

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Dominik Langer

**IMPLEMENTACIJA ML-AGENATA UNUTAR
KARTING UTRKE**

PROJEKT

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Langer

Matični broj: 0016116705

Studij: Organizacija poslovnih procesa

IMPLEMENTACIJA ML-AGENATA UNUTAR KARTING UTRKE

PROJEKT

Mentor:

Dr. sc. Bogdan Okreša Đurić

Varaždin, veljača 2021.

Dominik Langer

Izjava o izvornosti

Izjavljujem da je moj projekt izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrđio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Kroz projekt prikazati će kako se u igru može dodati strojno učenje i upotpunosti ukloniti potrebu za kontrolom od strane korisnika. Radi se o karting utrci u kojoj sudjeluju natjecatelji. Svaki natjecatelj ima svoj go-kart, cilj projekta je implementirati strojno učenje tako da se utrka izvršava sama. Agenti će predstavljati natjecatelje, a njihovim treniranjem želim postići da se svaki agent kreće samostalno bez potrebe da korisnik upravlja go-kartom. Fokusirati će se na implementaciju strojnog učenja i detaljno ju objasniti.

Ključne riječi: Unity, Strojno učenje, ML-Agents, utrka, C#

Sadržaj

1. Uvod	1
2. Metode i tehnike rada	2
3. Razrada teme	3
3.1. ML-Agenti	3
3.1.1. Osnovne komponente	3
3.1.2. Nenadzirano učenje	5
3.1.3. Nadzirano učenje	5
3.1.4. Strojno učenje s potporom	5
3.2. Osnovno razvojno okruženje	7
3.3. Potrebni paketi	9
3.4. Kreiranje igre	12
3.5. Agent	16
3.5.1. Treniranje	19
4. Zaključak	26
Popis literature	28
Popis slika	30
Popis popis tablica	31
1. Prilog 1	33
2. Prilog 2	34

1. Uvod

Sam razvoj video igara vrlo mi je zanimljiv, ali oduvijek me zanimalo kako se unutar neke video igre implementira strojno učenje. Trkaće igre su najbolji primjer. Ako pogledamo bilo koju trkaću igru osim nas korisnika koji upravljaju u ovom slučaju automobilom ili bilo kakvom drugom alternativom, na stazi se još uz nas nalaze brojni drugi automobili. Postavlja se pitanje tko njima upravlja? Ako igramo online onda je odgovor vrlo jednostavan, drugi korisnici (igraci) kontroliraju automobile, ali što je u slučaju kada igru igramo sami. Vožnja po praznoj stazi gdje nema drugih automobila nije toliko zanimljiva kao kad se utrkujemo protiv drugih vozača. Shodno tome bilo je potrebno implementirati druge vozače koji će za nas korisnika predstavljati neki oblik prepreke i povećati doživljaj natjecanja. Kroz ovaj projekt prikazati će kako se može implementirati strojno učenje unutar jedne takve igre. Karting utrke su mi dosta zanimljive, a samom korisniku su relativno dostupne. Nažalost niti jedna igra do sada nije u mogućnosti u potpunosti prenijeti atmosferu utrkivanja. Postoje brojni pokušaji, razvijeni su brojni simulatori koji simuliraju nagib bolida, jačinu skretanja i ostale fizičke parametre, ali niti jedan od njih nije onaj pravi oblik utrkivanja.

Odlučio sam se za izradu karting utrke uz pomoć alata Unity [1], uz Unity postoje brojni alati koje na omogućuju izradu video igara kao što su: GameMaker: Studio, Stencyl, Construct 2 ili Unreal Engine 4. Ipak je Unity jedan od najpopularnijih alata za razvoj video igara. Izrazito je dostupan i ima široku dokumentaciju, ali ono što nas najviše zanima u vidu ovog projekta je strojno učenje. Samo strojno učenje unutar programa Unity implementirano je 2018. godine. "ML-Agents toolkit" omogućuje jednostavan način ukomponiranja strojnog učenja unutar naše igre [2].

2. Metode i tehnike rada

Alate koje će koristiti u ovom projektu su Unity i Visual Studio. Alat Unity omogućiti će mi da na jednostavan način kreiram zamišljenu video igru. Unity je vodeća svjetska platforma za stvaranje i upravljanje interaktivnim 3D sadržajem, sadrži brojne alate koji omogućuju izradu nevjerljivih video igara i njihovo objavljivanje na velikom broju uređaja. Naglasak ove platforme je i na povećanoj mogućnosti rada među razvojnim timovima. Tako na ovoj platformi dizajneri, programeri i umjetnici mogu skladno raditi. Dostupna je izrada 2D ili 3D scena, animacija ili kinematografije izravno u Unity Editoru. Glavna prednost ove aplikacije je mogućnost objavljivanja video igara na više od 20 platformi, neke od njih su :

- **Windows**
- **Mac**
- **iOS**
- **Android**
- **PlayStation**
- **Xbox**
- **Nintendo Switch**
- **AR i VR platforme**

Jedno od najkorisnijih svojstava je prikaz grafike u stvarnom vremenu, a nevjerljive grafičke prikaze možemo kreirati pomoću skriptiranog cjevovoda za renderiranje [1]. Sama velika mogućnost koju aplikacija nudi nije dovoljna da pridobije toliki broj korisnika, vrlo je bitno da sve funkcionalnosti rade izrazito brzo i da sve svoje ideje možemo vrlo efikasno realizirati unutar samog alata. Ovdje na scenu stupa "Data-Oriented Technology Stack (DOTS)" koji nam upotpuni omogućuje da iskoristimo sirovu moć višejezgrenih procesora. Programski jezik C# također ubrzava samo izvođenje skripti i operacija, a dodatna je mogućnost višestruko korištenje istoga kroz neke druge projekte [1].

Visual studio je alat pomoću kojeg možemo razvijati aplikacije za Android, iOS, Mac ili Windows. Unity već unutar osnovnih postavki kreiranja skripti podržava Visual studio zbog dobre implementacije programskog jezika C# [3].

3. Razrada teme

Želja nam je kreirati vide igru, za to smo već pripremili alat Unity koji smo nešto detaljnije opisali prije, ali nismo se detaljno pozabavili njegovim funkcijama, niti ih pokazali. Sljedeće sekcije će se fokusirati na to kako implementirati strojno učenje unutar igre i kako kreirati jednu igru koja ne zahtjeva od korisnika nikakav input.

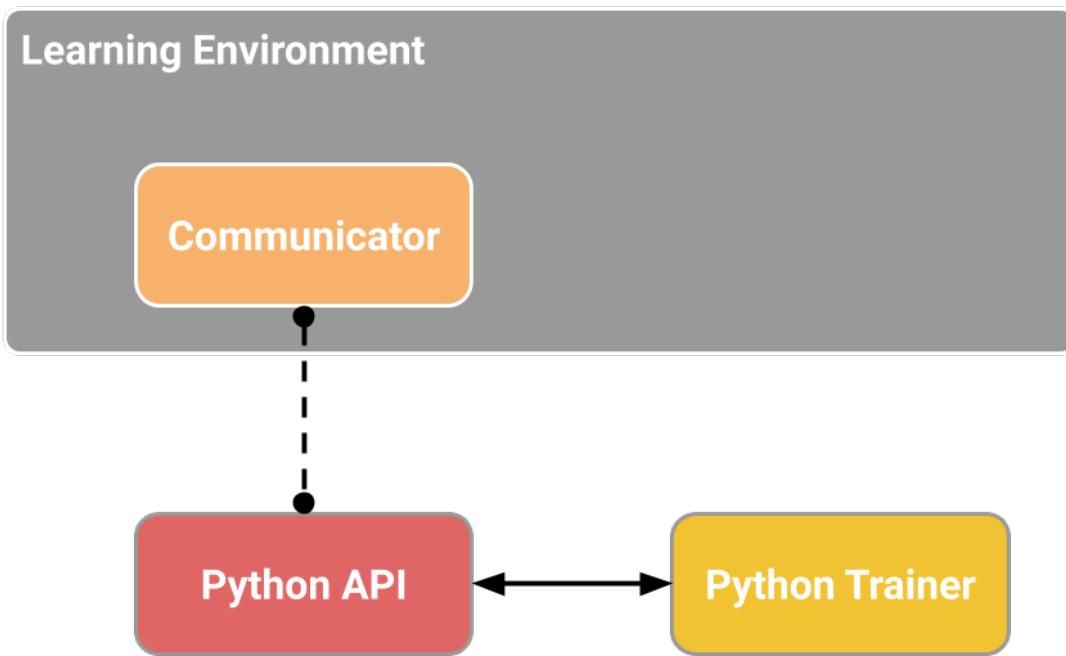
3.1. ML-Agenti

"The Unity Machine Learning Agents Toolkit" je projekt otvorenog koda koji omogućuje igrama i simulacijama implementaciju i treniranje inteligentnih agenata. Za proces treniranja koristi se vrlo jednostavan Python API. Tako trenirani agenti imaju široku primjenu, mogu se koristiti kao NPC, mogu služiti za automatsko testiranje verzija igre prije nego što je ona puštena u javnost [4] [5].

3.1.1. Osnovne komponente

Osnovne komponente od kojih je "Toolkit" sačinjen su:

- **Okruženje za učenje** - Okruženje koje sadrži sve scene i elemente unutar alata. Scene nam služe kao okruženje unutar kojega agent promatra, djeluje i uči. Mogućnost učenja i kreiranja samog okruženja je velika, a sve zavisi o scenariju za koji našeg agenta pripremamo.
- **Python API niske razine** - Ovaj API nije dio aplikacija, ali komunicira sa samom aplikacijom putem komunikatora. Njegova primjena je široka, ali u ovom kontekstu nam omogućuje da Unity koristimo kao okruženje za simuliranje i učenje našeg agenta.
- **Vanjski komunikator** - Služi kao komunikacija između aplikacije Unity i Python API-a, a sastavni je dio Okruženja za učenje.
- **Python treneri** - Sadrži sve algoritme strojnog učenja koji nam omogućuju treniranje agenata. Svi algoritmi su nam dostupni preko jedne naredbe: `mlagents-learn`.
- **"Gym Wrapper"** - Omotač predstavlja klasičan način na koji istraživači strojnog učenja komuniciraju sa okruženjem za simuliranje [4] [6].



Slika 1: Prikaz osnovnih komponenti [4]

Okruženje za učenje sastoji se od dvije osnovne komponente. Agenta koji je dodan objektu unutar igre, a zadužen je za zapažanja, izvođenje radnji i dodavanje pozitivne ili negativne nagrade ovisno o situaciji. Svaki agent je povezan sa svojim ponašanjem ("Behaviour"). Svako ponašanje definira specifične agentove atribute ovisno o broju radnji koje on može poduzeti. Svako ponašanje sadrži svoj jedinstven naziv, a na ponašanje možemo gledati kao funkciju koja bilježi zapažanja i nagrade od Agenta, a za uzvrat daje specifične radnje. Postoje tri tipa ponašanja: učenje, heuristika ili zaključivanje. Učenje samo po sebi nema neke mogućnosti, ali služi nam za treniranje agenta. Heuristika nam najčešće služi kada želimo sami kontrolirati agenta. Zadnji tip je zaključivanje, njega koristimo kada smo istrenirali agenta i imamo dokument koji sadrži neuronsku mrežu. Slijed ponašanja je ovakav: korisnik sam želi isprobati ponašanje agenta, zatim tog agenta unutar procesa učenja želi istrenirati da napravi željene radnje, na kraju dobije neuronsku mrežu koju koristi za pokretanje tog agenta [4] [6].

Kako sami agenti podržavaju strojno učenje potrebno je razjasniti koje tipove strojnog učenja podržavaju, a to su: nadzirano učenje, nenadzirano učenje i strojno učenje s potporom. Svaki tip strojnog učenja uči nad drugačijim skupom podataka. Strojno učenje je samo jedna grana umjetne inteligencije, a fokusira se kao učenje nad skupom podataka. Kako se radi o podacima cijelo vrijeme kroz razvoj sam moramo imati na umu da agenti znaju razaznati brojeve, tj. učenje se provodi nad skupom brojeva, vrijednostima koje agent ima. Zadatak strojnog učenja je izrada programa koji su previše kompleksni da bi se mogli programirati i moraju biti adaptivni. [6] i [7].

3.1.2. Nenadzirano učenje

Glavna zadaća ovog tipa učenja je grupirati slične stavke u jedinstven skup podataka. Najbolji primjer je grupiranje igrača. Skup igrača može razdvojiti na dva dijela, igrače koji vrlo često igraju igru i one koji igraju povremeno. Ovim grupiranjem one igrače koji su aktivniji možemo pozvati u neki "beta" program razvoja igre ili testiranje "beta" servera, a one manje aktivne možemo pokušati motivirati slanjem elektroničke pošte o nekim novim elementima dodanim u igru. Za takvo grupiranje moramo skupljati podatke o igraču. Današnje igre skupljaju podatke o provedenom vremenu, količini novca koje smo potrošili unutar igre ili trenutnom "levelu". Tako vrlo jednostavno možemo dobiti dva skupa podataka koji će nama predstavljati dva tipa igrača. Brojne "multiplayer" igre danas koriste takav oblik učenja, neke to koriste na pravedan način da bolje igrače stavlia u jedinstven "lobby", a druge one korisnike koji više potroše novaca u igri dodjeli slabije igrače [8] i [7].

3.1.3. Nadzirano učenje

Unutar ovog učenja želja nam nije samo grupirati skup podataka, nego kreirati način ponašanja za svaku grupu koji kasnije možemo koristiti kod predviđanja. Tako recimo na prijašnjem skupu podataka može odrediti koji bi od korisnika mogli prestati igrati igru, a dodatno ako neki novi igrač dođe igrati na temeljnu njegovih preferencija može odrediti hoće li prestati igrati nakon određenog vremena ili ne. Kada znamo skup igrača koji bi ubrzo mogli prestati igrati njih možemo specifično ciljati i motivirati ih da ostanu igrati još neko vrijeme [7].

3.1.4. Strojno učenje s potporom

Na ovaj tip učenja možemo gledati kao donošenje sekvencijalnih odluka, koje se često koristi kod kontrole robota. Strojno učenje s potporom uvelike se bazira na zapažanjima i radnjama. Zapažanja predstavljaju sve što naš objekt može zapaziti (izmjeriti) u svom okruženju, a radnja je promjena stanja objekta (kretanje, rotiranje, itd.). Zadnji, ali najvažniji dio ovog tipa učenja je nagrada. Objekt želi postići što veću nagradu. Nagrade mogu imati pozitivnu i negativnu vrijednost, kada objekt napravi baš ono što smo mi od njega tražili tada dobije pozitivnu nagradu, a ako dođe do neke greške tada ga kaznimo. Na ovakav način možemo što bolje trenirati objekt jer će on sam znati kada je nešto dobro napravio, dobio je nagradu i kada je pogriješio, a kako mu je krajnji cilj biti što uspješniji tada će se kretati samo prema nagradama [7]. Proces je lijepo opisan na slici ispod:



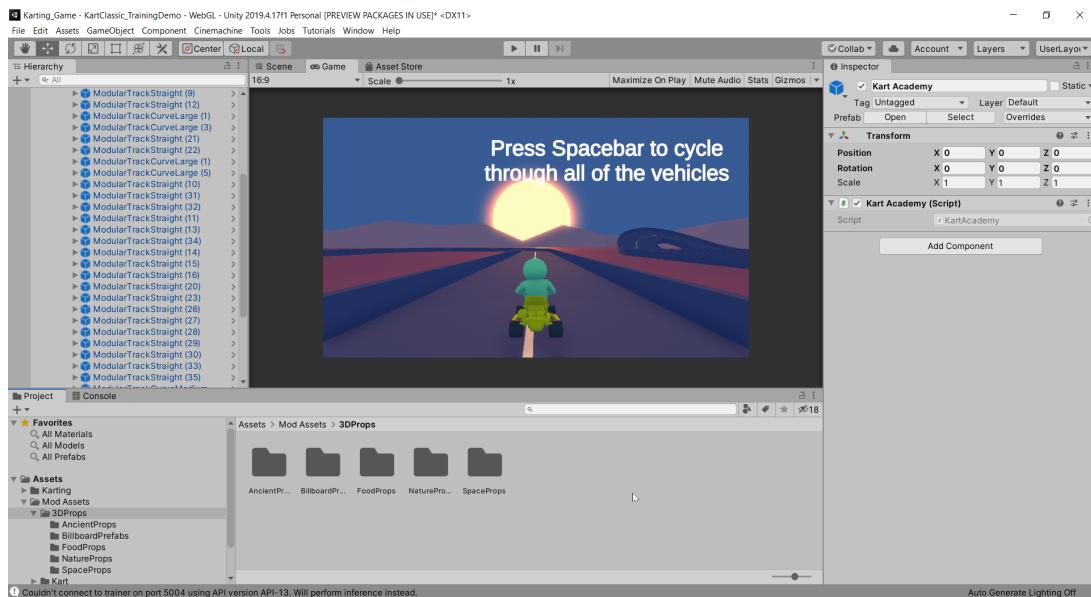
Slika 2: Ciklus strojnog učenja s potporom [7]

Sama kvaliteta učenja uvelike ovisi o kreiranom okruženju, ali pojам razvoja okruženja uvelike pada u sjenu u odnosu na razvijenost algoritma za učenje. Razlog tome može biti što je razvoj samog algoritma nešto lakši od razvoja kvalitetnog okruženja za simuliranje rada istoga. Brojne današnje igre imaju mogućnost vizualno predočiti stvarni svijet do najsitnijeg detalja, veliki se fokus kod razvoja igre stavlja na njen izgled, želi se dostići što bolji prikaz realnosti. Baš iz tog razloga virtualna realnost danas dobiva na velikom značaju, a sve veći napori na tom polju dovode do mogućnosti razvoja boljih simulacijskih okruženja koje doprinose razvoju učenja i implementacije umjetne inteligencije. Jedan zanimljiv primjer na koji sam naišao je: "Project Malmo" [5]. Radi se o simulacijskoj platformi koja se bazira na istraživanju i građenju kao u igri Minecraft. platforma pruža veliku razinu fleksibilnost u definiranju scenarija i tipovima okruženja što ju čini specifičnom za tu domenu. Jedina ograničenja koja se postavljaju je sam pokretač igre koji se temelji na pikseliziranom izgledu i ne baš zavidnoj razini fizike unutar igre [5].

Ovaj tip učenja koristim za izradu projekta, karting predstavlja objekt koji se kreće kroz kreiranu stazu, svaki put kada prođe kroz kontrolnu točku dobije nagradu, ali ima i vremensko ograničenje tako da sam karting želi biti što brži u prolasku. Karting sam po sebi nije u mogućnosti razaznati svoje okruženje zbog toga mu se mora dodati "radar" koji će pratiti kako se on kreće kroz stazu, koliko je blizu samog zida staze ili koliko je blizu kontrolne točke. Kada se proces učenja pokrene on kroz prvi prolaz neće dati najbolje rezultate, ali kako imam sustav nagrada shvatiti će kojim se putem mora kretati kako bi dobio što više nagrada. Na kraju će agent (karting) kreirati najbolju putanju po kojoj će se kretati. Osim za izradu video igara ovaj oblik učenja može se koristiti kod kreiranja autonomnih robota koje nismo u mogućnosti pustitit u stvaran svijet da se prvo jedno vrijeme uče pa da tek onda budu zapravo korisni. Zbog toga se koristi simulacijska sposobnost programa Unity koja može trenirati robota i pripremiti ga za stvaran svijet [7] [6].

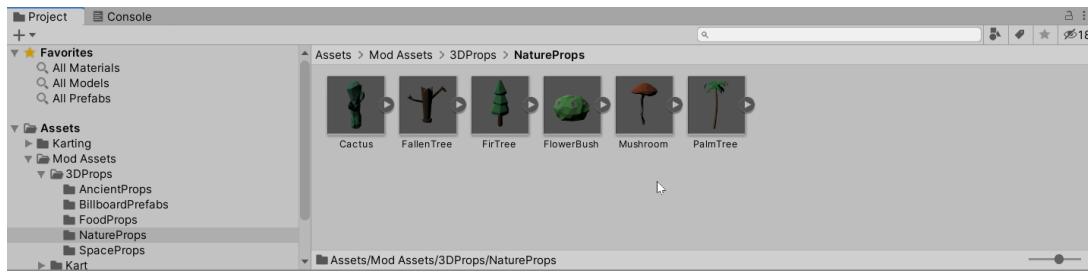
3.2. Osnovno razvojno okruženje

Unity je alat koji nam na vrlo jednostavan način omogućuje da sve svoje ideje za izgradnju video igara izvedemo u djelo. Sastoji se od četiri glavna dijela. Središnji dio predstavlja "prozor" u našu igru, unutar njega možemo dodavati ili manipulirati modelima naše igre. Lijevo od njega nalazi se "Hierarchy", prikazani su svi elementi koje smo koristili u igri. Unutar nje možemo pomicati elemente, grupirati ih. Desno od središnjeg prozora nalazi se "Inspector" kada odaberemo neki element on će nam onda pokazati sve bitne informacije o tome elementu i omogući nam da radio izmjene svih dostupnih parametara. Na dnu nalazi se "Project" koji nam pokazuje što nam se sve nalazi unutar našeg projekta, ovdje možemo pronaći modele, materijale ili unaprijed kreirane modele. Sam izgled našeg programa možemo modificirati i prilagoditi našim potrebama, potrebno je samo povući prozor na željeno mjesto, možemo ga proširiti ili smanjiti.



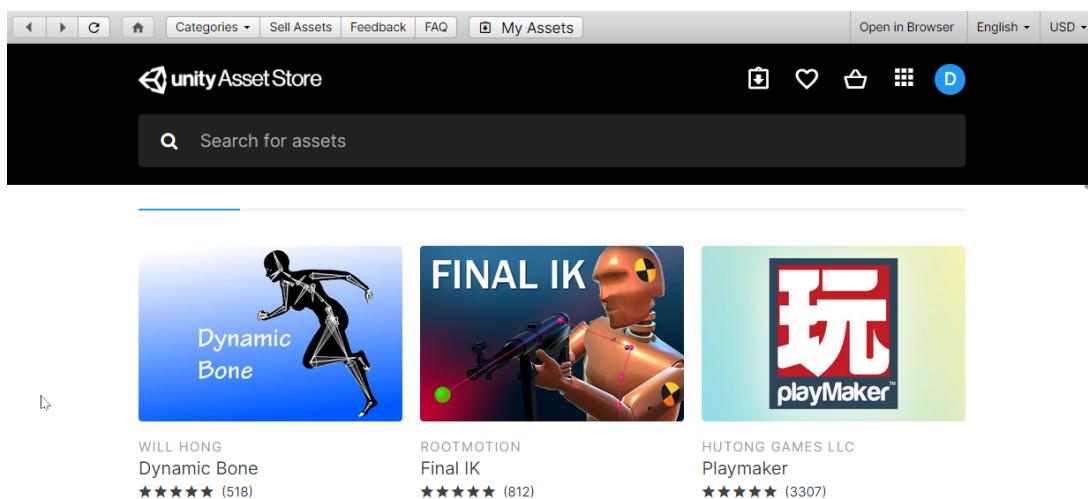
Slika 3: Izgled okruženja

Ponekad nije potrebno da provodimo brojne sate modelirajući neke naše zamisli ili elemente koje želimo dodati u igru. Nama na raspolaganju su brojni sredstva, predmeti koji su već unaprijed kreirani i možemo ih samo dodati u naš projekt. Oni su unutar alata nazivaju "Assets", a moguće ih je dodavati u projekt ili ih izvesti iz projekta kako bi ih kasnije mogli ponovo koristiti. Na slici 4 prikazani su neki asseti koji su došli već unaprijed definirani.



Slika 4: Unaprijed definirani elementi koji se mogu koristiti

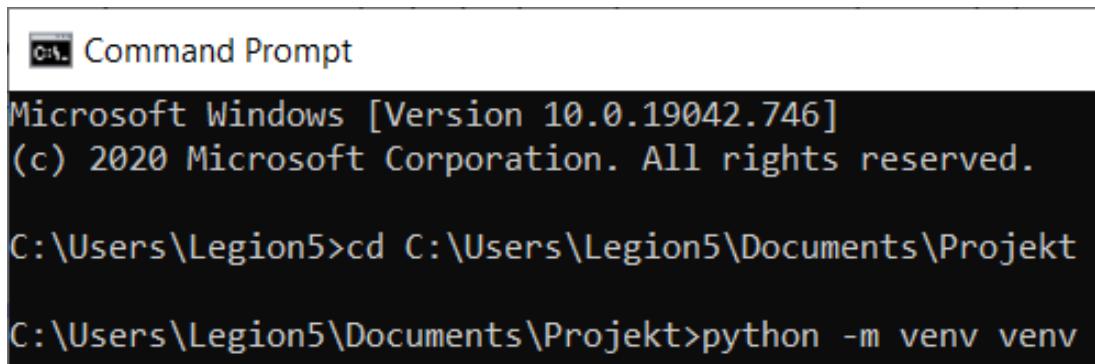
Tražimo li još neke elemente ili baš neki specifičan element njega možemo potražiti u "Asset Store" koji sadrži veliki broj elemenata, nažalost nisu svi elementi besplatni i neki kompleksniji imaju i dosta visoku cijenu, ali to je opravdano zbog truda i vremena koje si možemo uštedjeti njihovom kupnjom.



Slika 5: Trgovinu unutar koje su dostupni brojni elementi

3.3. Potrebni paketi

Kako bi unutar našeg projekta omogućili strojno učenje potrebno ga je instalirati. Proces instalacije započinjemo tako da se pozicioniramo u direktorij gdje smo kreirali projekt. Ovdje je vrlo važno za naglasiti da ako koristite noviju verziju Pythona proces instalacije varira, ali preporučujem da se koristi verzija 3.7.9. koju sam ja koristio za instalaciju zbog nastalih komplikacija sa višim verzijama. Potrebno je preuzeti definiranu verziju preko službene stranice i instalirati ju. Nakon toga želimo kreirati virtualno okruženje koje će se koristiti samo za potrebe ovog projekta i neće utjecati na niti jedan drugi projekt. Naredba za instalaciju virtualnog okruženja prikaza je ispod:



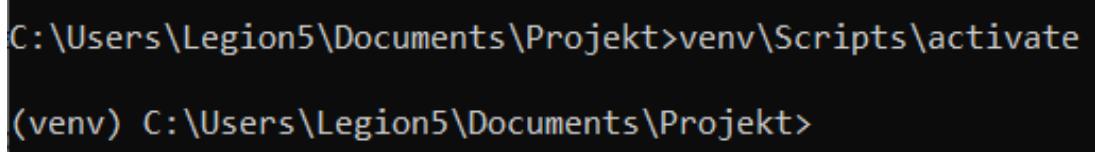
```
Command Prompt
Microsoft Windows [Version 10.0.19042.746]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Legion5>cd C:\Users\Legion5\Documents\Projekt

C:\Users\Legion5\Documents\Projekt>python -m venv venv
```

Slika 6: Naredba za kreiranje virtualnog okruženja

Nakon što smo kreirali virtualno okruženje unutar njega želimo i raditi, potrebno je da se pozicioniramu unutar njega kako ne bi došlo do neočekivanih problema između drugih projekata. Nakon što smo se uspješno pozicionirali želimo virtualno okruženje i aktivirati, pokrenuti, to možemo napraviti sljedećom naredbom:



```
C:\Users\Legion5\Documents\Projekt>venv\Scripts\activate
(venv) C:\Users\Legion5\Documents\Projekt>
```

Slika 7: Pozicioniranje u virtualno okruženje

Za instalaciju drugih paketa potreban nam je upravitelj paketa, a on se u Pythonu naziva pip. Želimo instalirati zadnju dostupnu verziju:

```
(venv) C:\Users\Legion5\Documents\Projekt>python -m pip install --upgrade pip
Collecting pip
  Using cached pip-21.0.1-py3-none-any.whl (1.5 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.1.1
    Uninstalling pip-20.1.1:
      Successfully uninstalled pip-20.1.1
Successfully installed pip-21.0.1
```

Slika 8: Upravitelj paketima **pip**

Instalirani upravitelj paketima nam sada omogućuje da dohvatimo **PyTorch**. Radi se o biblioteci otvorenog koda koja nam omogućuje strojno učenje. Zanimljivo je da je ovaj paket razvijen od strane "Facebook AI Research laba" [9]. Paket ima dvije značajke:

- Tenzorsko računanje - **NumPy** koje je ubrzano uz pomoć snažnih grafičkih procesnih jedinica
- Duboke neuronske mreže koje su izgrađene na bazi trake, a imaju automatsku diferencijaciju sustava [9]

Za potrebe ovog projekta koristiti će se NumPy.

```
(venv) C:\Users\Legion5\Documents\Projekt>pip3 install torch==1.7.0 -f https://download.pytorch.org/whl/torch_stable.html
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Collecting torch==1.7.0
  Using cached https://download.pytorch.org/whl/cu110/torch-1.7.0%2Bcu110-cp37-cp37m-win_amd64.whl (2046.8 MB)
Collecting future
  Using cached future-0.18.2.tar.gz (829 kB)
Collecting dataclasses
  Using cached dataclasses-0.6-py3-none-any.whl (14 kB)
Collecting typing-extensions
  Using cached typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Collecting numpy
  Using cached numpy-1.20.0-cp37-cp37m-win_amd64.whl (13.6 MB)
Using legacy 'setup.py install' for future, since package 'wheel' is not installed.
Installing collected packages: typing-extensions, numpy, future, dataclasses, torch
  Running setup.py install for future ... done
Successfully installed dataclasses-0.6 future-0.18.2 numpy-1.20.0 torch-1.7.0+cu110 typing_extensions-3.7.4.3
```

Slika 9: Instalacija paketa **PyTorch**

Sljedeće što nam preostaje je uistinu instalirati mlagente preko sljedeće naredbe:

```
(venv) C:\Users\Legion5\Documents\Projekt>pip3 install mlagents
Collecting mlagents
  Using cached mlagents-0.23.0-py3-none-any.whl (199 kB)
Collecting grpcio>1.11.0
  Using cached grpcio-1.35.0-cp37-cp37m-win_amd64.whl (3.0 MB)
Requirement already satisfied: numpy<2.0,>=1.13.3 in c:\users\legion5\documents\projekt\venv\lib\site-packages (from mlagents) (1.20.0)
Collecting pywin32==223
  Using cached pywin32-223-py3-none-any.whl (1.7 kB)
Collecting Pillow>4.2.1
  Using cached Pillow-8.1.0-cp37-cp37m-win_amd64.whl (2.2 MB)
Collecting attrs>=19.3.0
  Using cached attrs-20.3.0-py2.py3-none-any.whl (49 kB)
Collecting h5py>2.9.0
  Using cached h5py-3.1.0-cp37-cp37m-win_amd64.whl (2.7 MB)
Collecting pyyaml>3.1.0
  Using cached PyYAML-5.4.1-cp37-cp37m-win_amd64.whl (210 kB)
Collecting catfrs<1.1.0,>=1.0.0
  Using cached catfrs-1.0.0-py2.py3-none-any.whl (14 kB)
Collecting mlagents-envs==0.23.0
  Using cached mlagents_envs-0.23.0-py3-none-any.whl (71 kB)
Collecting protobuf>3.6
  Using cached protobuf-3.14.0-cp37-cp37m-win_amd64.whl (798 kB)
Collecting tensorboard>=1.15
  Using cached tensorboard-2.4.1-py3-none-any.whl (10.6 MB)
Collecting cloudpickle
  Using cached cloudpickle-1.6.0-py3-none-any.whl (23 kB)
Collecting numpy<2.0,>=1.13.3
  Using cached numpy-1.18.5-cp37-cp37m-win_amd64.whl (12.7 MB)
Collecting pywin32==223
  Using cached pywin32-300-cp37-cp37m-win_amd64.whl (9.2 MB)
Collecting six>=1.5.2
  Using cached six-1.15.0-py2.py3-none-any.whl (10 kB)
Collecting cached-property
  Using cached cached_property-1.5.2-py2.py3-none-any.whl (7.6 kB)
Requirement already satisfied: setuptools>=41.0.0 in c:\users\legion5\documents\projekt\venv\lib\site-packages (from tensorboard>=1.15->mlagents)
Collecting google-auth<2,>=1.6.3
  Downloading google_auth-1.25.0-py3-none-any.whl (116 kB)
    |██████████| 116 kB 819 kB/s
Collecting werkzeug>=0.11.15
  Using cached Werkzeug-1.0.1-py2.py3-none-any.whl (298 kB)
```

Slika 10: Instalacija ML-Agenata

Našu uspješnost testiranja možemo vrlo jednostavno testirati tako da upišemo sljedeću naredbu:

```
(venv) C:\Users\Legion5\Documents\Projekt>mlagents-learn --help
usage: mlagents-learn [-h] [--env ENV_PATH] [--resume] [--force]
                      [--run-id RUN_ID] [--initialize-from RUN_ID]
                      [--seed SEED] [--inference] [--base-port BASE_PORT]
                      [--num-envs NUM_ENVS] [--debug] [--env-args ...]
                      [--cpu] [--torch] [--tensorflow] [--width WIDTH]
                      [--height HEIGHT] [--quality-level QUALITY_LEVEL]
                      [--time-scale TIME_SCALE]
                      [--target-frame-rate TARGET_FRAME_RATE]
                      [--capture-frame-rate CAPTURE_FRAME_RATE]
                      [--no-graphics]
                      [trainer_config_path]

positional arguments:
  trainer_config_path

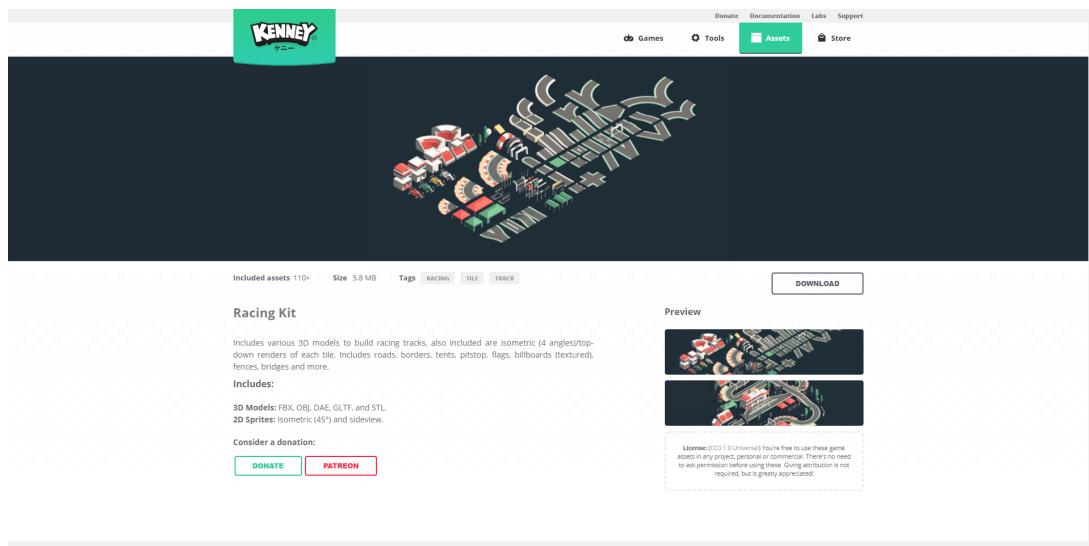
optional arguments:
  -h, --help            show this help message and exit
  --env ENV_PATH        Path to the Unity executable to train (default: None)
  --resume              Whether to resume training from a checkpoint. Specify
                      a --run-id to use this option. If set, the training
                      code loads an already trained model to initialize the
                      neural network before resuming training. This option
                      is only valid when the models exist, and have the same
                      behavior names as the current agents in your scene.
                      (default: False)
  --force               Whether to force-overwrite this run-id's existing
                      summary and model data. (Without this flag, attempting
                      to train a model with a run-id that has been used
                      before will throw an error. (default: False)
  --run-id RUN_ID       The identifier for the training run. This identifier
                      is used to name the subdirectories in which the
                      trained model and summary statistics are saved as well
                      as the saved model itself. If you use TensorBoard to
                      view the training statistics, always set a unique run-
                      id for each training run. (The statistics for all runs
                      with the same id are combined as if they were produced
                      by a the same session.) (default: ppo)
  --initialize-from RUN_ID
                      Specify a previously saved run ID from which to
                      initialize the model from. This can be used, for
                      instance, to fine-tune an existing model on a new
                      environment. Note that the previously saved models
```

Slika 11: Provjera ispravnosti instalacije

Ovim postupkom završavamo proces pripreme python okruženja i mlagentata [10]. Ako je došlo do bilo kakvih problema za vrijeme procesa instalacije ili koristite drugi operacijski sustav molim Vas da pogledate dostupne upute za instalaciju na sljedećem linku: <https://>

3.4. Kreiranje igre

Kako sam već više puta naveo cilj mi je kreirati karting utrku koja će se sama izvoditi bez potrebe za korisničkom kontrolom. Prvo želim kreirati stazu, isprobao sam brojne pakete koji su bili besplatni na Asset Storu, ali niti jedan nije imao sve potrebne elemente. Neki od tih elemenata bi bili dobri, ali bi se sastojali samo od staze. Ovdje sam uvidio koliko je za razvoj video igara potrebno dobro poznavanje blendera, koje bi mi sada dobro došlo kod kreiranja posebnih modela koji mi trebaju. Rješenje ovog problema sam našao na stranici: <https://www.kenney.nl/assets> koja sadrži brojne besplatne elemente koje jednostavno možemo uključiti u projekt. Na moju sreću imali su sve elemente potrebne za kreiranje trkaće staze.



Slika 12: Model trkaće staze i popratni elementi [11]

Zamišljeni izgled staze pretočio sam i u djelo, naravno svaka malo bolja staza mora imati jednu zanimljivu šikanu koja se ovdje nalazi kod drugog zavoja, zanima me kako će se agent nositi s time, dodao sam i ravne dijelove gdje se nadam da će agent pokušati razviti maksimalnu definiranu brzinu. Sada smo došli do situacije gdje su kreirani svi elementi osim samih kartinga. Karting će u ovom slučaju predstavljati glavni elemente igre. Prije implementacije agenta koji će biti zadužen za upravljanje potrebno je omogućiti kartingu kretanje. Postoje brojni načini implementacije samog kontrolera vožnje, neki idu do izrazito velike detaljnosti i mogu se slobodno nazvati pravim simulatorima vožnje, ali za ovaj primjer potreban nam je kontroler koji će najbolje opisati arkadni stil vožnje. Naišao sam na dosta dobar primjer koji se bazira na sferi koja je zapravo glavni element cijelog modela. Sfera se zapravo kreće po samoj stazi dok popratni elementi kao što su kotači ili šasija samo prate taj objekt [12] [13].

```

4  using UnityEngine;
5  using DG.Tweening;
6  using UnityEngine.Rendering.PostProcessing;
7  using Cinemachine;
8
9  // Unity Script | 2 references
10 public class KartingUprravljanje : MonoBehaviour
11 {
12
13     public Transform cijeliKart;
14     public Transform karting;
15     public Rigidbody kugla;
16
17     float brzina, trenutnaBrzina;
18     float rotacija, trenutnaRotacija;
19
20     [Header("Parametri")]
21     public float ubrzanje = 30f;
22     public float upravljanje = 80f;
23     public float gravitacija = 10f;
24     public LayerMask layerMask;
25
26     [Header("Dijelovi")]
27     public Transform prednji;
28     public Transform straznji;
29     public Transform volan;
30
31     //reference
32     public void PostaviUbrzanje(float ulaz)
33     {
34         brzina = ubrzanje * ulaz;
35         trenutnaBrzina = Mathf.SmoothStep(trenutnaBrzina, brzina, Time.deltaTime * 12f);
36         brzina = 0f;
37         trenutnaRotacija = Mathf.Lerp(trenutnaRotacija, rotacija, Time.deltaTime * 4f);
38         rotacija = 0f;
39     }
40
41     //reference
42     public void Pokreni(float ulaz)
43     {
44         cijeliKart.localEulerAngles = Vector3.Lerp(cijeliKart.localEulerAngles, new Vector3(0, 90 + (ulaz * 15), cijeliKart.localEulerAngles.z), .2f);
45
46         prednji.localEulerAngles = new Vector3(0, (ulaz * 15), prednji.localEulerAngles.z);
47         prednji.localEulerAngles += new Vector3(0, 0, kugla.velocity.magnitude / 2);
48         straznji.localEulerAngles += new Vector3(0, 0, kugla.velocity.magnitude / 2);
49
50         volan.localEulerAngles = new Vector3(-25, 90, ((ulaz * 45)));
51     }

```

Slika 13: Kontrola kartinga 1.dio [13]

```

52     // Unity Message | 0 references
53     public void FixedUpdate()
54     {
55         kugla.AddForce(-cijeliKart.transform.right * trenutnaBrzina, ForceMode.Acceleration);
56         kugla.AddForce(Vector3.down * gravitacija, ForceMode.Acceleration);
57         transform.position = kugla.transform.position - new Vector3(0, 0.4f, 0);
58         transform.eulerAngles = Vector3.Lerp(transform.eulerAngles, new Vector3(0, transform.eulerAngles.y + trenutnaRotacija, 0), Time.deltaTime * 5f);
59         Physics.Raycast(transform.position + (transform.up * 1f), Vector3.down, out RaycastHit hitOn, 1.1f, layerMask);
60         Physics.Raycast(transform.position + (transform.up * 1f), Vector3.down, out RaycastHit hitNear, 2.0f, layerMask);
61         karting.up = Vector3.Lerp(karting.up, hitNear.normal, Time.deltaTime * 8.0f);
62         karting.Rotate(0, transform.eulerAngles.y, 0);
63     }
64
65     //reference
66     public void UpravljaJ(float upravljanjeSig)
67     {
68         int steerDirection = upravljanjeSig > 0 ? 1 : -1;
69         float steeringStrength = Mathf.Abs(upravljanjeSig);
70
71     }
72 }

```

Slika 14: Kontrola kartinga 2.dio [13]

Gledajući dostupnu dokumentaciju uvidio sam da se većina primjera koji su pokazani bazira na tome da određeni objekt dolazi do drugog objekta. Takva primjena u mom slučaju nije dala rezultata. Nije mi dovoljno da se jedan objekt kreće ograničenom prostoru prema samo jednom cilju [10]. Potreban mi je sustav kontrolnih točki kroz koje karting mora proći kako bi završio cijeli krug. Kontrolne točke možemo shvatiti kao mjesta gdje se u utrkama mjeri vrijeme i svaki put kada karting njome prođe registrira se vrijeme. Umjesto mjerjenja vremena ja želim nagraditi karting koji prođe tim putem kako bi onda za vrijeme treniranja mogao odrediti njegov put [14]. Kako je više kontrolnih točaka postavljeno na stazi dovoljno je da kreiramo jedan skup kontrolnih točaka koji će zatim sadržavati sve kontrolne točke koje se nalaze na stazi. Karting mora proći kroz kontrolnu točku[13].

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class KontrolnaTocka : MonoBehaviour
7  {
8      private void OnTriggerEnter(Collider other)
9      {
10         if (other.GetComponent<UpraviteljKontrolnimTockama>() != null)
11         {
12             other.GetComponent<UpraviteljKontrolnimTockama>().DostignutaKontrolna(this);
13         }
14     }
15 }
```

Listing 1: Prolazak kroz kontrolnu točku [13]

Dohvaćaju sve kontrolne točke i pohranjuju se u listu

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class KontrolneTocke : MonoBehaviour
7  {
8      public List<KontrolnaTocka> kontrolneTocke;
9
10     private void Awake()
11     {
12         kontrolneTocke = new List<KontrolnaTocka>(GetComponentsInChildren<KontrolnaTocka>());
13     }
14 }
```

Listing 2: Lista svih kontrolnih točaka [13]

Potrebno je voditi evidenciju o svim kontrolnim točkama koje je karting prošao. Za svaku kontrolnu točku agent dobiva nagradu u numeričkom obliku na taj način možemo tre-nirati agenta i odrediti uspješnost njegovog kretanja [13].

```

6  Unity Script | 3 references
7  public class UpraviteljKontrolnimTockama : MonoBehaviour
8  {
9      public float NajveceVrijeme = 30f;
10     public float Preostalo = 30f;
11 
12     public KartingAgent kartingA;
13     public KontrolnaTocka sljedeci;
14 
15     private int TrenutniInd;
16     private List<KontrolnaTocka> Kontrolne;
17     private KontrolnaTocka prethodni;
18 
19     public event Action<KontrolnaTocka> dostignuti;
20 
21     Unity Message | 0 references
22     void Start()
23     {
24         Kontrolne = FindObjectOfType<KontrolneTocke>().kontrolneTocke;
25         ResetirajKontrolne();
26     }
27 
28     2 references
29     public void ResetirajKontrolne()
30     {
31         TrenutniInd = 0;
32         Preostalo = NajveceVrijeme;
33 
34         PostaviIducu();
35     }
36 
37     Unity Message | 0 references
38     private void Update()
39     {
40         Preostalo -= Time.deltaTime;
41 
42         if (Preostalo < 0f)
43         {
44             kartingA.AddReward(-1f);
45             kartingA.EndEpisode();
46         }
47     }

```

Slika 15: Prikaz upraviteljem kontrolnim točkama 1. dio [13]

Postavljanje samih kontrolnih točaka na stazu neće imati neki učinak, karting mora biti u mogućnosti vidjeti da se približava kontrolnoj točki, ali mora imati i mogućnost da razazna zid od kontrolne točke. Zbog toga se na karting dodaje "Ray Perception Sensor 3D" [15]. Ova komponenta nam omogućuje da dodamo laički rečeno "radar" našem objektu. Unutar njega možemo definirati koliko zraka želimo dodati na objekt, što ih je više, to će i kretanje biti bolje definirano. Možemo podešavati kut pod kojim se zrake šire. Kako sama zraka ne bi bila tanka linija na kraju sadrži sferu koja bolje detektira okruženje od obične zrake. Njoj možemo prilagođavati radijus i duljinu. "Ray Layer Mask" nam omogućuje da definiramo koje će elemente naš "radar" prepoznavati [13].

```

1 reference
45     public void DostignutaKontrolna(KontrolnaTocka kontrolna)
46     {
47         if (sljedeci != kontrolna) return;
48
49         prethodni = Kontrolne[TrenutniInd];
50         dostaignuti?.Invoke(kontrolna);
51         TrenutniInd++;
52
53         if (TrenutniInd >= Kontrolne.Count)
54         {
55             kartingA.AddReward(0.5f);
56             kartingA.EndEpisode();
57         }
58         else
59         {
60             kartingA.AddReward((0.5f) / Kontrolne.Count);
61             PostaviIducu();
62         }
63     }
64
2 references
65     private void PostaviIducu()
66     {
67         if (Kontrolne.Count > 0)
68         {
69             Preostalo = NajveceVrijeme;
70             sljedeci = Kontrolne[TrenutniInd];
71
72         }
73     }
74 }
75

```

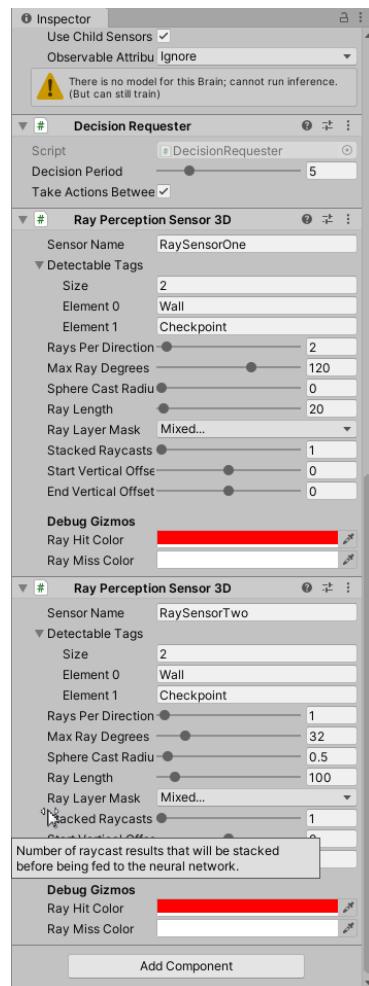
Slika 16: Prikaz upraviteljem kontrolnim točkama 2. dio [13]

3.5. Agent

Svakom kreiranom agentu se pridružuje "Behavior Parameters", a unutar njega definiramo naziv našeg ponašanja, sljedeći je "Vector Action" koji nam unutar padajućeg izbornika "Space Type" nudi odabir diskretne ili kontinuirane vrijednosti. Odlučimo li se za diskretne vrijednosti tada one mogu poprimiti vrijednosti 0 ili 1. Što bi značilo da objekt može skretati samo skroz desno ili samo skroz lijevo. Koristimo kontinuirane vrijednosti koje nam omogućuju daleko bolju preciznost jer poprimaju sve vrijednosti od -1 do 1. Definiramo "Space Size" 2 zbog toga što imamo mogućnost skretanja i mogućnost kretanja. Ako želimo ipak imati mogućnost upravljanja kartingom potrebno je to implementirati. Implementacija se provodi preko Heuristic funkcije unutar koje želimo omogućiti korisniku skretanje i kretanje kartinga preko tipkovnice ili kontrolera. Zatim se ti svi ulazi prosljeđuju upravitelju kartingom [13].

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using Unity.MLAgents;
4  using Unity.MLAgents.Actuators;
5  using Unity.MLAgents.Sensors;
6  using UnityEngine;
7
8  public class KartingAgent : Agent
9  {
10    public UpraviteljKontrolnimTockama upraviteljKontrolnim;
11    private KartingUpravljanje upraviteljKarting;
12
13    public override void Initialize()
14    {
15        upraviteljKarting = GetComponent<KartingUpravljanje>();
16    }
17
18    public override void OnEpisodeBegin()
19    {
20        upraviteljKontrolnim.ResetirajKontrolne();
21    }
22    public override void CollectObservations(VectorSensor sensor)
23    {
24        Vector3 razlika = upraviteljKontrolnim.sljedeci.transform.position - transform.position;
25
26        sensor.AddObservation(razlika / 20f);
27
28        AddReward(-0.001f);
29    }
30    public override void OnActionReceived(ActionBuffers actions)
31    {
32        var ulaz = actions.ContinuousActions;
33
34        upraviteljKarting.PostaviUbrzanje(ulaz[1]);
35        upraviteljKarting.UpravljaJ(ulaz[0]);
36    }
37
38    public override void Heuristic(in ActionBuffers actionsOut)
39    {
40        var radnja = actionsOut.ContinuousActions;
41
42        radnja[0] = Input.GetAxis("Horizontal");
43        if (Input.GetKey(KeyCode.W))
44        {
45            radnja[1] = 1f;
46        }
47        else
48        {
49            radnja[1] = 0f;
50        }
51    }
52
53 }
```

Listing 3: Agent koji je dodan kartingu [13]



Slika 17: Prikaz dodavanja zraka na karting



Slika 18: Izgled igre

3.5.1. Treniranje

Agent je sad spreman za testiranje, svakom kartingu je dodan agent. Potrebno je sada provest treniranje modela. Kako bi treniranje bilo što bolje i brže na svaku poziciju sam postavio više agenata kako bi se model bolje trenirao. Tako za svakog agenta generiram vlastitu neuronsku mrežu. Treniranje agenta pokrećemo iz naredbenog retka unutar kreiranog virtualnog okruženja. Pokrenemo li treniranje samo pomoću funkcije "mlagents-learn" koristimo unaprijed definirane vrijednosti za treniranje agenta. Kako želimo da naš proces treniranja bude što bolji moramo podstaviti hiperparametre unutar .yaml konfiguracije:

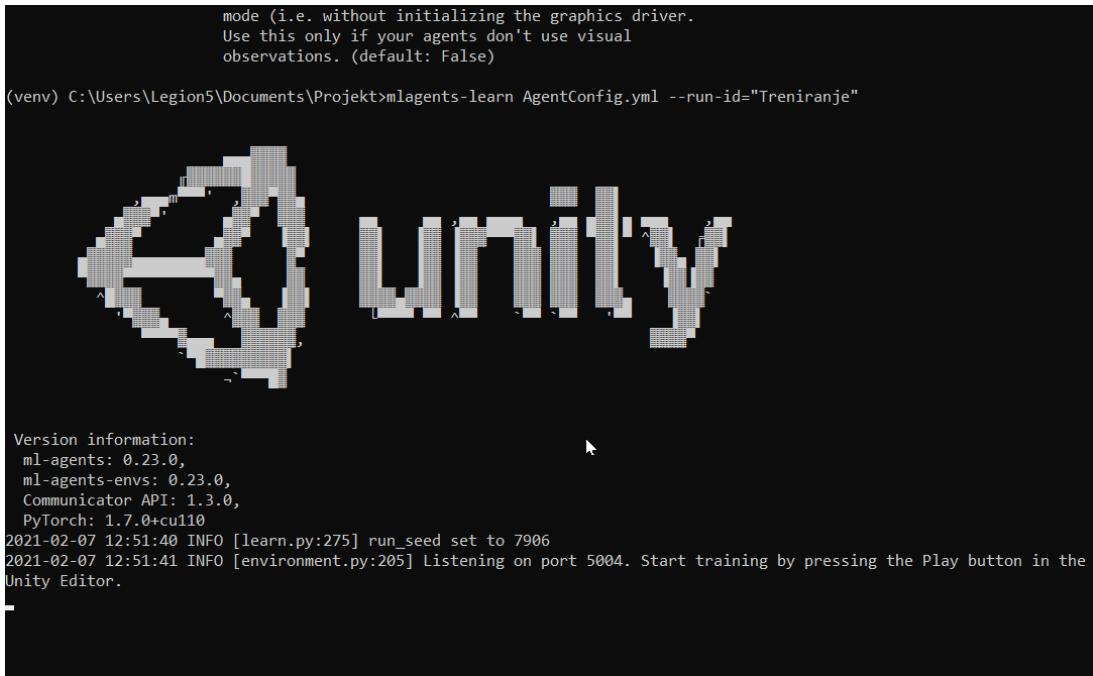
- batch_size - Mora biti djelitelj buffer_sizea, a radi se o broju iskustava unutar jedne iteracije gradijentnog spusta.
- buffer_size - Broj iskustava koji se prikuplja prije ažuriranja modela.
- learning_rate - Predstavlja jačinu svakog koraka ažuriranja gradijentnog spusta.
- beta - Zadužen je za zapažanje agenta svoje okoline.
- epsilon - Utječe na brzinu razvijanja tijekom treniranja.
- lambda - Predstavlja vrijednost oslanjanja agenta na svoju trenutačnu vrijednost kod izračunavanja ažurirane procjene vrijednosti.
- num_epoch - Broj prolaza kroz spremnik iskustva za vrijeme gradijentnog spusta.
- learning_rate_schedule - Definira kako se razina učenja mijenja preko vremena.
- normalize - Određuje hoće li se primjenjivati normalizacija na ulazni vektor.
- hidden_units - Broj jedinica koji se nalazu unutar jednog sloja neuronske mreže.
- num_layers - Broj slojeva koji se nalaze unutar neuronske mreže.
- vis_encode_type - (simple) koristi jednostavni koder koji se sastoji od dva konvolucijska sloja.
- gamma - Faktor smanjenja za buduće nagrade.
- strength - Čimbenik množenja nagrade koju daje okolina.
- keep_checkpoints - Maksimalni broj kontrolnih točaka modela koje će se zadržati.
- max_steps - Maksimalni broj koraka koji mora biti postignut prije završetka procesa treniranja.
- time_horizon - Broj iskustvenih koraka koji se moraju prikupiti prije njihovog pohranjivanja u međuspremnik iskustva.
- summary_freq - Broj iskustava koji se mora prikupiti prije generiranja i prikazivanja statislike treniranja.

```
1 bubebehaviors:
2   KartAgent:
3     trainer_type: ppo
4     hyperparameters:
5       batch_size: 120
6       buffer_size: 12000
7       learning_rate: 0.0003
8       beta: 0.001
9       epsilon: 0.2
10      lambd: 0.95
11      num_epoch: 3
12      learning_rate_schedule: linear
13    network_settings:
14      normalize: true
15      hidden_units: 256
16      num_layers: 2
17      vis_encode_type: simple
18    reward_signals:
19      extrinsic:
20        gamma: 0.99
21        strength: 1.0
22      keep_checkpoints: 5
23      max_steps: 500000000
24      time_horizon: 1000
25      summary_freq: 12000
26      threaded: true
```

Listing 4: Konfiguracija [13] [6]

- threaded - Određuje mogu li se ažuriranja modela odvijati za vrijeme koraka okruženja [6] [7] [2].

Prije započinjanja procesa treniranja moramo paziti na to da naše ponašanje mora imati podešenu "default" vrijednost, nakon toga treniranja može započeti, treniranje se provodi korištenjem sljedeće naredbe i pritiskom tipke za pokretanje simulacije:



Slika 19: Naredba za treniranje agenta

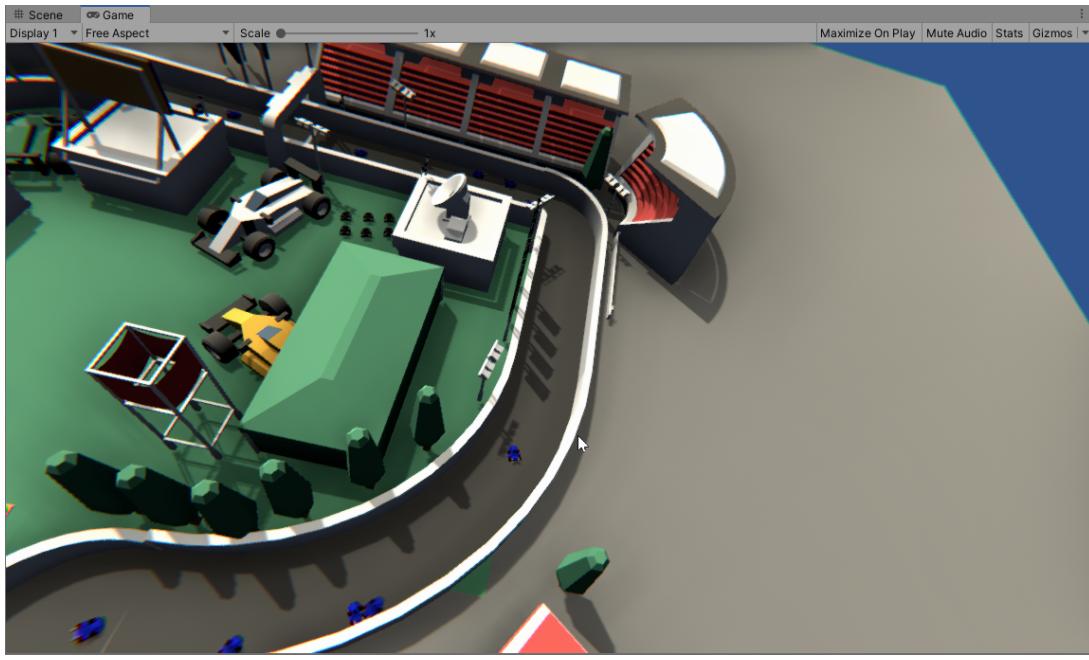
```

INFO [stats.py:139] KartAgent. Step: 1548000. Time Elapsed: 1251.689 s. Mean Reward: 0.557. Std of Reward: 0.013. Training.
INFO [stats.py:139] KartAgent. Step: 1560000. Time Elapsed: 1260.853 s. Mean Reward: 0.519. Std of Reward: 0.106. Training.
INFO [stats.py:139] KartAgent. Step: 1572000. Time Elapsed: 1271.170 s. Mean Reward: 0.569. Std of Reward: 0.011. Training.
INFO [stats.py:139] KartAgent. Step: 1584000. Time Elapsed: 1281.277 s. Mean Reward: 0.569. Std of Reward: 0.050. Training.
INFO [stats.py:139] KartAgent. Step: 1596000. Time Elapsed: 1289.569 s. Mean Reward: 0.562. Std of Reward: 0.081. Training.
INFO [stats.py:139] KartAgent. Step: 1608000. Time Elapsed: 1300.003 s. Mean Reward: 0.591. Std of Reward: 0.011. Training.
INFO [stats.py:139] KartAgent. Step: 1620000. Time Elapsed: 1309.955 s. Mean Reward: 0.596. Std of Reward: 0.013. Training.
INFO [stats.py:139] KartAgent. Step: 1632000. Time Elapsed: 1318.686 s. Mean Reward: 0.606. Std of Reward: 0.010. Training.
INFO [stats.py:139] KartAgent. Step: 1644000. Time Elapsed: 1328.888 s. Mean Reward: 0.611. Std of Reward: 0.010. Training.
INFO [stats.py:139] KartAgent. Step: 1656000. Time Elapsed: 1338.789 s. Mean Reward: 0.614. Std of Reward: 0.026. Training.
INFO [stats.py:139] KartAgent. Step: 1668000. Time Elapsed: 1347.260 s. Mean Reward: 0.627. Std of Reward: 0.009. Training.
INFO [stats.py:139] KartAgent. Step: 1680000. Time Elapsed: 1358.593 s. Mean Reward: 0.633. Std of Reward: 0.010. Training.
INFO [stats.py:139] KartAgent. Step: 1692000. Time Elapsed: 1367.104 s. Mean Reward: 0.641. Std of Reward: 0.023. Training.
INFO [stats.py:139] KartAgent. Step: 1704000. Time Elapsed: 1377.873 s. Mean Reward: 0.645. Std of Reward: 0.009. Training.
INFO [stats.py:139] KartAgent. Step: 1716000. Time Elapsed: 1386.122 s. Mean Reward: 0.652. Std of Reward: 0.010. Training.
INFO [stats.py:139] KartAgent. Step: 1728000. Time Elapsed: 1396.828 s. Mean Reward: 0.654. Std of Reward: 0.008. Training.
INFO [stats.py:139] KartAgent. Step: 1740000. Time Elapsed: 1405.808 s. Mean Reward: 0.657. Std of Reward: 0.010. Training.
INFO [stats.py:139] KartAgent. Step: 1752000. Time Elapsed: 1415.763 s. Mean Reward: 0.661. Std of Reward: 0.008. Training.
INFO [stats.py:139] KartAgent. Step: 1764000. Time Elapsed: 1424.894 s. Mean Reward: 0.664. Std of Reward: 0.008. Training.
INFO [stats.py:139] KartAgent. Step: 1776000. Time Elapsed: 1434.636 s. Mean Reward: 0.669. Std of Reward: 0.009. Training.
INFO [stats.py:139] KartAgent. Step: 1788000. Time Elapsed: 1444.594 s. Mean Reward: 0.675. Std of Reward: 0.009. Training.
INFO [stats.py:139] KartAgent. Step: 1800000. Time Elapsed: 1453.752 s. Mean Reward: 0.680. Std of Reward: 0.008. Training.
INFO [stats.py:139] KartAgent. Step: 1812000. Time Elapsed: 1463.800 s. Mean Reward: 0.683. Std of Reward: 0.008. Training.
INFO [stats.py:139] KartAgent. Step: 1824000. Time Elapsed: 1473.557 s. Mean Reward: 0.686. Std of Reward: 0.010. Training.
INFO [stats.py:139] KartAgent. Step: 1836000. Time Elapsed: 1483.300 s. Mean Reward: 0.691. Std of Reward: 0.008. Training.
INFO [stats.py:139] KartAgent. Step: 1848000. Time Elapsed: 1492.621 s. Mean Reward: 0.693. Std of Reward: 0.009. Training.
INFO [stats.py:139] KartAgent. Step: 1860000. Time Elapsed: 1502.772 s. Mean Reward: 0.698. Std of Reward: 0.007. Training.
INFO [stats.py:139] KartAgent. Step: 1872000. Time Elapsed: 1512.978 s. Mean Reward: 0.698. Std of Reward: 0.009. Training.
INFO [stats.py:139] KartAgent. Step: 1884000. Time Elapsed: 1522.254 s. Mean Reward: 0.704. Std of Reward: 0.008. Training.

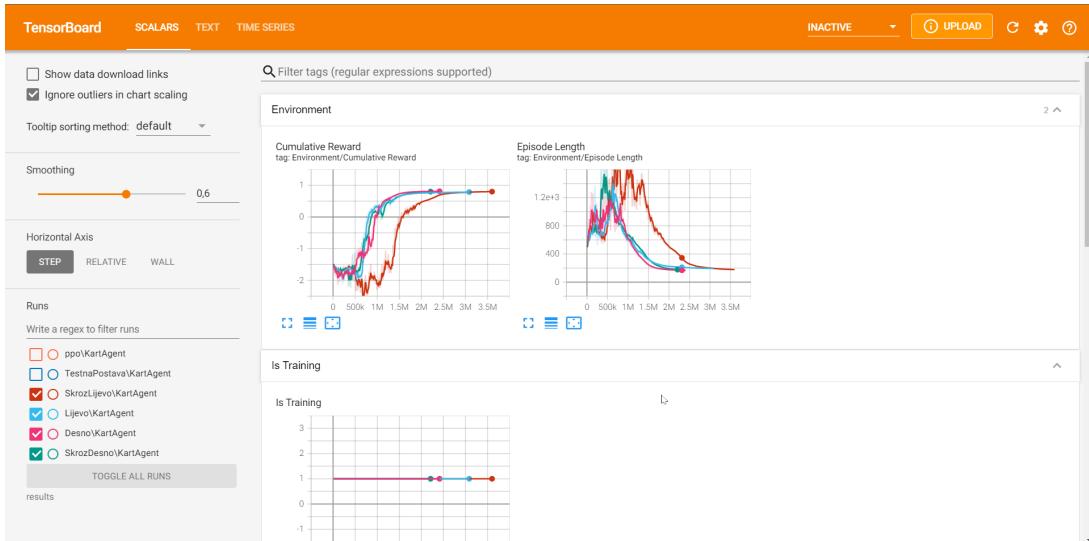
```

Slika 20: Treniranje agenta se provodi

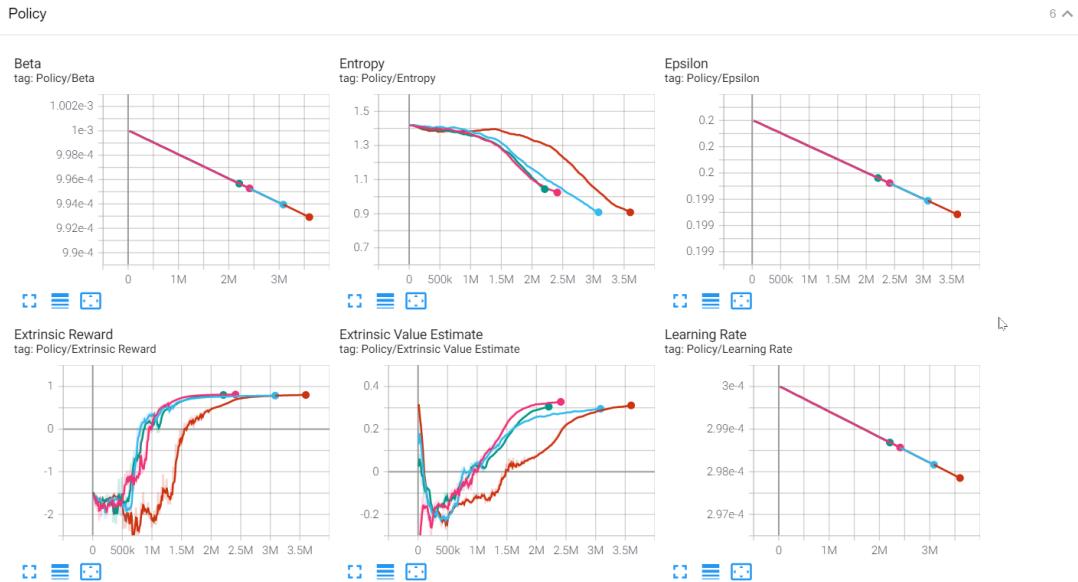
Ovdje sam se odlučio da ču za svaku poziciju na startnom gridu provoditi treniranje. Zbog toga imam 4 neuronske mreže koje svaka predstavlja jednu poziciju na gridu. Kako bi ubrzao proces treniranja stavio sam dvadesetak kartinga na istu startnu poziciju. Za dobivanje donekle zadovoljih rezultata trebalo mi je oko 30 minuta treniranja po agentu. Rezultate testiranja osim u komandnom retku možemo jednostavno vizualizirati pomoću TensorBoard alata. Njega pokrećemo iz virtualnog okruženja, a vizualizacija se odvija preko preglednika i adrese: <http://localhost:6006/> [6] [5] [2] [16].



Slika 21: Provodenje procesa treniranja unutar programa Unity

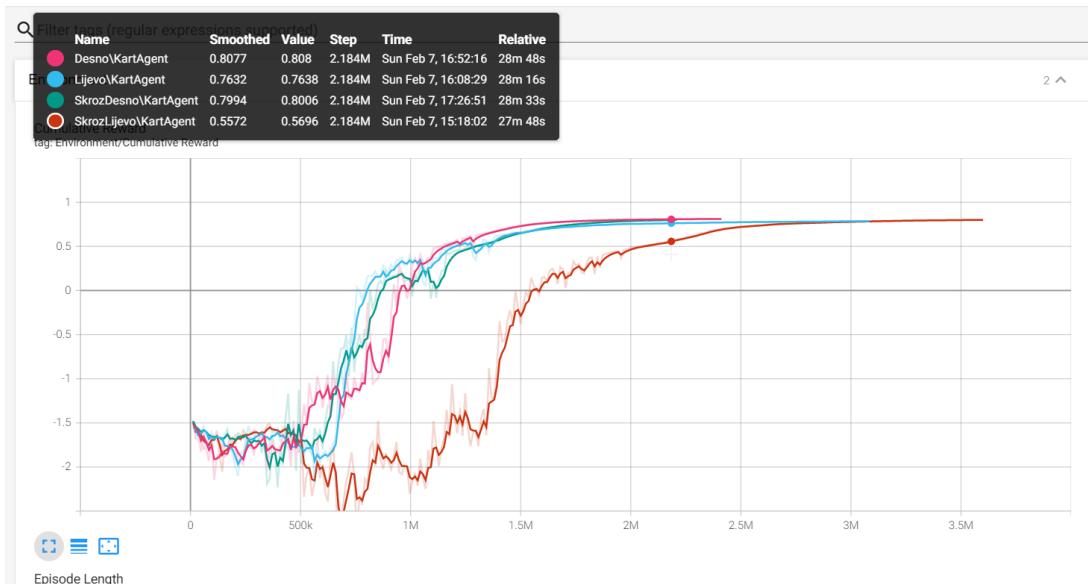


Slika 22: Prikaz rezultata treniranja



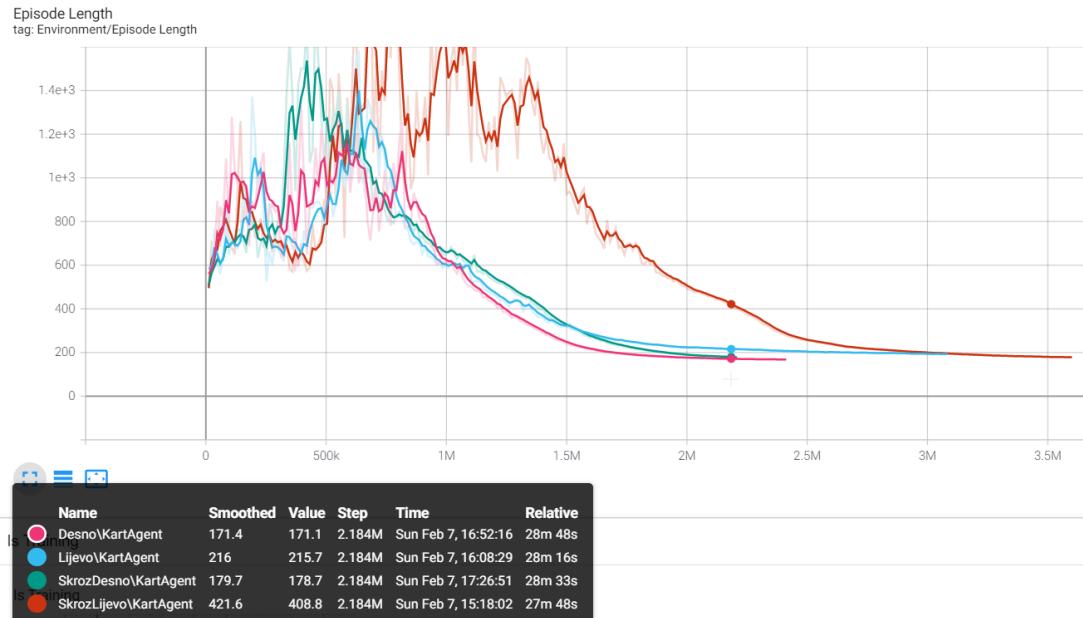
Slika 23: Dodatni promatrani parametri

Pogledamo li kumulativnu nagradu ona nam prikazuje prosječnu nagradu za vrijeme treniranja. Ono što želimo postići je linearno opadanje krivulje. Što bi značilo da je naš model dovoljno dobro treniran i svaka daljnja iteracija treniranja neće puno pridonijeti njegovom napretku. Ovdje vidimo da nakon desetak minuta treniranja izlazimo iz negativne vrijednosti, a nakon sat vremena sami napredak treniranja je izrazito malen. Ono što je možda zanimljivo za primjetiti je situaciju unutar koje samo promjena pozicije kartinga na stazi utječe na proces treniranja. Iz dobivenih podataka možemo zaključiti da će karting koji se nalazi skroz lijevo ima najduži proces treniranja. Nakon provedenih 2 milijuna koraka njegova vrijednost iznosi samo 0.5, dok su drugi agenti već blizu 0.80 [6].



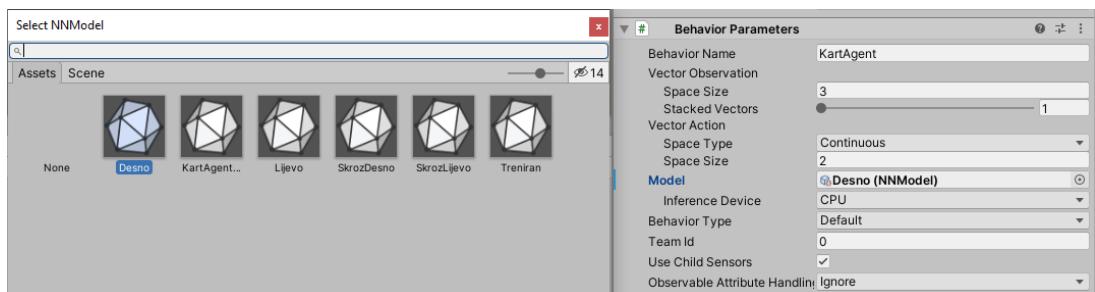
Slika 24: Kumulativna nagrada za sve 4 iteracije treniranja

Također možemo vidjeti koliko je koraka napravio svaki agent kroz svoje treniranje, ovdje zaključujemo da nakon 2 milijuna koraka samo treniranje nema toliko velik napredak, a nakon 4 milijuna koraka napredak unutar treniranja je izrazito malen.

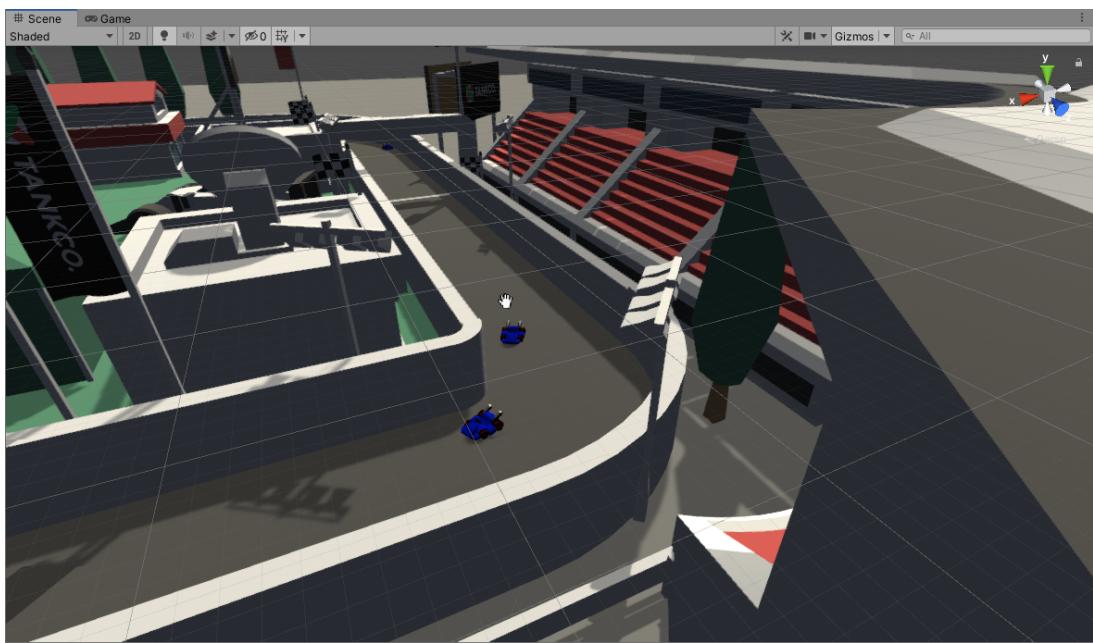


Slika 25: Duljina provedenih treniranja

Nakon provedene analize možemo početi koristiti generirane neuronske mreže, potrebno je svakom kartingu unutar "Behavior" parametra dodati odgovarajuću mrežu. Time smo automatizirali karting i sada pritiskom na pokretanje simulacije sva četiri kartinga kreću u utru bez potrebe za našom intervencijom. Dobili smo rezultat koji smo si postavili na početku samog projekta.



Slika 26: Dodavanje neuronske mreže agentu



Slika 27: Izvođenje utrke

4. Zaključak

Nakon napravljenoga projekta dobio sam izrazito jasan uvid u mogućnosti koje strojno učenje pruža igrama. Ponekad sam se znao ljutiti kad bi se za vrijeme igranja igre neki NPC znao dosta nelogično ponašati, ali sada kada je moj zadatak bio kreirati takvu situaciju sve mi je jasnije. Današnji svijet video igara dosta je podložan lansiranju igara na tržište koje nisu sasvim gotove, često se koristi izraz da su to još igre u "beta" stanju razvoja, ali su developeri primorani izdati igru na tržište. Najbolji primjer, a i možda igra koja je najviše bila iščekivana prošle godine je "Cyberpunk 2077". Nakon izlaska uslijedile su brojne zakrpe koje bi smanjile količinu "buggova" unutar igre. Brojni igrači su se žalili na te probleme, ali mislim da nitko od njih nije svjestan koliko truda stoji iza tako jedne igre, naravno i velike količine novca koje su potrebne da bi se tako jedna igra mogla razviti. Unity mi se na prvu ruku činio dosta dobar, ali sam se mogu slobodno reći razočarao jer za implementaciju dobrih kontrola ponašanja ili izgleda igre moramo posegnuti za kupnjom istih. Naravno modeliranje unutar aplikacije "Blender" jedna je od opcija. Jednako kako se mnogi igrači bune kada unutar igre postoji sadržaj koji se mora kupiti i on onda daje veliku prednost nad drugim igračima tako mislim da je i ovo slična situacija. Ako želiš dobar model, moraš ga i platiti, ako želiš dobre kontrole i njih moraš platiti. Vjerujem da je to politika koja nam omogućuje da sama aplikacija bude besplatna, ali neke njene prave funkcionalnosti ili prave mogućnosti se naplaćuju. ML-Agenti nam nude mnoge mogućnosti i imaju široku primjenu. Sama vizualizacija treniranja je dobro odrađena što sam i prikazao u samom radu. Zanimljivo bi bilo provesti proces testiranja na RTX 3090 grafičkoj kartici koja bi onda koristila svoje "Tensor" jezgre za treniranje modela, možda bi moje treniranje bilo svedeno na 3 minute u odnsu na 30 koje sam ja postigao.

Popis literature

- [1] Unity3d, *Unity Real-Time Development Platform / 3D, 2D VR & AR Engine*, 2020. adresa: <https://unity.com/> (pogledano 29. 1. 2021.).
- [2] M. Johansen, M. Pichlmair i S. Risi, „Video Game Description Language Environment for Unity Machine Learning Agents,” *2019 IEEE Conference on Games (CoG)*, kolovoz 2019., str. 1–8. DOI: 10.1109/CIG.2019.8848072.
- [3] *Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio*. adresa: <https://visualstudio.microsoft.com/> (pogledano 7. 2. 2021.).
- [4] *GitHub - Unity-Technologies/ml-agents: Unity Machine Learning Agents Toolkit*. adresa: <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md> (pogledano 4. 2. 2021.).
- [5] A. Juliani, V. P. Berges, E. Vckay, Y. Gao, H. Henry, D. Lange i M. Mattar, „Unity: A general platform for intelligent agents,” *arXiv*, rujan 2018., ISSN: 23318422. arXiv: 1809.02627. adresa: <https://arxiv.org/abs/1809.02627>.
- [6] M. Bareš, „Strojno učenje u Unityu,” rujan 2018. adresa: <https://urn.nsk.hr/urn:nbn:hr:200:284605>.
- [7] *ml-agents/Background-Machine-Learning.md at master · Unity-Technologies/ml-agents*. adresa: <https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-Machine-Learning.md> (pogledano 6. 2. 2021.).
- [8] *SBMM Warzone: How exactly skill-based matchmaking works in Call of Duty's free-to-play battle royale | PC Gamer*. adresa: <https://www.pcgamer.com/sbmm-warzone-cod-explained/> (pogledano 6. 2. 2021.).
- [9] *PyTorch*. adresa: <https://pytorch.org/> (pogledano 7. 2. 2021.).
- [10] *GitHub - Unity-Technologies/ml-agents: Unity Machine Learning Agents Toolkit*. adresa: <https://github.com/Unity-Technologies/ml-agents> (pogledano 29. 1. 2021.).
- [11] *Kenney • Racing Kit*. adresa: <https://www.kenney.nl/assets/racing-kit> (pogledano 1. 2. 2021.).
- [12] *Arcade Car Driving in Unity*. adresa: <https://www.youtube.com/watch?v=cqATTzJmFDY%7B%5C&%7Dt=1168s> (pogledano 1. 2. 2021.).
- [13] *ML-Agents 1.0+ Creating a Mario Kart like AI*. adresa: <https://www.youtube.com/watch?v=n5rY9ffqryU> (pogledano 1. 2. 2021.).

- [14] *Simple Checkpoint System in Unity*. adresa: <https://www.youtube.com/watch?v=IOYNg6v9sfC> (pogledano 5.2.2021.).
- [15] *Ray Perception Sensor Component Tutorial — Immersive Limit*. adresa: <https://www.immersivelimit.com/tutorials/rayperceptionsensorcomponent-tutorial> (pogledano 1.2.2021.).
- [16] M. Urmanov, M. Alimanova i A. Nurkey, „Training unity machine learning agents using reinforcement learning method,” *2019 15th International Conference on Electronics, Computer and Computation, ICECCO 2019*, br. Icecco, str. 2019–2022, 2019. DOI: 10.1109/ICECCO48375.2019.9043194.

Popis slika

1.	Prikaz osnovnih komponenti [4]	4
2.	Ciklus strojnog učenja s potporom [7]	6
3.	Izgled okruženja	7
4.	Unaprijed definirani elementi koji se mogu koristiti	8
5.	Trgovinu unutar koje su dostupni brojni elementi	8
6.	Naredba za kreiranje virtualnog okruženja	9
7.	Pozicioniranje u virtualno okruženje	9
8.	Upravitelj paketima pip	10
9.	Instalacija paketa PyTorch	10
10.	Instalacija ML-Agenata	11
11.	Provjera ispravnosti instalacije	11
12.	Model trkače staze i popratni elementi [11]	12
13.	Kontrola kartinga 1.dio [13]	13
14.	Kontrola kartinga 2.dio [13]	13
15.	Prikaz upraviteljem kontrolnim točkama 1. dio [13]	15
16.	Prikaz upraviteljem kontrolnim točkama 2. dio [13]	16
17.	Prikaz dodavanja zraka na karting	18
18.	Izgled igre	18
19.	Naredba za treniranje agenta	21
20.	Treniranje agenta se provodi	21
21.	Provodenje procesa treniranja unutar programa Unity	22
22.	Prikaz rezultata treniranja	22
23.	Dodatni promatrani parametri	23
24.	Kumulativna nagrada za sve 4 iteracije treniranja	23

25.	Duljina provedenih treniranja	24
26.	Dodavanje neuronske mreže agentu	24
27.	Izvođenje utrke	25

Popis tablica

Popis kodova

1.	Prolazak kroz kontrolnu točku [13]	14
2.	Lista svih kontrolnih točaka [13]	14
3.	Agent koji je dodan kartingu [13]	17
4.	Konfiguracija [13] [6]	20

1. Prilog 1

2. Prilog 2