ljudskih ili strojnim putem) te prikaz promjena koje ti unosi uzrokuju unutar virtualnog svijeta. Osim toga, nužno je osigurati realistične sustave fizike i međudjelovanja objekata unutar tog svijeta, uključujući osvjetljenje, sjene, sudare i kretanje.

S druge strane, tehničke aspekte potrebno je povezati u cjelovito i privlačno iskustvo. Igra mora uspješno uroniti igrača u virtualni svijet koji je dovoljno intuitivan za razumijevanje, a istovremeno dovoljno zanimljiv da potakne želju za sudjelovanjem.

Dok je kreativni aspekt razvoja isključivo odgovornost game developera, tehnički aspekt razvoja značajno olakšavaju specijalizirani alati poznati kao game engines. Ovi sustavi ubrzavaju proces razvoja igara nudeći unaprijed pripremljena rješenja za grafičko renderiranje, simulaciju fizike, upravljanje unosima i druge ključne funkcionalnosti.

Na tržištu postoji više različitih game enginea, no za potrebe ovog rada korišten je Unity.

2.1. Unity - Platforma za Razvoj Računalnih Igara

Unity [1] je višenamjenski game engine koji se koristi za razvoj videoigara, simulacija i interaktivnih aplikacija. Razvijen od strane tvrtke Unity Technologies i prvi put predstavljen 2005. godine, Unity je danas jedan od najpopularnijih alata u industriji razvoja igara. Njegova fleksibilnost, pristupačnost i podrška za više platformi čine ga idealnim rješenjem za širok spektar korisnika, od početnika do iskusnih profesionalaca.

Unity omogućuje razvoj za različite platforme, uključujući računala (Windows, ma-

cOS i Linux), mobilne uređaje (iOS i Android), konzole (PlayStation, Xbox, Nintendo Switch) te uređaje za virtualnu (VR) i proširenu stvarnost (AR). Osim podrške za više platformi, Unity nudi alate za razvoj igara u 2D i 3D okruženju. Njegovi napredni sustavi omogućuju stvaranje složenih projekata, uključujući vizualno impresivne igre s realističnim fizikalnim simulacijama i naprednim animacijama.

Glavni jezik programiranja u Unityju je C#, koji omogućuje visok stupanj prilagodljivosti i proširivosti funkcionalnosti. Unity dolazi s integriranim sustavima, poput sustava za renderiranje grafike, simulaciju fizike, upravljanje zvukom i unosom korisnika. Također, Unity Asset Store, online platforma za razmjenu resursa, omogućuje programerima brzu integraciju gotovih modela, tekstura, zvukova i drugih elemenata.

Unity nije ograničen samo na razvoj igara. Koristi se i u drugim industrijama, uključujući arhitekturu, automobilski sektor, obrazovanje i filmsku industriju, za stvaranje interaktivnih simulacija, vizualizacija i multimedijskih iskustava.

Jedan od glavnih razloga popularnosti Unityja je njegova sposobnost da značajno ubrza proces razvoja. Programerima nudi unaprijed pripremljene sustave koji eliminiraju potrebu za izradom osnovnih funkcionalnosti od nule, čime se omogućuje veći fokus na kreativni aspekt razvoja. Uz intuitivno korisničko sučelje i bogatu dokumentaciju, Unity je pristupačan i početnicima, dok njegove napredne mogućnosti zadovoljavaju zahtjeve profesionalnih programera.

Slika 2.1. Unity logo
[1]

2.2. Razvoj Igre Utrke u Unity-u

U ovom poglavlju ću opisati korake koje sam poduzeo za izradu igre utrke. Fokus je na razvoju temeljnih mehanika koje su ključne za stvaranje funkcionalnog i stabilnog temelja igre.

U ovoj početnoj fazi razvoja, odlučio sam se usmjeriti na osnovne aspekte igre, poput kreiranja trkaće staze, implementacije vozila i osnovnih pravila igre. Cilj nije stvaranje vizualno impresivne igre, već izrada sustava koji omogućuje igrivost i testiranje osnovnih značajki.

Važno je napomenuti da u ovoj fazi ne implementiramo napredne tehnike poput podržanog učenja (AI) za protivničke igrače. Igra je zamišljena kao jednostavno okruženje koje omogućuje istraživanje osnovnih mehanika, poput upravljanja vozilima, interakcije s preprekama i osnovne navigacije kroz stazu. Ove komponente postavljaju temelj za daljnji razvoj i nadogradnju igre u kasnijim fazama.

Također, dizajn i vizualni aspekt igre u ovoj fazi nisu prioritet. Iako će vizualna privlačnost igre biti važan aspekt konačnog proizvoda, trenutačni fokus je na funkcionalnosti i osiguravanju da svi ključni sustavi rade kako je predviđeno. Ovo uključuje osiguranje pravilne fizičke simulacije, glatkog upravljanja vozilom i osnovne interakcije između elemenata igre.

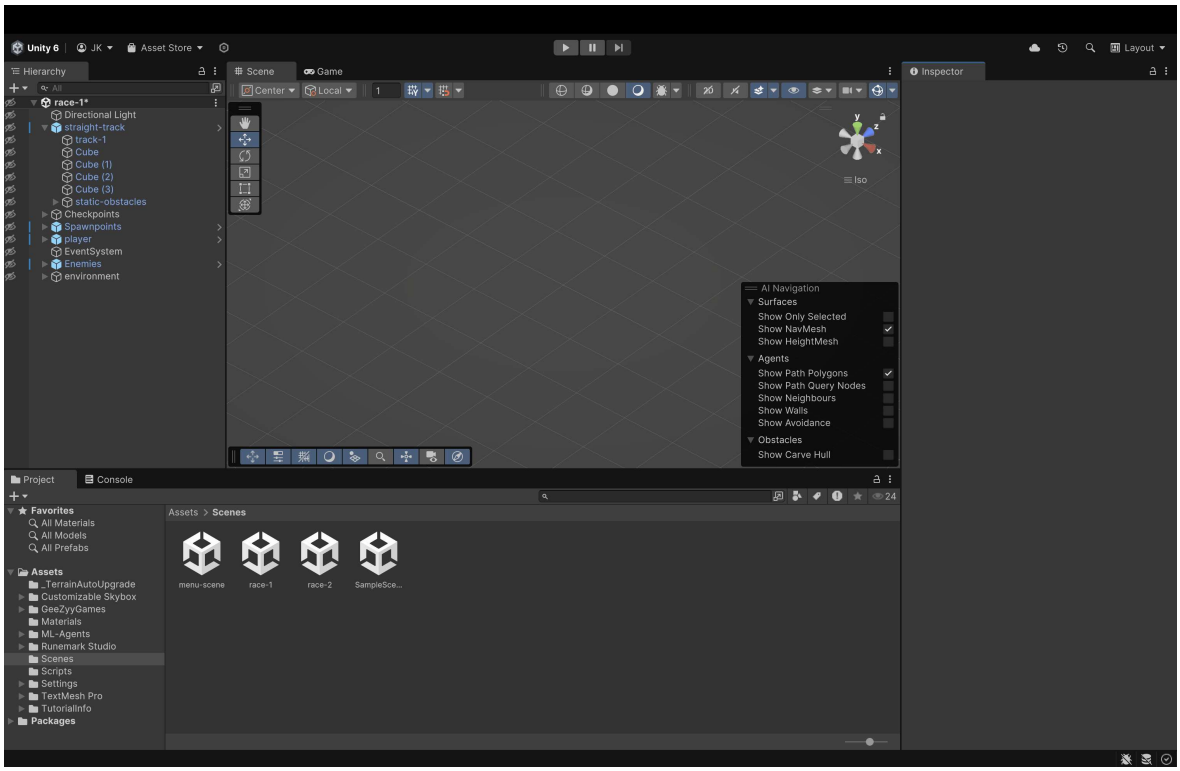
2.2.1. Dizajn i Izrada Trkaće Staze

Započinjemo s praznom scenom unutar Unity okruženja, koja predstavlja početnu točku za izgradnju igre. Na početku se na ekranu prikazuje jednostavno rešetkasto polje koje služi kao referentni prostor za pozicioniranje objekata.

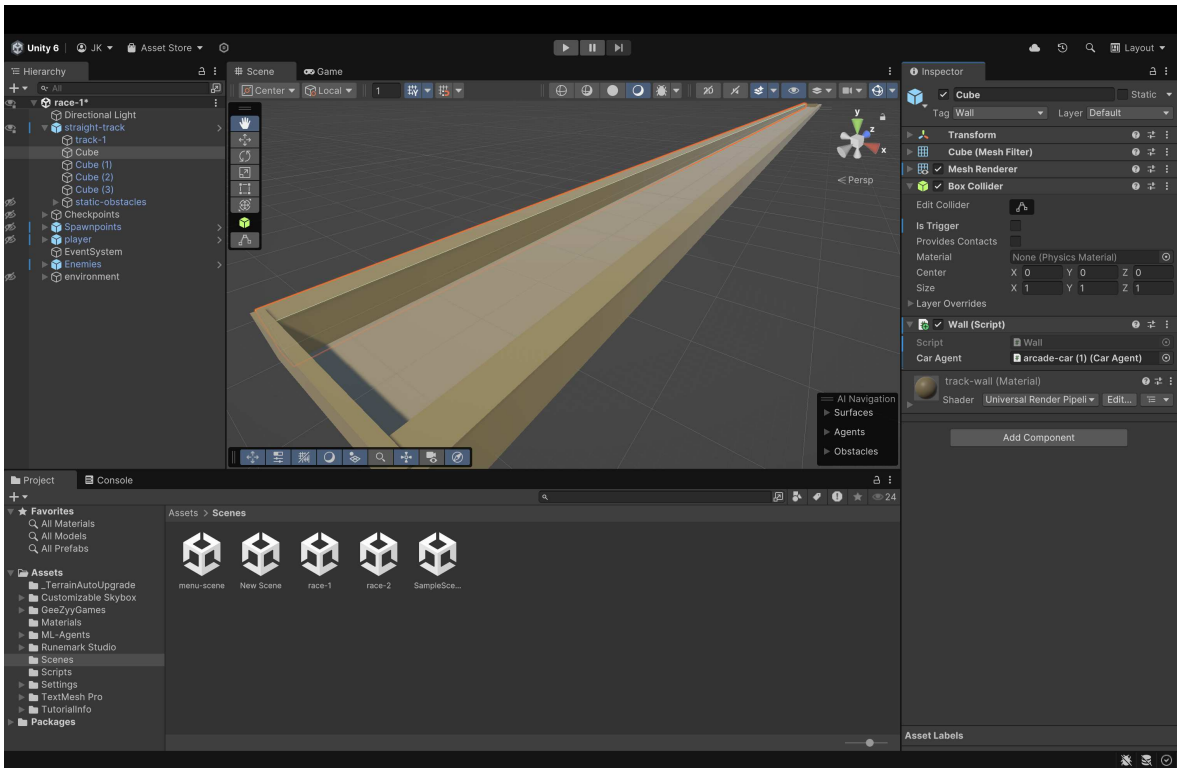
Uz osnovno rešetkasto polje, scena automatski uključuje nekoliko početnih objekata koje Unity generira po defaultu: izvor svjetlosti i kamera. Izvor svjetlosti osigurava osnovno osvjetljenje u sceni, omogućujući da objekti budu vidljivi, a kamera definira pogled igrača na scenu.

Prvu stazu koju želim napraviti je ravna staza. Dodajem prve elemente u scenu, Plane i Cube. Plane služi kao ploha po kojoj će auti voziti, a pomoću Cube objekta oblikujem zidove staze kako auti ne bi otišli s nje. Ovo je vrlo jednostavan proces koji se svodi na konfiguriranje veličine, položaja i oblika kocaka u Transform atributu koji svaki objekt u Unityju posjeduje.

Na sličan način oblikujem i drugu stazu koju želim da bude ovalnog oblika. Pošto je oblik malo kompliciraniji, druga staza se sastoji od više ploha i kocaka od prve staze.

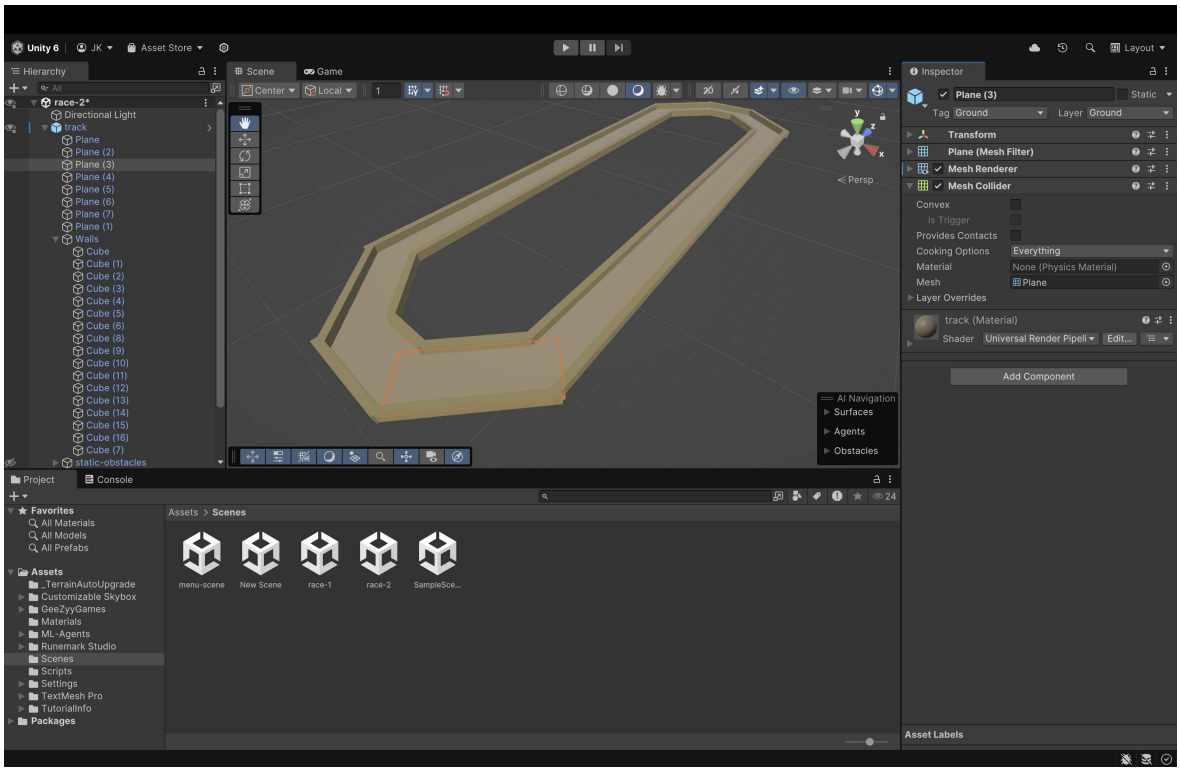


Slika 2.2. Prazna scena



Slika 2.3. Ravna staza

No, i dalje je cijeli proces vrlo jednostavan - postavljanje kocaka, konfiguriranje veličine i oblika te grupiranje.



Slika 2.4. Kružna staza

Svaka staza u igri definirana je kao zasebna scena unutar Unityja, što omogućuje jednostavno upravljanje i organizaciju raznih elemenata igre. Na taj način, svaka staza može biti nezavisno uređena, testirana i prilagođena bez utjecaja na druge dijelove igre.

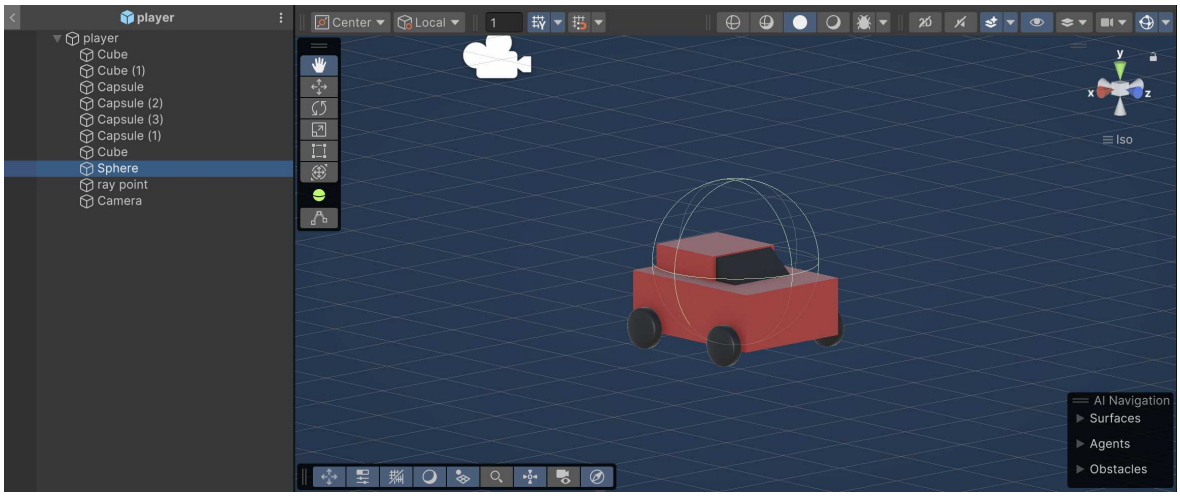
Sve plohe (Plane) i kocke (Cube) koje čine stazu imaju pridružen Collider atribut. Collider je komponenta u Unityju koja omogućuje interakciju objekata s fizičkim okruženjem igre. Ova komponenta definira oblik objekta koji sudjeluje u simulaciji sudara i drugih fizičkih interakcija. Plohe s Colliderom će omogućiti da vozilo ostane na podlozi, sprječavajući njegovo "propadanje" kroz stazu, dok kocke s Colliderom će osigurati da vozilo ne prolazi kroz zidove, već reagira na njih kao fizičku prepreku.

2.2.2. Razvoj Modela i Kontrola Vozila

Svaka igra utrka, osim staza, zahtijeva i neku vrstu utrkivača, u mom slučaju su to automobili. Model automobila je bio jednostavan za složiti. Koristio sam Unity-eve kocke za tijelo automobila, te kapsule za kotače. Na tijelo automobila sam dodao i crveni materijal, a na kotače i kocku koja glumi prednje staklo - crni materijal. Sve objekte sam grupirao zajedno pod prazan objekt zvan “player”. Grupiranje objekata mi omogućuje

pomicanje svih odjednom kao da su jedno tijelo.

Nakon što je model automobila izrađen, sljedeći je korak omogućiti njegovo pokretanje unutar igre. Da bih postavio osnovne osnove kretanja, dodao sam još jedan objekt u strukturu automobila: sferu (Sphere).



Slika 2.5. Automobil

Kao što se može vidjeti na slici, u sredini automobila se nalazi sfera. Sfera nema na sebi mesh (vidljiv materijal), ali ima Collider i RigidBody atribut. Ona služi kao nevidljivo polje koje moj auto okupira u virtualnom svijetu. Makar se ne poklapa u potpunosti s modelom automobila (što će dovoditi do pojave clippinga - kada jedan objekt u igri “ulazi” u drugi), ova sfera će mi olakšati programiranje kretanja automobila. Ideju sa sferom sam vidio u GitHub repozitoriju "Mix and Jam" [2].

Pogledamo li skriptu CarController koja je vezana uz automobil, vidjet ćemo da se automobil pokreće tako što dodajemo silu na RigidBody (sferu). Taj model kretanja je odličan za pravilno kretanje automobila, bez puno logike u koju se mogu zavući bugovi.

No, sfera se kreće tako što se kotrlja, a to nije prirodno kretanje za automobil. Kako bi izbjegli kotrljanje automobila, prilikom pokretanja skripte odvajamo sferu od ostatka automobila. Zatim u Update() funkciji pokrećemo automobil tako što pratimo poziciju sfere.

Listing 2..1: CarController.cs

```
1 void Start()
2 {
```

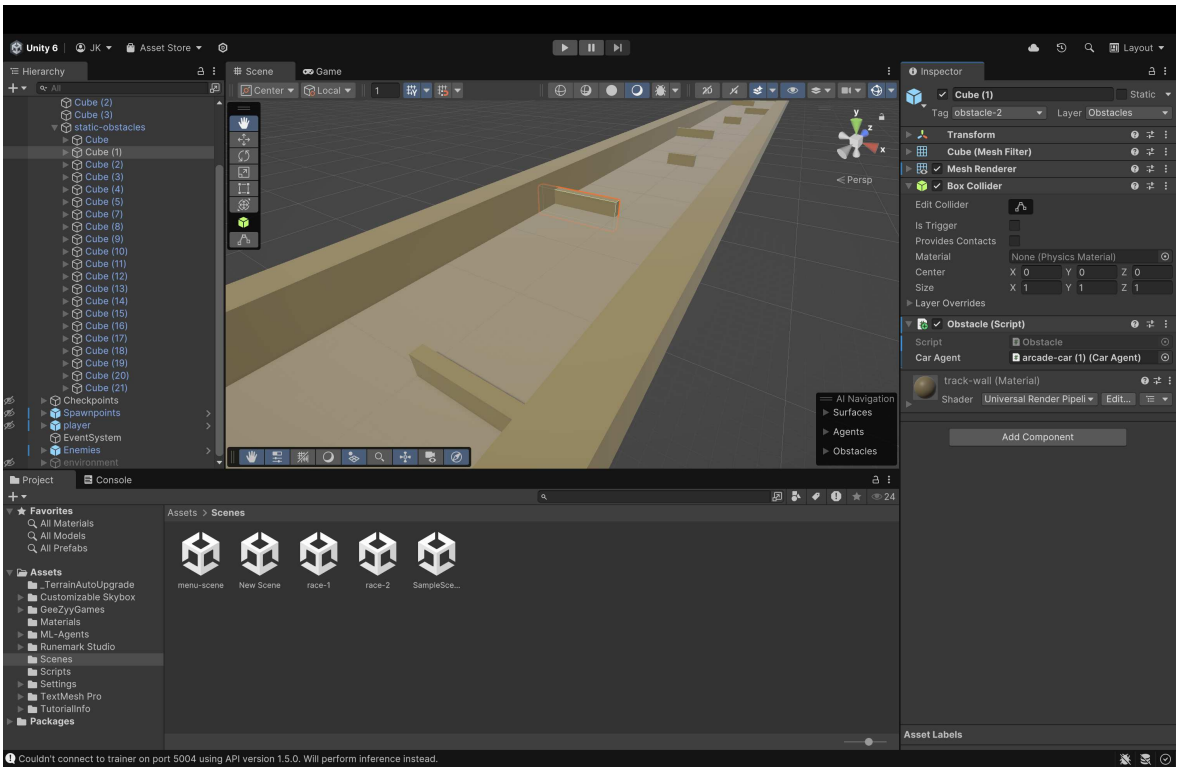
```
3       theRB.transform.parent = null;
4   }
5
6   void Update()
7   {
8       transform.position = new Vector3(
9           theRB.transform.position.x,
10          theRB.transform.position.y - 0.5f,
11          theRB.transform.position.z
12      );
13  }
14
15  private void FixedUpdate()
16  {
17      if (Mathf.Abs(speedInput) > 0)
18      {
19          theRB.AddForce(transform.forward * speedInput);
20      }
21      theRB.linearDamping = dragOnGround;
22  }
```

2.2.3. Implementacija Prepreka i Dinamičnosti Staze

Sljedeći korak u izradi igre je dodavanje prepreka na stazu, čime se uvodi dodatni izazov za igrača i povećava raznolikost igrivosti. Za izradu prepreka ponovno sam iskoristio jednostavne Unity-eve kocke (Cube) kao osnovne građevne blokove.

Svaka prepreka ima pridružen Collider atribut, koji omogućuje fizičku interakciju s automobilom. Ova komponenta osigurava da automobil ne može prolaziti kroz prepreke, već da se s njima sudara, što dodaje realizam i zahtjeva preciznost u vožnji.

Također, svim preprekama je dodijeljen jedan od tri Tag-a: obstacle-1, obstacle-2 i obstacle-3. Ovi Tag-ovi će mi u sljedećem koraku omogućiti da igrač odabire "težinu" staze.



Slika 2.6. Prepreke

2.2.4. Glavni Izbornik

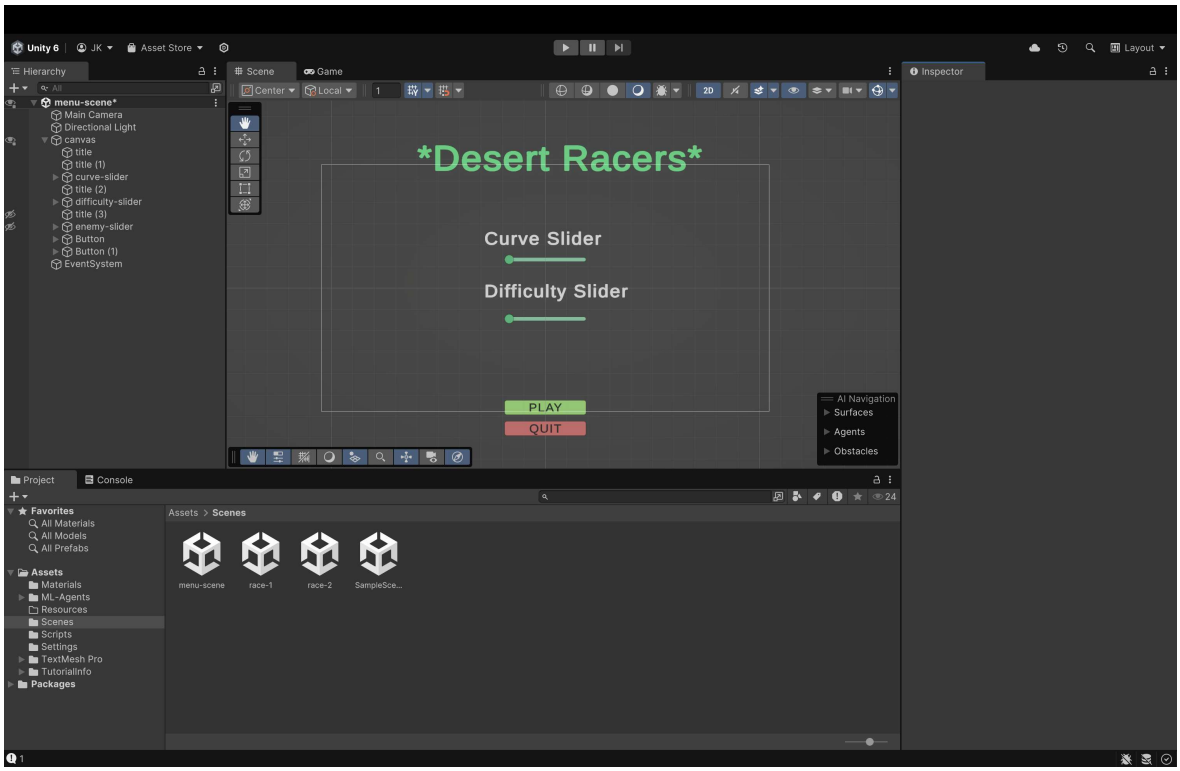
Trenutačno imamo dvije staze (s preprekama) i automobil koji možemo voziti na tim stazama. Zadnji korak u ovom poglavlju je izrada glavnog izbornika.

Glavni izbornik je početna scena koju igrač vidi. Kroz to sučelje, korisnik odabire stazu koju želi voziti i težinu staze - broj prepreka na njoj.

Za izradu glavnog izbornika koristi se Unity-ev Canvas objekt, koji omogućuje postavljanje elemenata korisničkog sučelja unutar 2D prostora. Ključni dio ovog sučelja su klizači, putem kojih igrač može prilagoditi postavke igre, poput težine staze.

Za upravljanje odabranim vrijednostima koristi se skripta GameManager, koja prati interakcije igrača sa sučeljem. Kada igrač promijeni vrijednost klizača, ta se informacija zapisuje u skriptu GameSettings.

GameSettings skripta je drugačije od ostalih jer se koristi kao “spremište”. Jedinu informaciju koju (trenutačno) sadrži je Difficulty koja se postavlja prilikom mijenjanja slidera. Ta vrijednost ostaje sačuvana i između scena, te ju mogu dohvatiti pri pokretanju scene utrke (bilo koje od staza).



Slika 2.7. Glavni izbornik

Listing 2..2: GameSettings.cs

```
public static class GameSettings
{
    public static int Difficulty { get; set; }
}
```

Pomoću ove vrijednosti, dinamički odabirem koliko prepreka će se stvoriti na stazi. Kao što sam napomenuo, sve prepreke imaju jedan od tri Tag-a: obstacle-1, obstacle-2 ili obstacle-3. Prilikom pokretanja staze, skripta ObstacleSpawner gleda vrijednost Difficulty iz GameSetting skripte i odlučuje koje prepreke će se stvoriti na sceni.

Listing 2..3: ObstacleSpawner.cs

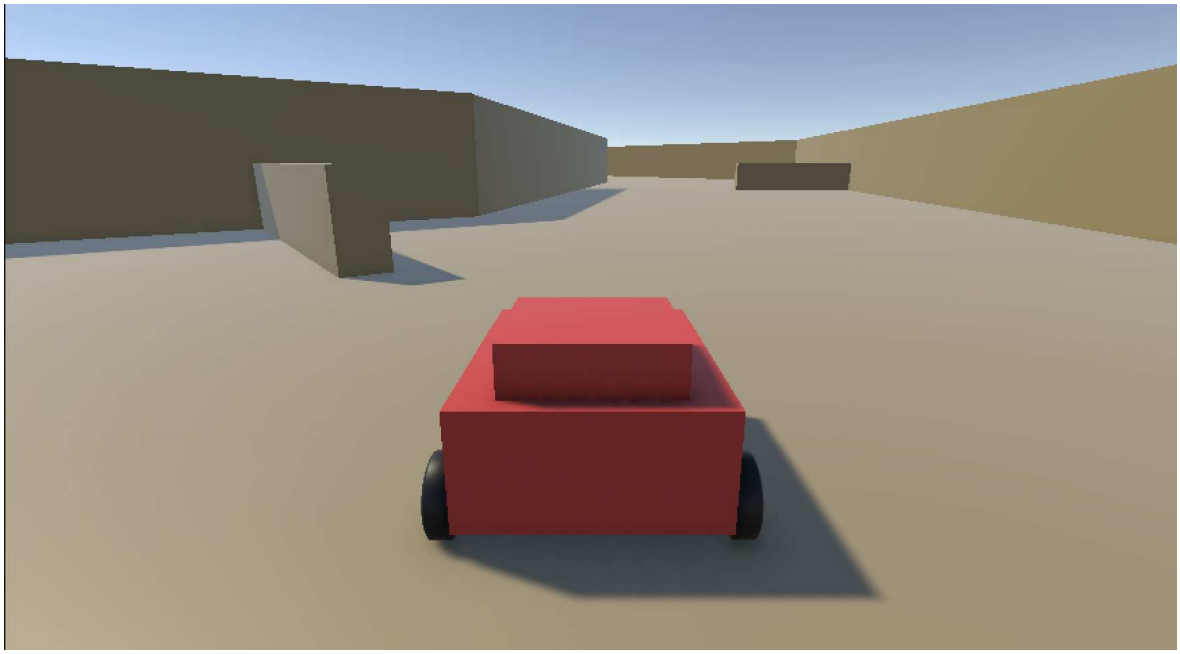
```
void Start()
{
    difficulty = GameSettings.Difficulty;
    SpawnObstacles(difficulty);
}

```

```
7  void SpawnObstacles(int difficultyLevel)
8  {
9      foreach (Transform child in transform)
10     {
11         // Deactivate all obstacles by default
12         child.gameObject.SetActive(false);
13
14         // Activate obstacles based on difficulty level
15         if (difficultyLevel >= 1 && child.CompareTag("obstacle-1"))
16         {
17             child.gameObject.SetActive(true);
18         }
19         else if (difficultyLevel >= 2 && child.CompareTag("obstacle-2"))
20         {
21             child.gameObject.SetActive(true);
22         }
23         else if (difficultyLevel >= 3 && child.CompareTag("obstacle-3"))
24         {
25             child.gameObject.SetActive(true);
26         }
27     }
28 }
```

2.2.5. Osnovna Igra Utrke

Kroz prijašnje korake, došli smo do osnovne igre utrke. U ovoj fazi, igra je sagrađena od jednostavnih poligona, a sastoji od glavnog izbornika, dvije staze s prilagodljivim preprekama i automobila kojim igrač može upravljati.

Nad ovom osnovom, dalje ćemo nadograditi igru uključivanjem AI protivnika i vizualnim unaprjeđenjem svih elemenata.



Slika 2.8. Osnovna igra utrke

3. Strojno Učenje

Strojno učenje (machine learning) [3] jedno je od najvažnijih područja umjetne inteligencije, čiji je cilj razvoj algoritama i modela koji omogućuju računalima da automatski uče i donose odluke na temelju podataka bez potrebe za eksplicitnim programiranjem za svaku pojedinačnu situaciju.

Ključne značajke strojnog učenja uključuju učenje iz podataka gdje algoritmi koriste povijesne podatke za prepoznavanje uzoraka i odnosa, generalizaciju pri kojoj se modeli treniraju na podacima te koriste za analizu novih, neviđenih podataka, i prilagodljivost jer se sustavi strojnog učenja mogu poboljšavati dodavanjem novih podataka i ponovnim treniranjem modela.

Pristupi strojnog učenja obično se dijele u tri glavne kategorije, koje odgovaraju različitim paradigmama učenja, ovisno o vrsti "signala" ili "povratne informacije" koje sustav može koristiti: nadzirano učenje, nenadzirano učenje i podržano učenje.

Skice u ovom poglavlju (i sljedećim poglavljima) su napravljene uz pomoć Excalidraw alata [4].

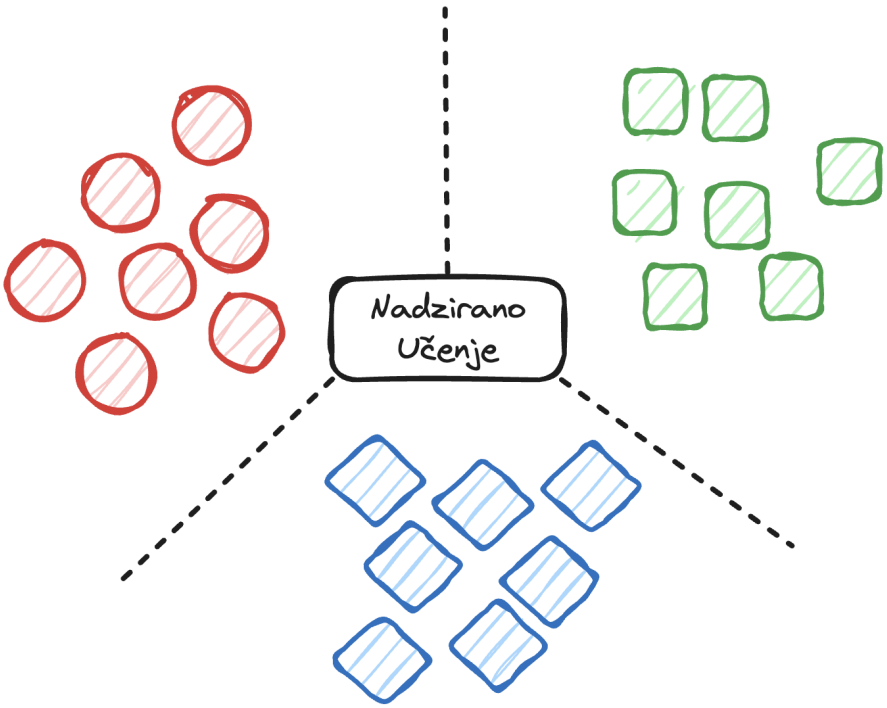
3.1. Nadzirano Učenje

Nadzirano učenje (supervised learning) je metoda strojnog učenja u kojoj algoritmi uče na temelju označenih podataka. Svaki ulazni podatak u skupu za obuku povezan je s odgovarajućom izlaznom vrijednošću, što omogućuje modelu da prepozna uzorke i odnose između ulaza i izlaza.

Cilj nadziranog učenja je izgraditi model koji može generalizirati naučene odnose kako bi točno predvidio ili klasificirao rezultate za nove, nepoznate podatke. Ova me-

toda koristi se u različitim kontekstima, poput klasifikacije, gdje se podaci razvrstavaju u diskretne kategorije, i regresije, gdje se predviđaju kontinuirane vrijednosti.

Primjene nadziranog učenja uključuju prepoznavanje govora, analizu slika, financijske predikcije i mnoge druge zadatke. Uspješnost nadziranog učenja ovisi o kvaliteti i količini označenih podataka, što znači da su kvalitetni podaci ključni za točnost modela.



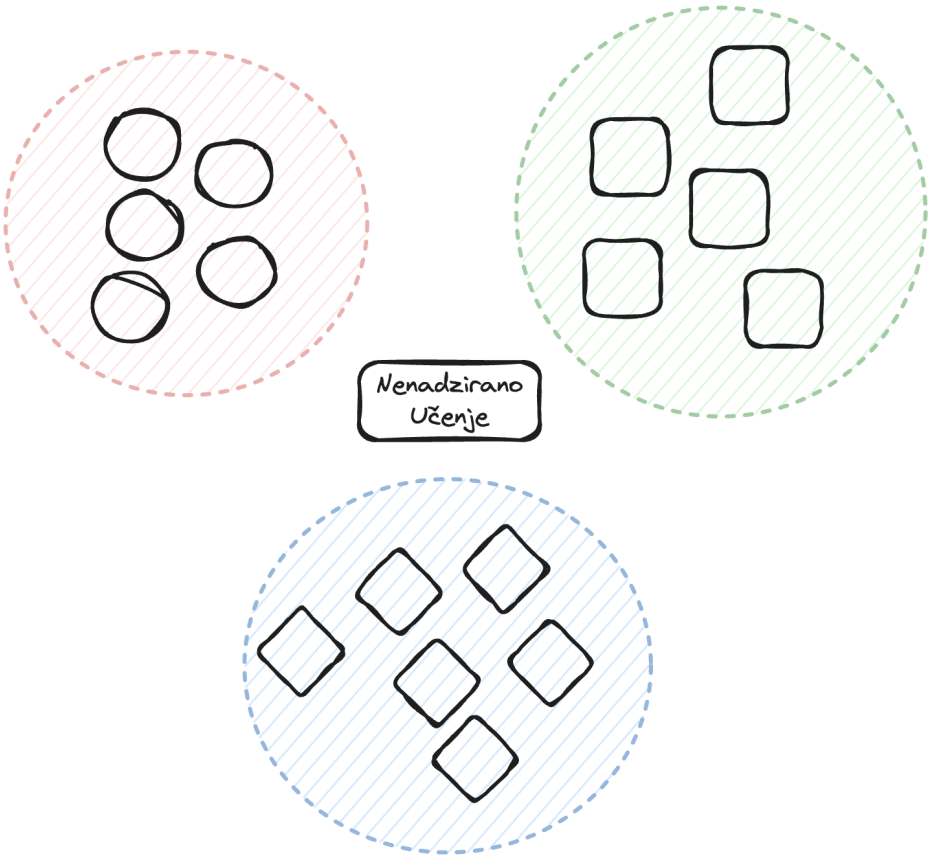
Slika 3.1. Rezultat nadziranog učenja je klasifikacija

3.2. Nenadzirano Učenje

Nenadzirano učenje (unsupervised learning) je metoda strojnog učenja u kojoj algoritmi rade s neoznačenim podacima, bez unaprijed definiranih izlaznih vrijednosti. Cilj ovog pristupa je otkriti skrivene strukture, obrasce ili veze u podacima bez ikakvih smjernica ili oznaka. Algoritam analizira ulazne podatke i pokušava identificirati grupe, zajedničke značajke ili odnose među podacima.

Nenadzirano učenje često se koristi za klasteriranje, gdje se podaci grupiraju na temelju sličnosti, ili za smanjenje dimenzionalnosti, što omogućuje bolje razumijevanje i vizualizaciju kompleksnih podataka.

Primjena ove metode uključuje segmentaciju tržišta, analizu ponašanja korisnika i otkrivanje anomalija. Nenadzirano učenje korisno je kada nije dostupno dovoljno označenih podataka ili kada je cilj istražiti i razumjeti osnovne strukture podataka bez prethodnih pretpostavki.



Slika 3.2. Rezultat nenadziranog učenja je klasteriranje

3.3. Podržano Učenje

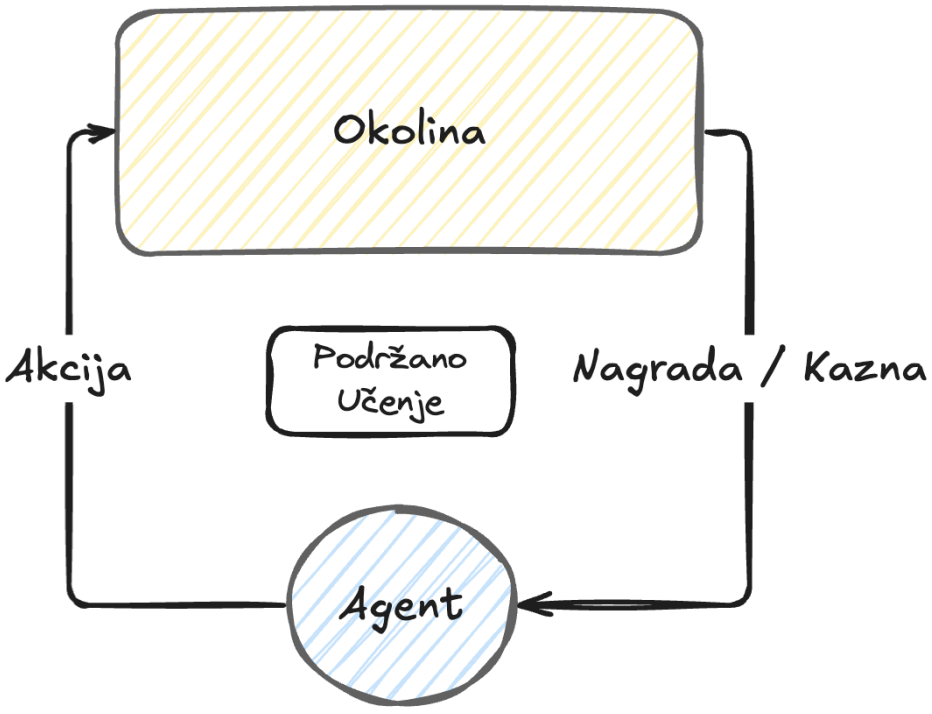
Podržano učenje (reinforcement learning) je metoda strojnog učenja u kojoj algoritam uči donošenje odluka putem interakcije s okolinom. Za razliku od nadziranog učenja, gdje model uči na temelju označenih podataka, podržano učenje temelji se na sustavu nagrada i kazni. Algoritam, često nazivan "agent", donosi odluke u dinamičkom okruženju s ciljem maksimiziranja ukupne nagrade tijekom vremena.

Proces podržanog učenja uključuje ponavljajuće donošenje odluka, gdje agent izvršava akciju, dobiva povratnu informaciju iz okoline u obliku nagrade i ažurira svoju stra-

tegiju na temelju toga. Ključna komponenta ovog procesa je ravnoteža između istraživanja novih mogućnosti i iskorištavanja dosadašnjeg znanja za postizanje boljih rezultata.

Podržano učenje nalazi široku primjenu u zadacima gdje je potrebno prilagodljivo odlučivanje, poput autonomne vožnje, igranja igara, upravljanja robotima i optimizacije industrijskih procesa. Posebno je korisno u situacijama gdje okolina nije u potpunosti predvidljiva i gdje se optimalne akcije ne mogu unaprijed definirati. Ovaj pristup omogućuje stvaranje sustava koji uče iz vlastitog iskustva i neprestano se poboljšavaju kroz iterativnu interakciju s okolinom.

U ovom radu fokusirat ćemo se na podržano učenje, a u sljedećem poglavlju istražit ćemo kako se implementira u Unity okruženju.



Slika 3.3. Podržano Učenje

4. Podržano Učenje u Unity Okruženju

Unity Machine Learning Agents Toolkit (ML-Agents Toolkit) [5] je open-source projekt koji omogućuje korištenje igara i simulacija kao okruženja za treniranje inteligentnih agenata. S pomoću ovog alata agenti se mogu trenirati koristeći podržano učenje, učenje imitacijom, neuroevolucija ili druge metode strojnog učenja, putem jednostavnog Python API-ja. Toolkit također nudi implementacije suvremenih algoritama temeljenih na PyTorch-u, čime se programerima igara i entuzijastima olakšava treniranje inteligentnih agenata za 2D, 3D i VR/AR igre. Trenirani agenti mogu imati različite primjene, poput upravljanja ponašanjem neigrivih likova (NPC) u različitim okruženjima, uključujući višestruke agente i kompetitivne scenarije, automatiziranog testiranja verzija igara te procjene različitih dizajnerskih odluka prije objave igre.

S pomoću ML-Agents alata moguće je trenirati ponašanje neigrivih likova (NPC), poznatih kao agenti, koristeći različite metode. U svakom momentu igre (environment) potrebno je definirati tri ključna elementa.

Prvi element su opažanja (observations), odnosno ono što agent, poput automobila, percipira o okruženju. Opažanja mogu biti numerička ili vizualna. Numerička opažanja odnose se na atribute okruženja iz perspektive agenta, poput značajki ceste i prepreka koje su vidljive automobilu. Za složenija okruženja, agent često treba kontinuirana numerička opažanja. Vizualna opažanja, s druge strane, predstavljaju slike koje dolaze s kamera pričvršćenih na agenta i pokazuju što agent vidi u određenom trenutku. Važno je razlikovati opažanja agenta od stanja okruženja. Stanje okruženja sadrži sve informacije o sceni, uključujući i one koje agent ne može percipirati, dok opažanja obuhvaćaju samo informacije dostupne agentu. Na primjer, automobil neće moći opažati prepreka koje su u daljini ili skrivene iza drugih prepreka.

Drugi element su akcije (actions), odnosno radnje koje agent može poduzeti. Kao i opažanja, akcije mogu biti kontinuirane ili diskretne, ovisno o složenosti okruženja. U jednostavnom scenariju, poput rešetkastog (grid) svijeta gdje je bitna samo lokacija automobila, diskretne akcije koje uključuju kretanje prema sjeveru, jugu, istoku ili zapadu bile bi dovoljne. Međutim, u složenijem okruženju, automobil bi mogao imati kontinuirane akcije, poput odabira smjera i brzine kretanja.

Treći ključni element su nagradni signali (reward signals), koji predstavljaju numeričku vrijednost koja ukazuje na to koliko dobro agent obavlja svoj zadatak. Nagrada se ne mora dodijeliti nakon svake akcije, već samo u trenucima kad agent napravi nešto značajno, dobro ili loše. Na primjer, automobil može dobiti negativnu nagradu ako se sudari s preprekom ili pozitivnu nagradu ako dostigne kontrolnu točku (checkpoint). Nagradni signal ključan je za komunikaciju ciljeva zadatka agentu, pa ga treba postaviti tako da maksimiziranje nagrada dovede do željenog optimalnog ponašanja.

Nakon što se definiraju ova tri elementa - opažanja, akcije i nagrade - agent može započeti s učenjem. To se postiže simuliranjem okruženja u brojnim ponavljanjima, tijekom kojih agent postupno uči koje su akcije optimalne za svaku situaciju kako bi maksimizirao svoju buduću nagradu. Tako, automobil uči ponašanja koja ga čine uspješnim, poput brzog prolaženja kroz stazu. U terminologiji podržanog učenja, naučeno ponašanje naziva se politika ili strategija, što predstavlja optimalno preslikavanje opažanja u akcije. Proces učenja politike kroz simulacije naziva se fazom treniranja, dok se primjena naučene politike u stvarnoj igri naziva fazom inferencije.

ML-Agents Toolkit osigurava sve potrebne alate za korištenje Unityja kao simulacijskog okruženja za učenje politika različitih objekata unutar Unity okruženja.

4.1. Ključne Komponente ML-Agents Toolkit-a

ML-Agents Toolkit sastoji se od četiri glavne komponente. Prva komponenta je okruženje za učenje. Okruženje za učenje sadrži Unity scenu i sve objekte unutar igre. Unity scena definira okruženje u kojem agenti opažaju, djeluju i uče. Unity scene možemo koristiti i za treniranje i za testiranje agenata.

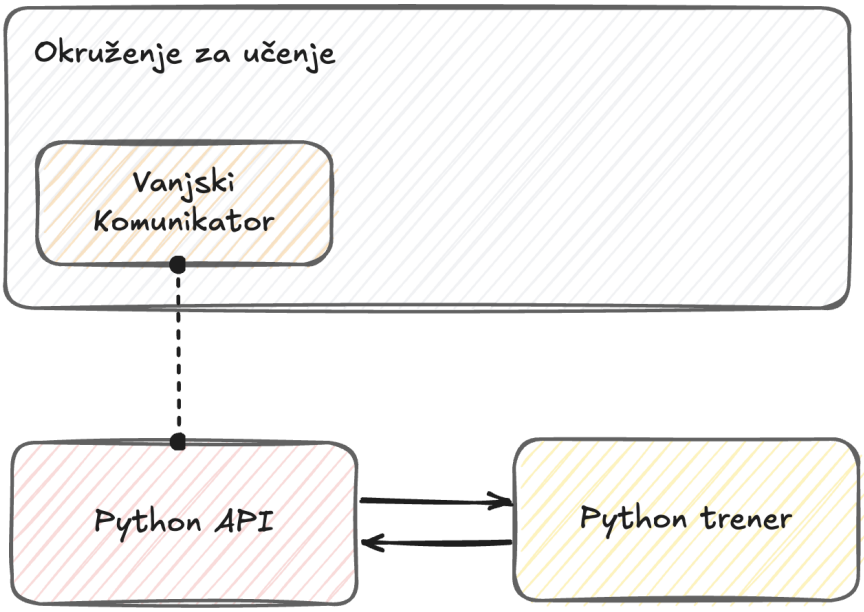
Druga važna komponenta su Python treneri. Oni sadrže algoritme strojnog učenja

potrebne za treniranje agenata, Python treneri komuniciraju isključivo s niskorazinskim Python API-jem.

Python API pruža sučelje a interakciju i upravljanje okruženjem za učenje. Ovo sučelje nije dio Unityja, već funkcionira izvan njega i komunicira s Unityjem putem komunikatora.

Vanjski komunikator služi za povezivanje okruženja za učenje s niskorazinskim Python API-jem. Dio je okruženja za učenje.

Ove komponente zajedno omogućuju korisnicima Unityja da ga učinkovito koriste kao platformu za treniranje i testiranje inteligentnih agenata u raznim simulacijskim okruženjima.



Slika 4.1. Podržano Učenje
[5]

4.2. Okruženje za učenje

U okruženju za učenje u Unityju postoje dvije ključne komponente koje pomažu u organizaciji scene. Prva je agent, koji se povezuje s nekim objektom u sceni (poput auto-

mobila) i odgovoran je za generiranje opažanja, izvođenje primljenih akcija i dodjelu odgovarajućih nagrada, bilo pozitivnih ili negativnih. Svaki agent povezan je s određenim ponašanjem.

Ponašanje (behaviour) definira specifične karakteristike agenta, poput broja akcija koje on može izvesti. Svako ponašanje jedinstveno je određeno putem polja nazvanog "Ime Ponašanja". Ponašanje se može zamisliti kao funkcija koja prima opažanja i nagrade (pozitivne i negativne) od agenta te vraća odgovarajuće akcije.

Postoje tri vrste ponašanja: učenje, heurističko i inferencijsko ponašanje. Učenje se odnosi na ono koje još nije definirano, ali je predviđeno za treniranje. Heurističko ponašanje temelji se na unaprijed definiranim pravilima implementiranim u kodu, dok inferencijsko ponašanje uključuje već istreniranu neuronsku mrežu. Tako, ponašanje koje započinje kao učenje, nakon treniranja prelazi u inferencijsko.

Svako okruženje za učenje u sceni mora imati jednog agenta za svakog lika u sceni. Iako svaki agent mora biti povezan s nekim ponašanjem, moguće je da agenti sa sličnim opažanjima i akcijama dijele isto ponašanje. Na primjeru igre, ako imamo dva automobila, u okruženju će postojati dva agenta, po jedan za svaki automobil. Oba automobila mogu imati isto ponašanje, iako to ne znači da će u svakom trenutku njihova opažanja i akcije biti potpuno identične.

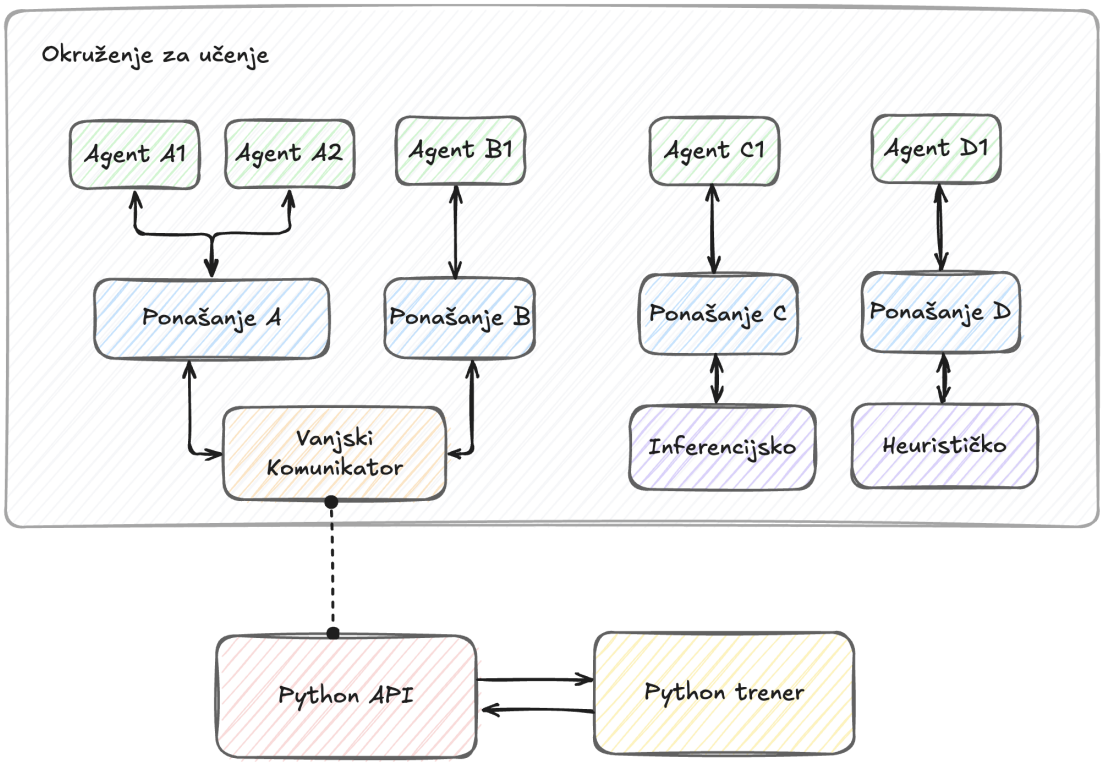
Važno je napomenuti da u jednom okruženju može postojati više agenata i više ponašanja istovremeno. U takvom slučaju, svaka grupa agenata zahtjeva posebno definirano ponašanje kako bi se prilagodila njihovim specifičnim zadacima.

4.3. Mogući Scenariji za Treniranje

U okruženju za učenje postoji nekoliko različitih scenarija interakcije među agentima.

Klasičan pristup uključuje jednog agenta s vlastitim sustavom nagrađivanja, što je idealno za tradicionalne igre za jednog igrača poput utrke automobila.

Kada postoji više neovisnih agenata koji imaju iste parametre ponašanja, ali svaki ima svoj sustav nagrađivanja, može se primijeniti paralelni pristup koji ubrzava i stabilizira proces učenja. Ovaj pristup je koristan, primjerice, kada više automobila istovremeno se



Slika 4.2. Međuodnosi agenata i ponašanja
[5]

utrkuju.

Jedan zanimljiv pristup je protivničko samoučenje, gdje dva agenta međusobno djeluju, ali imaju suprotno definirane sustave nagrađivanja. To omogućuje agentima da postanu sve vještiji dok istovremeno osiguravaju sebi idealnog protivnika - vlastitu repliku.

Kooperativni scenariji uključuju više agenata koji dijele sustav nagrađivanja, pri čemu svi moraju surađivati kako bi postigli zajednički cilj. Takvi zadaci obično zahtijevaju dijeljenje informacija, primjerice kod rješavanja kompleksnih zagonetki. S druge strane, u natjecateljskim scenarijima, agenti djeluju s obrnuto definiranim sustavima nagrađivanja i konkuriraju jedni drugima, bilo u natjecanjima ili u borbi za ograničene resurse, kao što je slučaj u timskim sportovima.

U nekim složenijim okruženjima može se simulirati ekosustav u kojem brojni agenti, s neovisnim ciljevima, ponašanjima i sustavima nagrađivanja, međusobno djeluju. To može uključivati simulacije prirodnih staništa, poput savane s različitim vrstama životi-

nja, ili urbano okruženje u kojem se testiraju sustavi autonomne vožnje.

Za ovaj rad, najzanimljiviji scenariji za treniranje će biti korištenje više agenta koji imaju iste parametre ponašanja.

5. Podržano Učenje u Igri Utrke

Nakon što smo u prethodnim poglavljima izgradili osnovnu igru utrke i prošli osnove Unity ML-Agents Toolkita, sada je vrijeme za podizanje igre na višu razinu. Cilj nam je obogatiti iskustvo igranja implementacijom inteligentnih protivnika s pomoću podržanog učenja.

Sada kada imamo postavljenu bazu igre, stazu, vozila i osnovnu mehaniku, prelazimo na razvoj agenata koji će upravljati protivničkim vozilima. Kroz proces podržanog učenja, ovi agenti će učiti optimalne strategije vožnje, kao što su izbjegavanje prepreka, pronalaženje najbrže putanje i prilagodba promjenama u okruženju. Koristeći ML-Agents Toolkit, omogućit ćemo agentima da postupno razviju vještine vožnje kroz interakciju s definiranim okruženjem i povratnim signalima koje ono šalje.

U ovom poglavlju fokusirat ćemo se na tri ključna koraka. Prvo ćemo definirati agenta i njegov način interakcije s okolinom. Potom ćemo prilagoditi okolinu kako bi mogla slati povratne informacije u obliku nagrada i kazni, omogućujući agentima učenje kroz iskustvo. Na kraju, integrirat ćemo sve elemente i započeti proces treniranja, gdje će agenti učiti kako savladati stazu.

Agente ću trenirati isključivo na jednoj od dvije dostupne staze kako bih mogao procijeniti je li njihovo naučeno ponašanje dobro, ili su pak previše prilagođeni specifičnostima te staze. Za inspiraciju sam koristio repozitorij AI-Racing-Karts [6]

5.1. Osposobljavanje Agenta

Prvi korak je osposobljavanju agenta. U mom slučaju, jedan agent će upravljati jednim automobilom, a za to mi je potrebna nova skripta koja će nasljeđivati klasu Agent [7]. Agent predstavlja entitet unutar okruženja koji opaža svoju okolinu, donosi odluke na

temelju tih opažanja i izvršava odgovarajuće akcije.

Dvije ključne funkcije u ovom procesu su OnActionReceived() i CollectObservations(). Funkcija OnActionReceived() služi za definiranje agentovih akcija i povezana je s komponentom BehaviourParameters, koja omogućuje konfiguraciju ponašanja agenta. Ova komponenta je neophodna za svakog agenta jer određuje kako će se agent ponašati unutar okruženja. Ako postoji već istrenirani model u ONNX formatu, on se kroz ovu komponentu može postaviti kao inferencijski model ponašanja.

Za ovu skriptu sam postavio atribut "Use Child Sensor" na true, a unutar odjeljka Actions, u sekciji Continuous Actions, definirao sam polje s dva elementa. Ova polja po defaultu nemaju značenje, ali ih kroz kod definiram tako da prvo polje (Actions[0]) određuje smjer upravljanja, dok drugo (Actions[1]) kontrolira brzinu vozila kroz skriptu CarController. Kroz ova polja model šalje akcije agentu, omogućujući mu da dinamički upravlja automobilom unutar simuliranog okruženja.

Listing 5..1: CarAgent.cs

```
1 public override void OnActionReceived(ActionBuffers actions)
2 {
3     var Actions = actions.ContinuousActions;
4
5     _carController.ApplyAcceleration(Actions[1]);
6     _carController.Steer(Actions[0]);
7 }
```

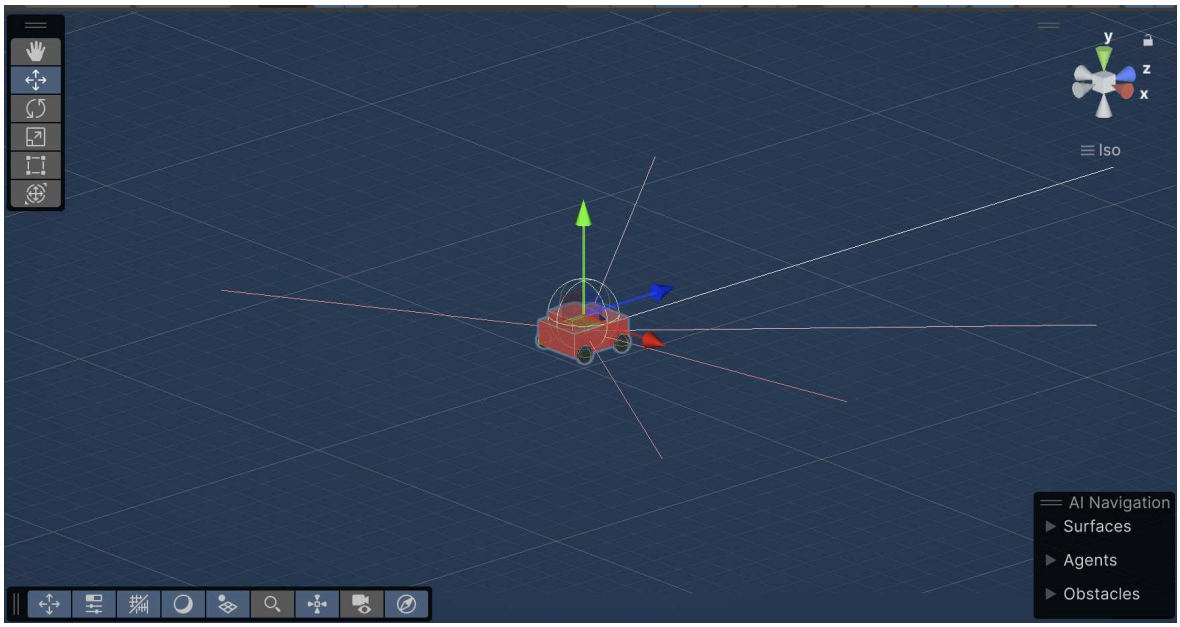
Uz pomoć komponente DecisionRequester [8], koja je pridružena istom Unity objektu kao i CarAgent i BehaviourParameters, automatski se inicira zahtjev za sljedećim korakom ili akcijom modela - bilo da je model u procesu treniranja ili već istreniran.

Funkcija CollectObservations() odgovorna je za prikupljanje vektorskih opažanja agenta tijekom svakog koraka. Ova opažanja omogućuju agentu razumijevanje trenutačnog stanja okoline iz njegove perspektive. Postoji više načina za prikupljanje informacija iz okruženja, poput korištenja kamere, no odlučio sam se za pristup temeljen na RayTracingu. Ova metoda omogućuje precizno detektiranje prepreka i objekata u blizini automobila, što agentu daje relevantne informacije za donošenje odluka u realnom vremenu.

Listing 5..2: CarAgent.cs

```
public override void CollectObservations(VectorSensor sensor)
{
    Vector3 diff = _checkpointManager.nextCheckPointToReach.transform.position - transform.position;
    sensor.AddObservation(diff / 20f);
    AddReward(-0.1f);
}
```

Na automobil sam dodao komponentu RayPerceptionSensor3D [9], koju sam konfigurirao da detektira zidove i prepreke koristeći Tag sistem. Ovim pristupom agent dobiva informacije o položaju prepreka u okolini, omogućujući mu da ih pravovremeno prepozna i izbjegne. RayPerceptionSensor3D funkcionira tako da emitira niz zraka (raycastova) u definiranim smjerovima te vraća podatke o udaljenosti i tipu objekta koji je detektiran. Tako automobil može "vidjeti" svoju okolinu i donositi odluke na temelju primljenih opažanja.



Slika 5.1. RayCast

Zadnja funkcija koju ću spomenuti je Heuristic(). Implementacijom ove funkcije omogućujem si ručno upravljanje automobilom pomoću tipkovnice, što mi olakšava testiranje. Tako mogu provjeriti kako se vozilo ponaša u različitim situacijama prije nego što ga prepustim modelu. Kasnije, zahvaljujući komponenti BehaviourParameters, moći ću koristiti isti Prefab automobila i za agente i za igrača, bez potrebe za dodatnim prilagod-

bama. To znači da će isti model automobila moći funkcionirati i kao autonomni agent i kao vozilo pod mojom kontrolom, ovisno o tome koji način ponašanja odaberem.

Listing 5..3: CarAgent.cs

```
1  public override void Heuristic(in ActionBuffers actionsOut)
2  {
3      var continousActions = actionsOut.ContinuousActions;
4      continousActions.Clear();
5
6      continousActions[0] = Input.GetAxis("Horizontal");
7      continousActions[1] = Input.GetAxis("Vertical");
8  }
```

5.2. Osposobljavanje Okoline

Kako bih mogao naučiti agenta željenom ponašanju, mora biti sposoban primati povratne informacije iz okoline. Kada agent napravi ispravan korak, želim da ga okolina nagradi, dok u suprotnom želim da ga kazni. Tako oblikujemo njegovo ponašanje i usmjeravam ga prema cilju.

Sustav kazni za utrku bio mi je prilično intuitivan. Svaki put kada automobil udari u zid ili prepreku, dobiva kaznu. Osim toga, agent dobiva malu kaznu za svaki korak koji napravi, bez obzira na to je li dobar ili loš. Time ga potičem da se što brže kreće po stazi (ipak je utrka u pitanju).

Definiranje sustava nagrađivanja bilo je izazovnije. Bez njega bi optimalna strategija za agenta bila stajanje u mjestu, jer bi tako minimalizirao broj sudara i izbjegao kazne. Međutim, nagrade trebaju potaknuti automobil da se aktivno kreće po stazi i dovrši utrku.

Najbolje rješenje koje sam osmislio za nagrađivanje agenta jest sustav kontrolnih točaka. Na obje staze postavio sam veliki broj kontrolnih točaka kako bih osigurao da agent napreduje u pravom smjeru. Kako bih ih vizualno istaknuo, dodao sam im poluprozirni narančasti materijal. Svaka kontrolna točka ima Collider komponentu s uključenim atributom "Is Trigger", što znači da mogu detektirati sudare s automobilom, ali ga neće fi-

zički zaustaviti kao što bi to učinili zidovi. Uz to, svaka kontrolna točka ima jednostavnu skriptu koja omogućuje praćenje napretka agenta kroz stazu.

Listing 5..4: Checkpoint.cs

```
public class Checkpoint : MonoBehaviour
{
    private void OnTriggerEnter(Collider other)
    {
        if (other.GetComponent<CheckpointManager>() != null)
        {
            other.GetComponent<CheckpointManager>().CheckPointReached(this);
        }
    }
}
```

Uz pomoć ove skripte signaliziram drugom objektu, odnosno automobilu (agentu), da je dosegao kontrolnu točku. Budući da su jedini objekti koji mogu prolaziti kroz kontrolne točke upravo automobili, svaki takav sudar znači da je agent ostvario napredak na stazi.

Najvažnija skripta u sustavu nagrađivanja okoline je CheckpointManager. Njegova glavna uloga je praćenje napretka automobila i dodjeljivanje nagrada na temelju prijeđenih kontrolnih točaka. Kroz ovaj sustav mogu precizno definirati koliko daleko je agent stigao i osigurati da nagrađivanje potiče poželjno ponašanje.

Prva funkcija koje ću se dotaknuti je Update(). Vrijeme koje automobil ima na raspolaganju da stigne od jedne kontrolne točke do sljedeće iznosi 30 sekundi. Svake sekunde agent dobiva malu kaznu, čime ga potičem na brzo kretanje kroz stazu. Ako mu vrijeme istekne prije nego što dosegne sljedeću kontrolnu točku, epizoda za njega završava i automatski kreće ispočetka. Ovim pristupom osiguravam da se treniranje odvija dinamično i da epizode ne traju beskonačno, već se resetiraju ako automobil zapne na određenom dijelu staze.

Listing 5..5: CheckpointManager.cs

```
private void Update()
```

```
{
    TimeLeft -= Time.deltaTime;

    if (TimeLeft < 0f)
    {
        carAgent.AddReward(-1f);
        carAgent.EndEpisode();
    }
}
```

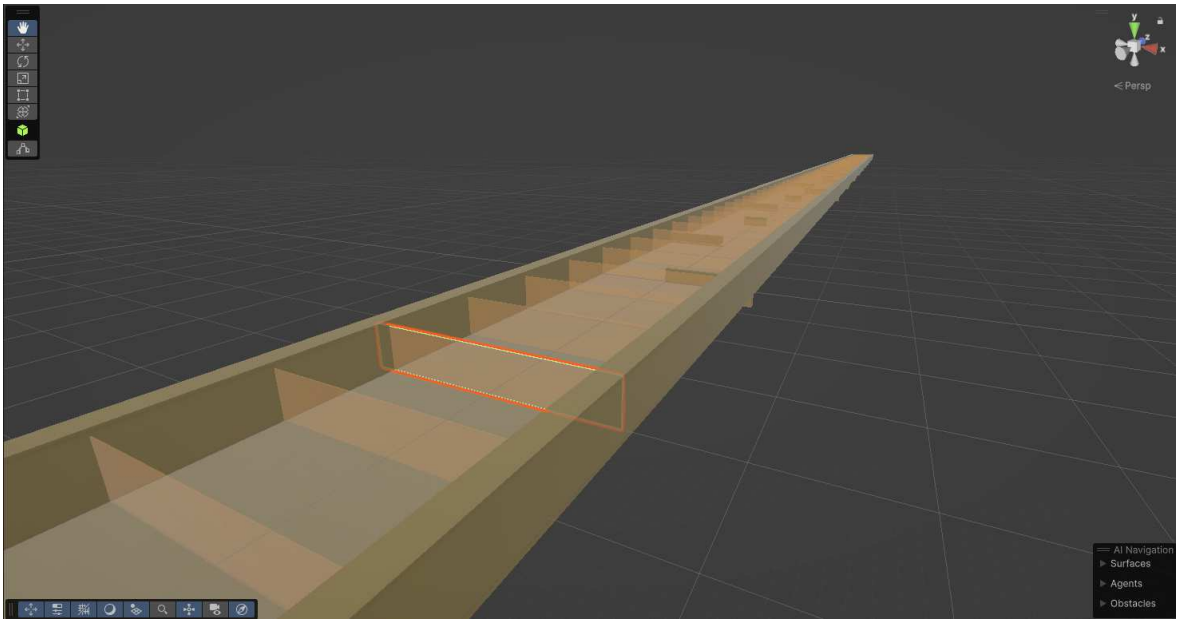
Svaki put kada agent dosegne novu kontrolnu točku, tajmer se resetira i ponovno započinje odbrojavanje. Tako osiguravam da agent ne ostane bez vremena sve dok kontinuirano napreduje kroz stazu.

Osim resetiranja tajmera, agent također prima nagradu, ali samo ako je prvi put prošao kroz tu kontrolnu točku. Time sprječavam iskorištavanje sustava, odnosno da agent ne kruži oko iste kontrolne točke u pokušaju da prikupi više nagrada. Ako agent dođe do zadnje kontrolne točke na stazi, smatra se da je uspješno završio utrku i tada dobiva veliku nagradu kao poticaj da dovrši cijelu stazu.

Listing 5..6: CheckpointManager.cs

```
public void CheckPointReached(Checkpoint checkpoint)
{
    if (nextCheckPointToReach != checkpoint) return;

    lastCheckpoint = Checkpoints[CurrentCheckpointIndex];
    reachedCheckpoint?.Invoke(checkpoint);
    CurrentCheckpointIndex++;

    if (CurrentCheckpointIndex >= Checkpoints.Count)
    {
        carAgent.AddReward(100.0f);
        carAgent.EndEpisode();
    }
    else
```

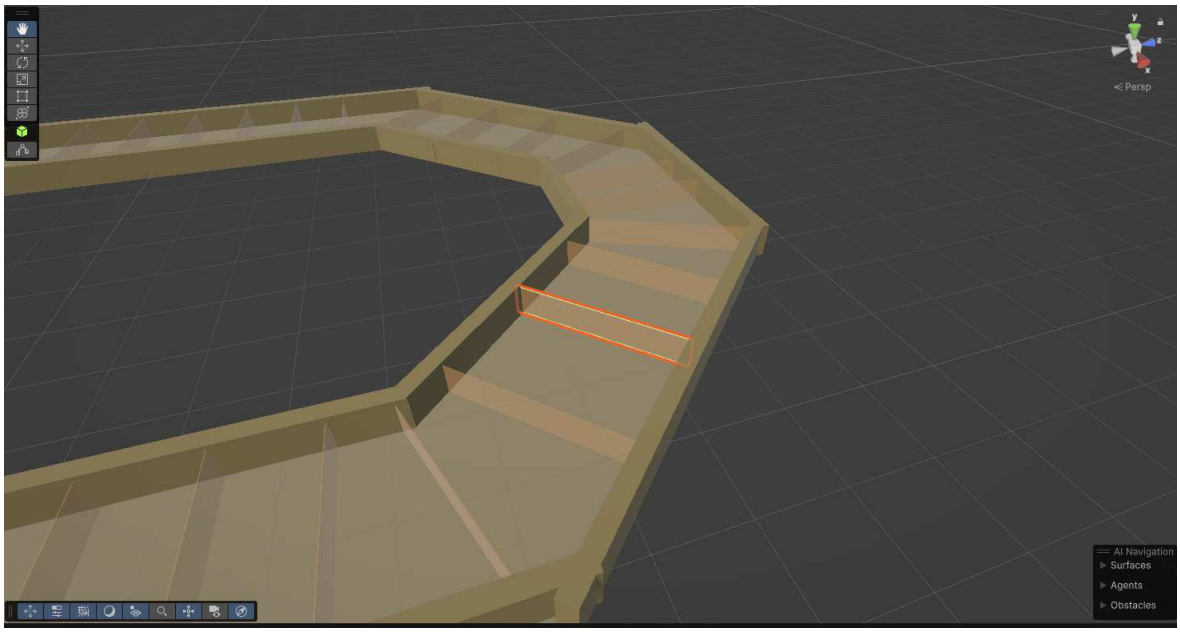
```
15      {
16          carAgent.AddReward((100.0f) / Checkpoints.Count);
17          SetNextCheckpoint();
18      }
19  }
20
21  private void SetNextCheckpoint()
22  {
23      if (Checkpoints.Count > 0)
24      {
25          TimeLeft = MaxTimeToReachNextCheckpoint;
26          nextCheckPointToReach = Checkpoints[CurrentCheckpointIndex];
27
28      }
29  }
```



Slika 5.2. Kontrolne točke na prvoj stazi

5.3. Treniranje Agenata

Agent i okruženje za učenje su sada potpuno spremni, što znači da možemo započeti proces treniranja.



Slika 5.3. Kontrolne točke na drugoj stazi

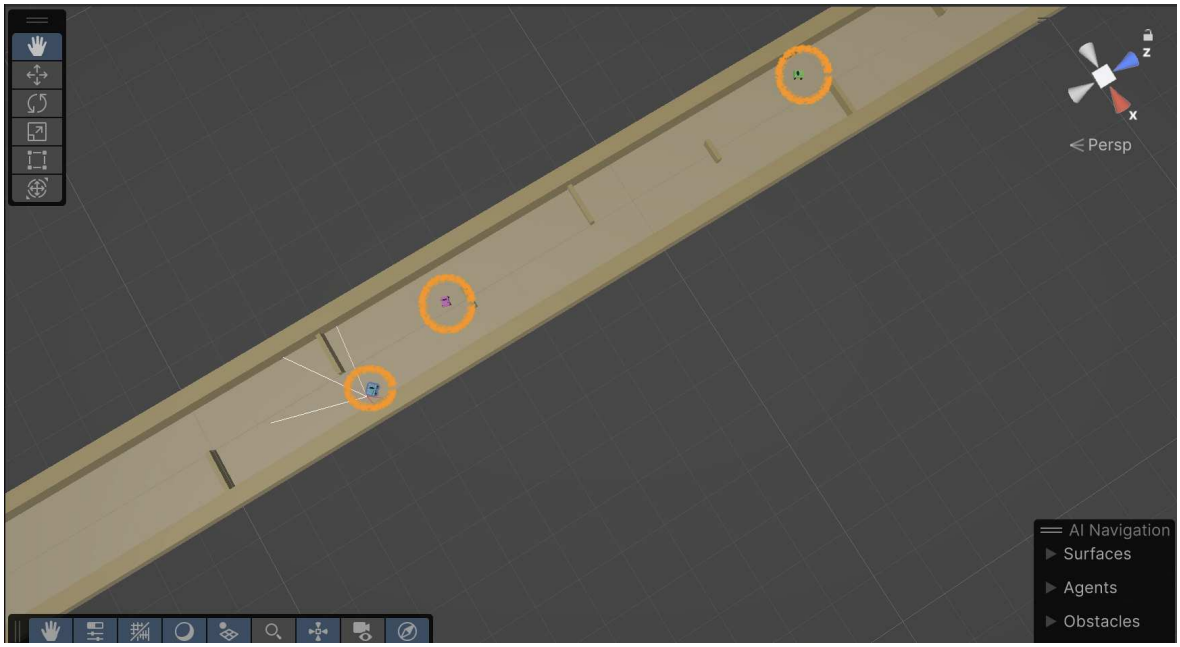
Kako bih pokrenuo treniranje, prvo je bilo potrebno postaviti Python virtualno okruženje i instalirati Unity ML-Agents Toolkit. Nakon toga, pokretanje učenja je vrlo jednostavno - u terminalu jednostavno upišem “mlagents-learn” i zatim pokrenem Unity scenu.

Treniranje se može obavljati izravno iz Unity editora ili putem izgrađenog (build) okruženja. Treniranje unutar Unity editora je praktičnije za testiranje i iteracije, ali ako izradim build igre, mogu pokrenuti više instanci simultano, što značajno ubrzava proces treniranja. Korištenjem izgrđenog okruženja moguće je iskoristiti više računalnih resursa i trenirati agenta u paralelnim sesijama, čime se postiže brža konvergencija modela.

Također, postavljanjem više automobila (agenata) na stazu istovremeno, značajno ubrzavamo proces treniranja. Kada na istoj stazi trenira više agenata, povećava se količina prikupljenih podataka po iteraciji, što omogućuje brže učenje optimalnog ponašanja.

Svaki agent prikuplja vlastita opažanja i donosi odluke, ali svi zajedno doprinose poboljšanju modela. Time se smanjuje ukupno vrijeme potrebno za treniranje jer model brže konvergira prema učinkovitijim strategijama vožnje.

Slika 5.4. mlagents-learn



Slika 5.5. Treniranje agenata

Dodatno, proces treniranja možemo prilagoditi putem configuration.yaml datoteke. Većinu postavki sam ostavio nepromijenjenima, ali najvažnija prilagodba bila je povećanje maksimalnog broja koraka s 500.000 na 1.500.000. Ova promjena omogućuje agentu da se dulje trenira i time razvije naprednije strategije vožnje, čime se poboljšava kvaliteta završnog modela.

Svako treniranje završava generiranjem modela ponašanja, koji se sprema u ONNX datoteku. Ovaj završni model zatim možemo dodati u BehaviourParameters, čime agent

dobiva mogućnost korištenja naučenog ponašanja. Na taj način, od treniranja podržanim učenjem, došli smo do agenta koji koristi naučeno ponašanje (inferencija).

6. Finalni produkt

Napravio sam osnovnu igru utrkivanja i podržanim učenjem istrenirao model ponašanja za automobile. Sljedeći korak je omogućiti igraču da se natječe protiv AI vozača i dati igri vizualni identitet.

Kako bih omogućio igraču da igra protiv AI protivnika, na obje staze ću postaviti pet vozila. Jednim vozilom će upravljati igrač koristeći heurističko ponašanje, što znači da će moći voziti automobil ručno s pomoću tipkovnice. Na to vozilo ću također pričvrstiti kameru kako bi igrač imao odgovarajući pogled na utrku. Ostala četiri vozila koristit će istrenirani AI model, što znači da će voziti autonomno prema naučenom ponašanju. Slično kao što sam omogućio igraču da odabere broj prepreka na stazi, dodat ću opciju da igrač može odabrati koliko AI protivnika želi imati u utrci.

Za završni korak, želim igri dati vizualni identitet. Odlučio sam se za automobilske utrke po pustinji. Kako bih to ostvario, koristio sam Unity Asset Store, odakle sam preuzeo "Lowpoly Environment Extreme Pack" [10], "POLYDesert" [11] i "Customizable Skybox" [12] pakete. Korištenjem asseta iz tih paketa redizajnirao sam izgled staza, prilagođavajući ih pustinjskoj temi i poboljšavajući vizualnu privlačnost igre.

Osim poboljšanja staza, unaprijedio sam i glavni izbornik. Uz pomoć ChatGPT-a [13], kreirao sam pozadinsku sliku te dodao boje na klizače i gumbe, čime sam postigao atraktivniji izgled sučelja.

6.1. Desert Racers

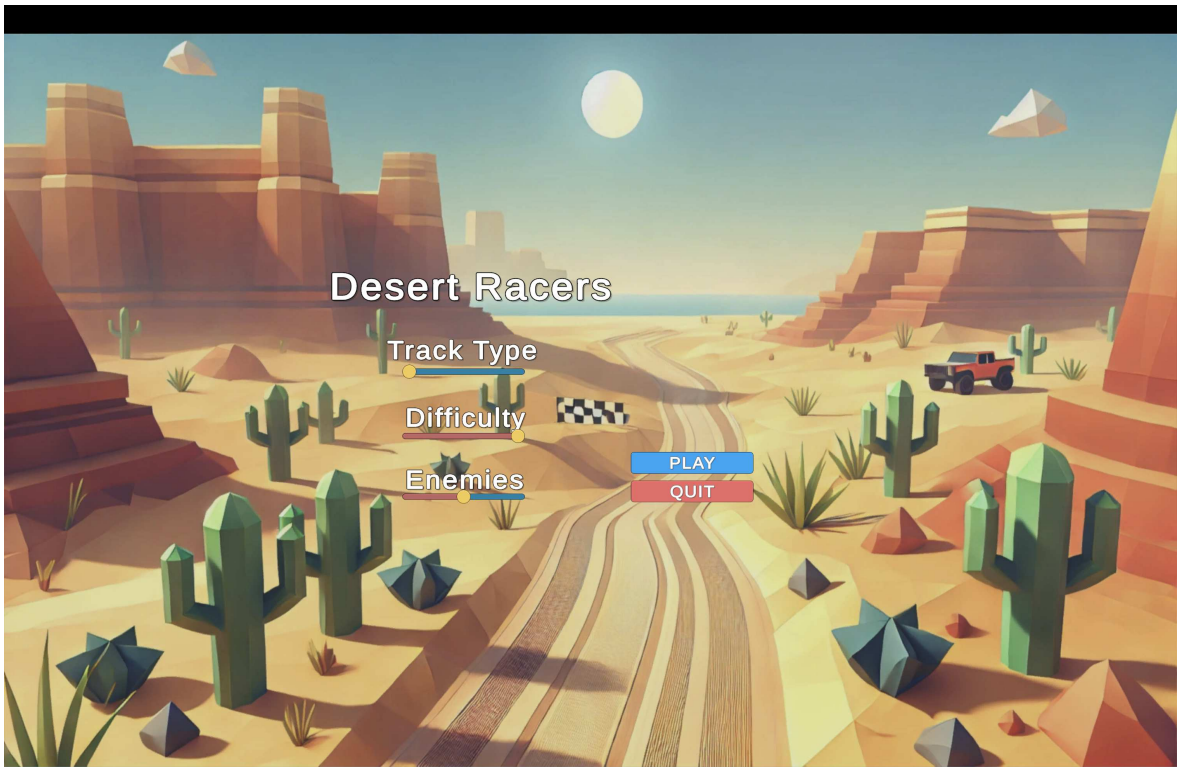
Finalni produkt nosi naziv Desert Racers. Igra započinje scenom glavnog izbornika, gdje igrač može prilagoditi utrku prema svojim željama uz pomoć tri klizača. Može odabrati stazu koju želi voziti, broj prepreka koje će se pojaviti na stazi i broj AI protivnika protiv

Slika 6.1. Pustinjska staza 1

Slika 6.2. Pustinjska staza 2

kojih će se natjecati.

Nakon što igrač postavi željene postavke, klikom na gumb "Play" učitava se jedna od dvije staze, generiraju se prepreke i protivnici, a igrač se postavlja na startnu liniju. Igrač započinje utrku i pokušava prvi doći do cilja.



Slika 6.3. Glavni izbornik

Slika 6.4. Scena utrke #1

Slika 6.5. Scena utrke #2

Slika 6.6. Scena utrke #3

6.2. Daljnja evolucija igre

Igra je u svom trenutačnom izdanju potpuno igriva i stabilna, bez grešaka koje bi narušile osjećaj uranjanja u svijet utrka. Iako se već sada može uživati u igri, postoji mnogo

Slika 6.7. Scena utrke #4

prostora za nadogradnju i poboljšanja. Zahvaljujući modularnom pristupu u dizajnu i razvoju, unapređenja se mogu lako implementirati bez narušavanja osnovne strukture.

Jedan od ključnih elemenata koji bi mogao podići dojam igre na višu razinu je njezin vizualni identitet. Prvotna zamisao bila je smjestiti utrke u postapokaliptičnu pustinju, inspiriranu svijetom "Mad Maxa", u kojem se luđaci bore za posljednje zalihe vode i goriva. Iako su tragovi te ideje još uvijek prisutni u projektu, pa čak i u samom nazivu Unity projekta “MadRacers”, trenutačno okruženje sadrži tek jednostavnu pustinju s kamenjem kao preprekama. Prostor bi se mogao obogatiti ruševinama, metalnim šiljcima, bačvama i raznim zamkama koje bi dodatno naglasile postapokaliptičnu atmosferu. Automobili, koji su trenutačno sačinjeni od osnovnih oblika, također bi mogli dobiti kompleksnije modele s više detalja. Kamera koja prati vozilo zasad je statična, no mogla bi postati dinamičnija, reagirati na nagla ubrzanja i skretanja te tako dodatno pojačati dojam brzine i adrenalina.

Još jedan važan segment koji bi poboljšao doživljaj igre je zvuk. Trenutačno, igra je potpuno tiha, što umanjuje dojam uranjanja u svijet utrka. Dodavanje zvukova motora, škripanja guma, sudara i eksplozija učinilo bi iskustvo vožnje mnogo intenzivnijim. Uz

zvučne efekte, vizualni elementi poput dima, prašine, iskri i eksplozija dodatno bi naglasili kaotičnost utrka i dali im dodatnu dinamiku.

Osim poboljšanja u vizualnom i zvučnom aspektu, igra bi mogla profitirati od veće raznolikosti. Više različitih staza s raznovrsnim preprekama omogućilo bi igračima novo iskustvo sa svakom utrkom. Prepreke bi mogle imati različite efekte, dok neke potpuno zaustavljaju automobil, druge bi ga samo usporavale. Također, umjesto isključivo statičnih prepreka, mogli bi se dodati i pokretni elementi koji bi testirali reflekse igrača i učinili utrke nepredvidljivijima.

Varijacija bi se mogla uvesti i u neprijateljske vozače. Trenutačno svi protivnici koriste isti model ponašanja. Moglo bi se istrenirati više modela ponašanja, svaki s određenom razinom vještine što bi stvorilo izazovnije i zanimljivije utrke. Uz dodatni klizač u glavnom izborniku, igrač bi mogao sam odabrati koliko će inteligentni i kompetentni biti AI protivnici, prilagođavajući igru vlastitim preferencijama.

Još jedno poboljšanje koje bi povećalo preglednost tijekom utrka je dodavanje mini karte u kut ekrana. Tako bi igrač mogao u svakom trenutku pratiti izgled staze i položaj svojih protivnika. Također, rang lista s prikazom vodećih mjesta omogućila bi bolji uvid u poredak automobila i povećala natjecateljski duh igre.

Ono što osjetno nedostaje u igri su prijelazne faze između scena. Kada igrač odabere postavke i započne utrku, odmah se nađe u vožnji bez ikakve pripreme. Uvođenje kratkog odbrojavanja prije starta dalo bi mu priliku da se pripremi i uskladi s ritmom igre. Na sličan način, kraj utrke također bi trebao biti jasnije definiran. Trenutačno igra završava naglo nakon prolaska posljednje kontrolne točke (koje su nevidljive igraču), a igrač se automatski vraća u glavni izbornik. Vizualna oznaka ciljne linije, kao i završni prikaz rezultata s mogućnošću ponavljanja utrke ili povratka u glavni izbornik, učinili bi kraj znatno uglađenijim i jasnijim.

Sve ove nadogradnje ne samo da bi poboljšale vizualni i zvučni identitet igre, već bi i proširile njezinu dubinu i mogućnosti, omogućujući igračima još zabavnije iskustvo utrkivanja u nemilosrdnom postapokaliptičnom svijetu.

Literatura

[1] Unity Technologies, “Unity official website”, https://unity.com/.

[2] Mix and Jam, “Mario kart drift github repository”, https://github.com/mixandjam/MarioKart-Drift.

[3] Wikipedia contributors, “Machine learning”, https://en.wikipedia.org/wiki/Machine_learning.

[4] Excalidraw, “Excalidraw”, https://excalidraw.com/.

[5] Unity Technologies, “Unity ml-agents github repository”, https://github.com/Unity-Technologies/ml-agents.

[6] Sebastian Schuchmann, “Ai racing karts github repository”, https://github.com/Sebastian-Schuchmann/AI-Racing-Karts.

[7] Unity Technologies, “Unity documentation - agent”, https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Agent.html.

[8] ——, “Unity documentation - decision requester”, https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.DecisionRequester.html.

[9] ——, “Unity documentation - rayperception sensor”, https://docs.unity3d.com/Packages/com.unity.ml-agents@1.0/api/Unity.MLAgents.Sensors.RayPerceptionSensor.html.

[10] Unity Asset Store - Gee Zyy Games, “Low poly environment extreme pack”, https://assetstore.unity.com/packages/3d/environments/lowpoly-environment-

extreme-pack-238098.

[11] Unity Asset Store - Runemark Studio, “Poly desert”, https://assetstore.unity.com/packages/3d/environments/landscapes/polydesert-107196.

[12] Unity Asset Store - Key Mouse, “Customizable skybox”, https://assetstore.unity.com/packages/2d/textures-materials/sky/customizable-skybox-174576.

[13] OpenAI, “Chatgpt”, https://chatgpt.com.

Sažetak

Primjena podržanog učenja u računalnoj igri utrke

Juraj Karadža

Brzi napredak strojnog učenja doveo je do njegove široke primjene u raznim područjima, uključujući računalne igre. Podržano učenje, podskup strojnog učenja, omogućuje agentima da kroz interakciju s okolinom nauče optimalna ponašanja, što ga čini posebno korisnim za dinamične scenarije koji zahtijevaju donošenje odluka, poput igara utrkivanja.

Ovaj diplomski rad istražuje integraciju podržanog učenja u okruženje igre utrkivanja razvijeno u Unityju korištenjem ML-Agents Toolkita. Glavni cilj je osmisliti i implementirati sustav u kojem AI agenti autonomno uče kako se kretati trkaćim stazama, izbjegavati prepreke i natjecati se protiv ljudskih igrača. Proces uključuje definiranje opažanja, akcija i sustava nagrađivanja, omogućujući agentima postupno poboljšavanje vještina vožnje kroz iterativne treninge.

Igra Desert Racers sastoji se od više trkaćih staza s prilagodljivim razinama težine, omogućujući igračima da sami odaberu broj prepreka i AI protivnika. Korištenjem Unityjevog ugrađenog fizičkog sustava, razvijeno je strukturirano okruženje koje omogućuje treniranje AI vozila primjenom tehnika podržanog učenja.

Ključne riječi: igra utrke; podržano učenje; podržano učenje u igri utrke; Unity

Abstract

Using Reinforcement Learning in a Racing Game

Juraj Karadža

The rapid advancement of machine learning has led to its widespread application in various fields, including gaming. Reinforcement Learning, a subset of machine learning, enables agents to learn optimal behaviors through interaction with their environment, making it particularly useful for dynamic and decision-intensive scenarios such as racing games.

This thesis explores the integration of reinforcement learning into a racing game environment developed in Unity using the ML-Agents Toolkit. The primary goal is to design and implement a system where AI-driven agents can autonomously learn how to navigate race tracks, avoid obstacles, and compete against human players. The process involves defining observations, actions, and reward functions, allowing the agents to improve their driving skills through iterative training sessions.

The game Desert Racers consists of multiple racing tracks with adjustable difficulty settings, enabling players to modify the number of obstacles and AI competitors. Using Unity's built-in physics engine, a structured environment was developed that enables the training of AI-controlled cars using reinforcement learning techniques.

Keywords: racing game; reinforcement learning; reinforcement learning in a racing game; Unity