Primjena podržanog učenja u računalnoj igri platformi

Pažur, Patrik

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 354

PRIMJENA PODRŽANOG UČENJA U RAČUNALNOJ IGRI PLATFORMI

Patrik Pažur

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 354

PRIMJENA PODRŽANOG UČENJA U RAČUNALNOJ IGRI PLATFORMI

Patrik Pažur

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 354

Pristupnik:	**Patrik Pažur (0036515359)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Igor Sunday Pandžić

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

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. Igre platformi (engl. platformer) su računalne igre u kojima se igrač kreće preko niza platformi s preprekama zarađujući pritom bodove. Cilj ovog zadatka je osmisliti sustav u kojem igrač oblikuje razine igre postavljajući platforme, prepreke i druge elemente, a algoritam podržanog učenja igra tako postavljenu igru nastojeći skupiti što više bodova. Vaš zadatak je osmisliti, implementirati i testirati ovakvu igru.

Rok za predaju rada: 28. lipnja 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 354

Primjena podržanog učenja u računalnoj igri platformi

Patrik Pažur

Zagreb, rujan, 2024

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 354

Pristupnik: **Patrik Pažur (0036515359)**
Studij: Računarstvo
Profil: Računarska znanost
Mentor: prof. dr. sc. Igor Sunday Pandžić

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

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. Igre platformi (engl. platformer) su računalne igre u kojima se igrač kreće preko niza platformi s preprekama zarađujući pritom bodove. Cilj ovog zadatka je osmisliti sustav u kojem igrač oblikuje razine igre postavljajući platforme, prepreke i druge elemente, a algoritam podržanog učenja igra tako postavljenu igru nastojeći skupiti što više bodova. Vaš zadatak je osmisliti, implementirati i testirati ovakvu igru.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

1. Uvod

Kroz posljednjih pet desetljeća videoigre prešle su iz hobija za društveno neprilagođene *nerdove* u jednu od glavnih aktivnosti kojima većina populacije popunjava svoje slobodno vrijeme. Vrlo brzo nakon pojave prvih igara (*Pong*, *Space Invaders*, itd.) došlo je i do potrebe za korištenjem umjetne inteligencije ne bi li se unaprijedilo iskustvo igranja. Jedan od najranijih primjera primjećujemo u popularnoj igri *Pac-Man* tj. u načinu na koji se kreću neprijateljski duhovi ovisno o tome mogu li ili ne mogu nauditi glavnom liku.[1] Godine 2005. svjedočili smo zadnjoj pobjedi čovjeka nad najnaprednijim šahovskim algoritmom, a nedavna otkrića u polju neuronskih mreža otvorila su nove i uzbudljive pristupe implementiranju umjetne inteligencije u videoigre. Danas je računalo sposobno za štošta (npr. generiranje čitavih svjetova u poznatoj igri *No Man's Sky).*

U ovom je radu istreniran model koji koristeći podržano učenje ima zadatak optimalno preći jednu netrivijalnu razinu igre platformi. Podržano učenje je pristup strojnom učenju koji se temelji na maksimizaciji ukupne nagrade donošenjem ispravnih odluka u dinamičnom okruženju. Zadatak modela obavljen je uspješno ako se izbjegnu sve prepreke te se dođe do cilja u što kraćem vremenu. Poligon se postavlja nasumično kako model ne bi naučio samo jednu specifičnu situaciju već kvalitetno naučio koristiti dobiveno znanje. Sama je igra napravljena u programu *Unity*, a podržano učenje izvedeno je kroz paket *ML-Agents.*

U nastavku rada proći će se kroz korištene tehnologije, detaljno objasnit proces podržanog učenja i treniranja optimalnog modela za spomenuti problem, analizirati dobivene rezultate te objasniti izradu same igre u programu *Unity*.

2. Korištene tehnologije

U ovom poglavlju ukratko su opisane glavne tehnologije i alati korišteni pri razvoju same igre i treniranju modela. Sama igra izrađena je u programu Unity, kod je pisan u razvojnom okruženju *Visual Studio Code*, a za treniranje modela korišten je paket

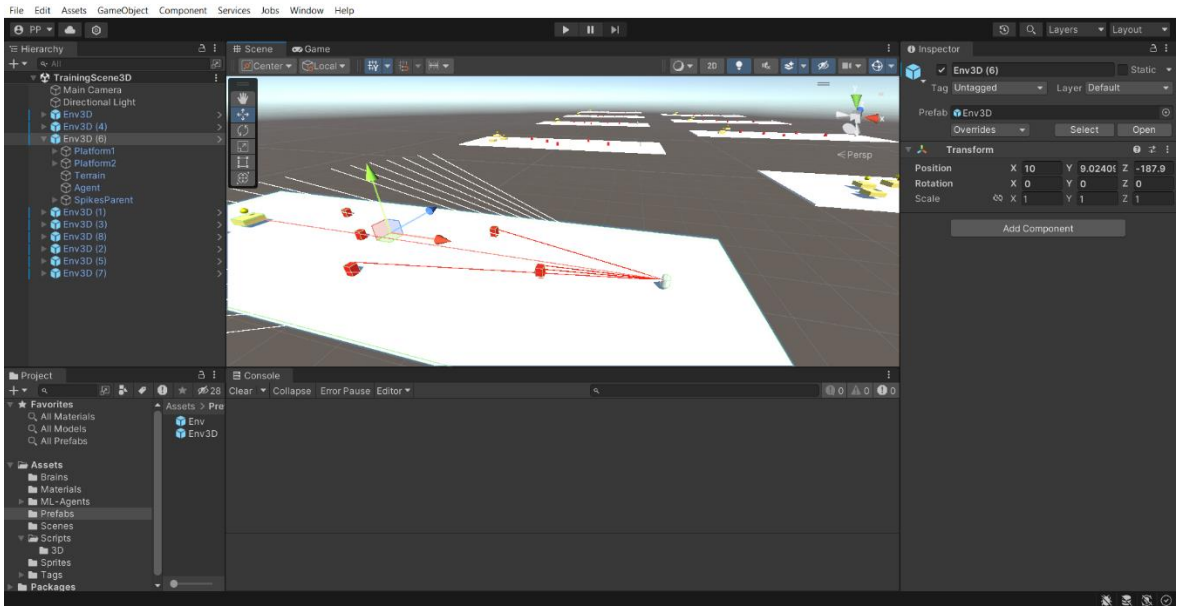
MLAgents temeljen na programskom jeziku *Python*. Uz glavne alate, korišten je i alat *TensorBoard* za vizualizaciju rezultata treniranja te repozitorij *Unity Asset Store* za preuzimanje modela, nebeskih okvira i materijala koji se koriste u igri.

2.1. Unity

Unity je program za izradu višeplatformskih 2D i 3D videoigra razvijen 2005. godine od strane *Unity Tecnologies.* Od samih početaka ideja programa *Unity* bila je demokratizacija razvoja igara što je ostvareno kroz intuitivno sučelje i činjenicu da je alat besplatan dok igra ne ostvari određeni profit. Ovakav pristup omogućio je početnicima da svoju ideju za igru provedu do kraja te je lansiraju na jednu od 19 različitih platformi kao što su razni desktop i mobilni operacijski sustavi, igraće konzole te web platforme.[2] Iz tih razloga oko programa *Unity* nastala je mnogobrojna zajednica što uvelike olakšava traženje savjeta i rješavanje izazova tijekom razvoja igre. Dostupna je i baza korisnih resursa *Unity Asset Store* gdje korisnici mogu preuzeti i kupiti već gotove modele, skripte, specijalne efekte i još mnogo toga.

Unity koristi programski jezik C# za pisanje skripti te funkcionira na principu različitih prozora vezanih uz pojedine aspekte igre koju razvijamo. Najbitniji od njih su prikazani na slici i objašnjeni u nastavku.

- ***Scene*** – scena u koju postavljamo objekte. Ona može predstavljati jednu razinu igre ili glavni izbornik na početku igre.
- ***Hierarchy*** – sadrži popis svih objekata koji se nalaze u sceni te uspostavlja odnose roditelj-dijete među njima.
- ***Game*** – pokazuje kako će izgledati konačna verzija igre.
- ***Inspector –*** sadrži i dopušta manipuliranje svih komponenti koje su pridružene odabranom objektu.
- ***Project*** – repozitorij svih datoteka koje sačinjavaju trenutni projekt.
- ***Console*** – prozor za ispis svih greški, upozorenja i *debug* informacija

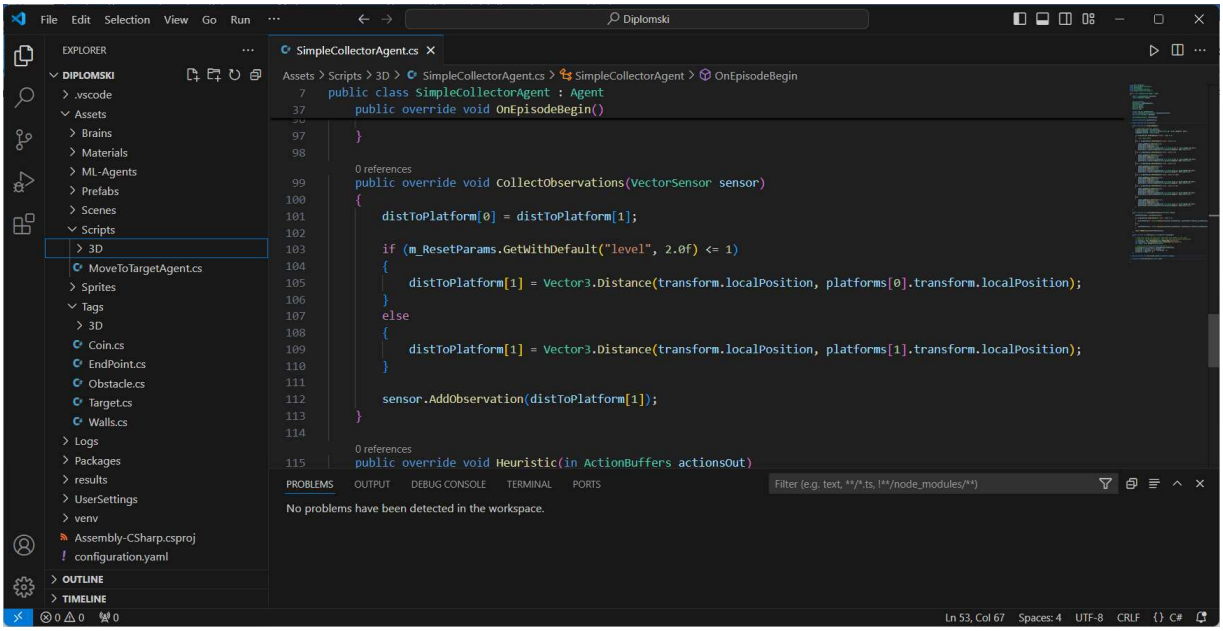


Slika 1 – Unity sučelje

Iz gore navedenih razloga Unity je odabran kao najprikladniji program za razvitak same logike i izgleda videoigre.

2.2. Visual Studio Code

Visual Studio Code (poznatiji kao VS Code) je uređivač izvornog koda razvijen od strane Microsofta i objavljen 2015. godine.[3] Za razliku od programa Visual Studio koji predstavlja kompletnu razvojnu okolinu, VS Code koristeći ekstenzije (C#, Unity) omogućuje intuitivno i efikasno pisanje koda koji se treba izvršiti u programu Unity. Alat nudi uslugu automatske nadopune teksta, samostalnog otklanjanja grešaka i mnoge druge pogodnosti te je stoga odabran za izradu ovog projekta. Njegovo sučelje prikazano je u nastavku.



Slika 2 – Visual Studio Code sučelje

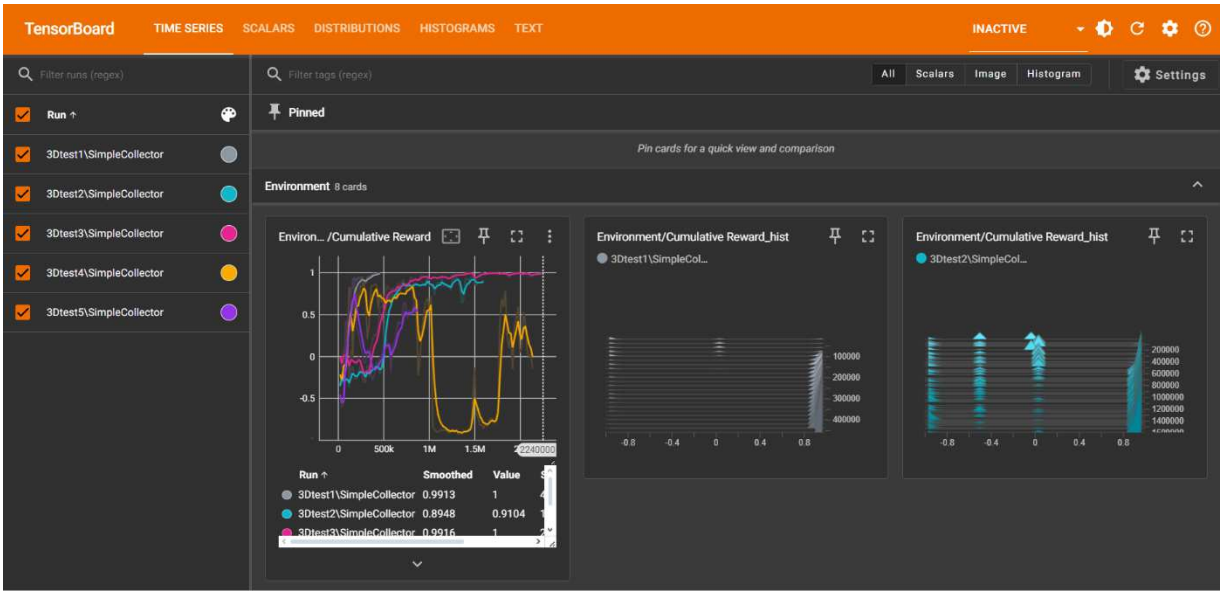
2.3. ML-Agents

Unity Machine Learning Agents toolkit (poznatiji kao ML-Agents) je projekt otvorenog koda koji omogućuje da igre i simulacije postanu okružja za treniranje inteligentnih modela. Temeljen je na biblioteci PyTorch te omogućuje implementaciju suvremenih algoritama podržanog učenja, učenja imitacijom i drugih metoda u Unity projekte.[4]

Preporučena praksa kod korištenja alata ML-Agents je stvoriti virtualnu okolinu, u njoj instalirati sve potrebne biblioteke te iz nje pozivati naredbe za treniranje modela tj. agenata. Detaljniji opis funkcionalnosti alata ML-Agents dan je u potpoglavlju 3.3.

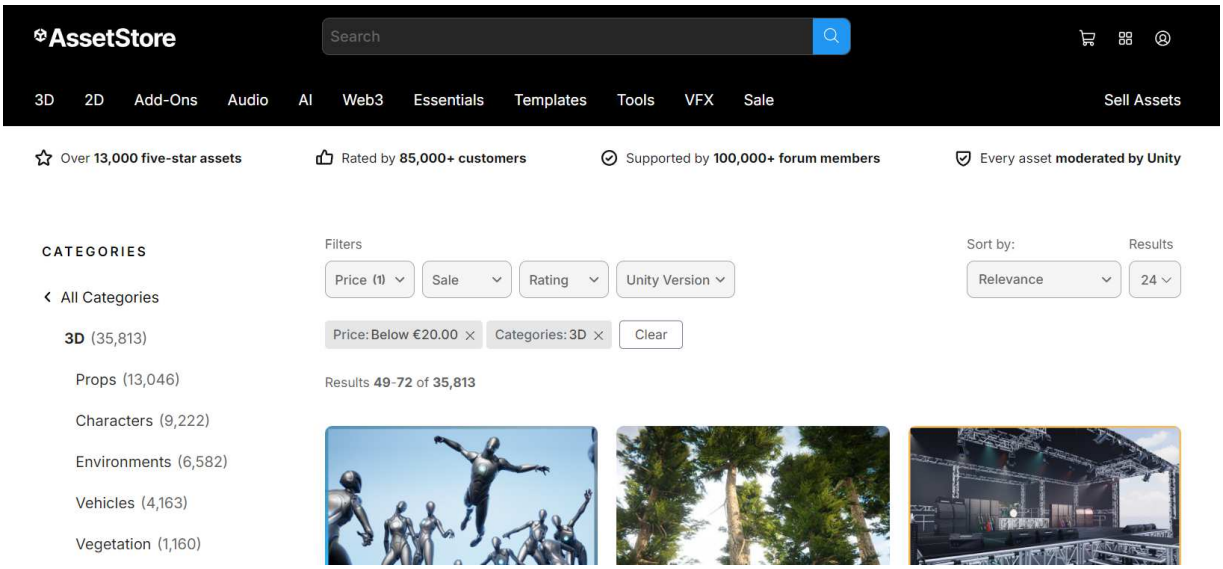
2.4. Ostali alati

TensorBoard je pomoćni alat koji se koristi uz PyTorch za vizualizaciju rezultata strojnog učenja. Omogućuje praćenje raznih metrika poput točnosti i gubitka, vizualizaciju grafa modela te pregled histograma.[5] Svi grafovi u ovom radu stvoreni su uz pomoć alata TensorBoard, a u nastavku je prikazano njegovo sučelje u web pregledniku.



Slika 3 – TensorBoard sučelje

Modeli, nebeski okviri i slični resurrsi preuzeti su iz *Unity Asset Store* repozitorija koji nudi besplatne i plaćene resurse za izradu igara.



Slika 4 – Unity Asset Store

3. Podržano učenje i alat ML-Agents

Fokus ovog rada je na specifičnoj verziji strojnog učenja znanoj kao podržano učenje. Podržano učenje koristi sistem nagrada i kazni kako bi doveo model do željenog ponašanja. U nastavku su dani kratki opći pregled strojnog učenja i njegova podjela, objašnjenje principa podržanog učenja te pristupi podržanom učenju u sklopu alata ML-agents.

3.1. Strojno učenje

Strojno učenje je područje proučavanja u umjetnoj inteligenciji koje se bavi proučavanjem algoritama koji uče iz podataka te generalizacijom rješavaju probleme na do tad neviđenim podacima. Danas je to jedno od najuzbudljivijih i najbrže razvijajućih područja istraživanja te smo svjedoci velikim postignućima na poljima obrade prirodnog jezika te generiranja i prepoznavanja slika. Tri su osnovna tipa strojnog učenja: nadzirano, nenadzirano i podržano učenje.[6]

Algoritmi nadziranog učenja koriste klasificirane ulazne skupove podataka kako bi naučili neviđene podatke svrstavati u jednu od klasa (klasifikacija) ili im pridati neki odziv (regresija). Proces nadziranog strojnog učenja počinje sakupljanjem podatka koje dijelimo na podatke za treniranje i podatke za testiranje. Cilj je odabrati optimalni algoritam koji će pronaći uzorke među podacima za treniranje te ih točno iskoristiti pri klasifikaciji/regresiji podataka za testiranje. Neki od najpoznatijih algoritama nadziranog učenja su: linearna regresija, logistička regresija, K-najbliži susjedi, stabla odlučivanja i podrška vektorskih strojeva.[7]

Algoritmi nenadziranog učenja računalu daje neoznačene podatke, a algoritam pokušava pronaći inherentne sličnosti između nekih instanci. Podatkovne točke odvojene su na temelju toga na kojoj se strani hiper-ravnine nalaze. Neki od najpoznatijih algoritama nenadziranog učenja su analiza glavnih komponenata i klasteriranje K-sredstava.[7]

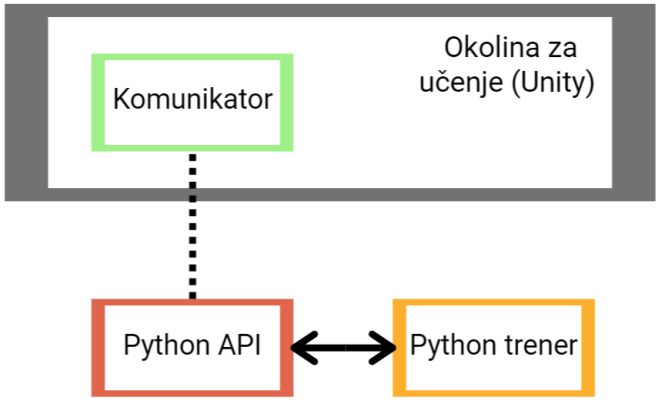
3.2. Podržano učenje

Podržano učenje predstavlja interdisciplinarno područje unutar strojnog učenje koje proučava kako bi se inteligentni agent trebao ponašati u dinamičnom okruženju ne bi li maksimizirao ukupnu nagradu. Osnovni princip je taj da agenta nagrađujemo za akcije koje ga dovode bliže cilju, a kažnjavamo za akcije koje ga od cilja udaljuju. Tijekom odlučivanja o nagradama i prilagodbe algoritma valja voditi brigu o kompromisu između istraživanja i iskorištavanja. Agenti koji previše teže istraživanju neće dovoljno konzistentno koristiti svoje znanje kako bi maksimizirali nagradu dok agenti skloni iskorištavanju riskiraju zapinjanje u lokalnom maksimumu bez da se istraže svi putevi koji vode do nagrade.[8]

Postoji mnogo algoritama podržanog učenja, no fokusirat ćemo se na algoritme PPO i SAC koje koristi alat ML-Agents.

3.3. Podržano učenje unutar alata ML-Agents

Podržano učenje odvija se u okolinu za učenje tj. u Unity sceni. Tamo definiramo ponašanje agenta i njegovu okolinu za treniranje. Samo učenje odvija se u Python virtualnom okruženju koje komunicira s alatom Unity. Parametre algoritma stoga definiramo u komandnoj liniji kada pozivamo ML-Agents. Ovaj odnos pojednostavljeno je ilustriran na slici 4.

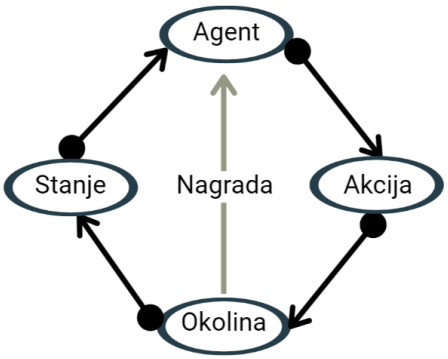


Slika 5 - komunikacija između Unitya i Pythona

Treniranje agenta sastoji se od epizoda. Na kraju epizode poželjno je agentu dati neku nagradu ili kaznu, a nakon toga postavljamo okolinu i agenta natrag u početni položaj.[4]

ML-Agents omogućuje nam da treniramo agente raznim metodama. U svakom koraku igre bitno je imati definirane sljedeće tri stavke:

- **Opažanja** – agent kroz opažanja tumači svoju okolinu. Opažanja mogu biti numerička i/ili vizualna te se prikazuju iz perspektive agenta. Primjer numeričkog opažanja može biti udaljenost agenta od cilja ili broj preostalih novčića na trenutnoj razini. Moguće je kao opažanje uzeti neke od poznatih tipova podataka koji se koriste u programu Unity kao što je Vector3. Vizualna opažanja mogu biti slike kamere pridružene agentu. Bitno je naglasiti da opažanja moraju biti vrijednosti kojih je agent „svjestan“ (npr. opažanje vezano uz neprijatelja koji se skriva ne bi se trebalo koristiti).
- **Akcije** – agent kroz akcije djeluje unutar okoline. Akcije mogu biti diskretne ili kontinuirane. Primjer diskretne akcije je pucanje agenta iz puške. U svakom trenu agent može ispaliti metak (vrijednost 1) ili ostati miroljubiv (vrijednost 0). Vrijednosti kontinuiranih akcija nalaze se između -1 i 1 te su korisne za kompleksnije akcije kao što je odabir brzine.
- **Nagrade** – agenta nagrađujemo i kažnjavamo ovisno o njegovom ponašanju. Nagrade se ne dijele u svakom koraku igre već samo kada agent napravi nešto vrijedno nagrade (ili kazne). Najjednostavniji primjer je agent koji dobiva nagradu vrijednosti 1 ako dođe do cilja te kaznu vrijednosti -1 ako padne s platforme.[4]



Slika 6 – vizualizacija agentove interakcije s okolinom

Postoje nekoliko scenarija treniranja u okolini za učenje.

- Najjednostavniji je treniranje jednog agenta koji sam uči željeno ponašanje.
- Proces se može ubrzati paralelnim treniranjem više agenata s neovisnim sustavima nagrada (npr. simultano treniranje 10 robotskih ruka da otvore vrata).
- Moguće je i da dva agenta treniraju jedan protiv drugoga u kojem slučaju su njihovi signali nagrade inverzni. Ova je metoda korištena pri razvijanju agenta koji pobjeđuje ljude u videoigri *Dota 2.*[1]
- Različiti agenti mogu dijeliti nagradu što ih potiče na međusobnu suradnju. Ovo koristimo u slučajevima gdje agenti moraju riješiti problem, ali svaki od njih ima samo parcijalne informacije ili u simulaciji timskih sportova.
- Najkompliciraniji scenarij treniranja je sustav u kojem mnogo agenata (neki od njih istog ponašanja) dijele nagrade. Time možemo simulirati čitavi ekosustav te modelirati ponašanje različitih životinja u njemu.[4]

Glavna dva algoritma koja koristi alat ML-Agents su PPO (Proximal Policy Optimization) i SAC (Soft Actor-Critic). Za razliku od algoritma PPO, SAC ima mogućnost učenja iz prijašnjih iskustava. Iskustva koja se skupljaju spremaju se u poseban međuspremnik iz kojeg se nasumično povlače tijekom treninga. SAC je prikladniji za teže i sporije okoline (cca. 0.1s između koraka), dok PPO daje bolje perfomanse u bržim okolinama kao što su fizikalne simulacije koje se u alatu Unity ažuriraju 30 puta u sekundi.[4]

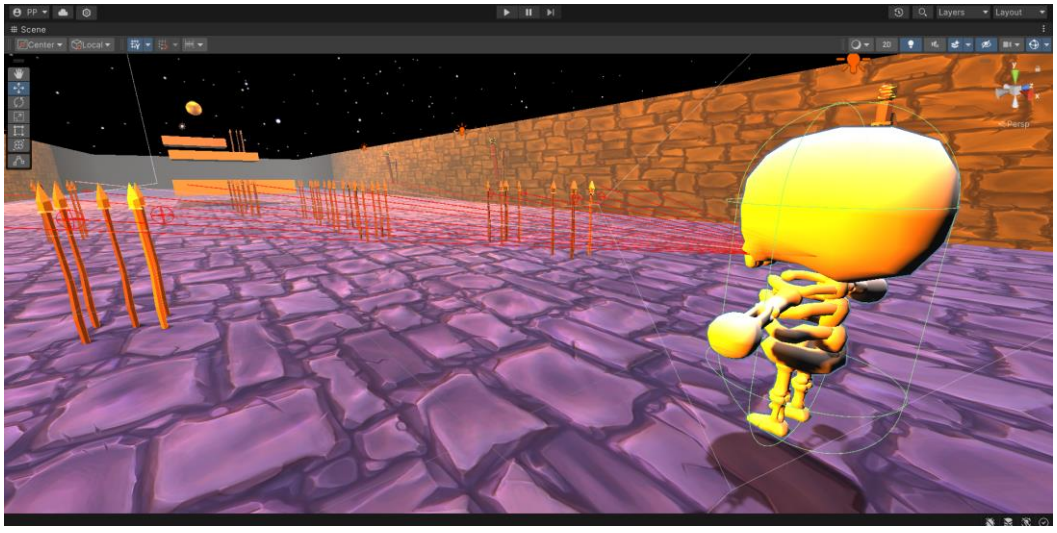
Pri treniranju kompleksnijih zadataka poželjno je agentu postepeno davati sve teže zahtjeve. U tom slučaju poželjno je koristiti takozvano učenje kurikulumom koje učenje dijeli na određeni broj lekcija. Agent prelazi na sljedeću lekciju tek kada na prethodnoj ostvari dovoljno dobar rezultat i tako prenosi znanje na kompliciraniji zadatak. Upravo je ovaj pristup korišten pri treniranju modela u ovom projektu (o čemu će biti riječ u sljedećim poglavljima). [4]

4. Opis zadatka i razine u igri

Prije nego što objasnimo proces treniranja inteligentnog modela potrebno je objasniti kontekst u kojem od djeluje. Zadatak je naizgled jednostavan – potrebno je pokupiti novčić koji se nalazi na kraju razine. Agent djeluje u 3D okružju nalik na kutiju bez poklopca. Uzevši u obzir da ideja ovog rada nije razvijanje kompletne igre razina već treniranje što inteligentnijeg agenta fokus će biti na jednoj razini tj. *Unity* sceni. Cilj je da trenirani agent konzistentno prolazi najkompleksniju inačicu razine.

4.1. Objekti unutar jedne razine

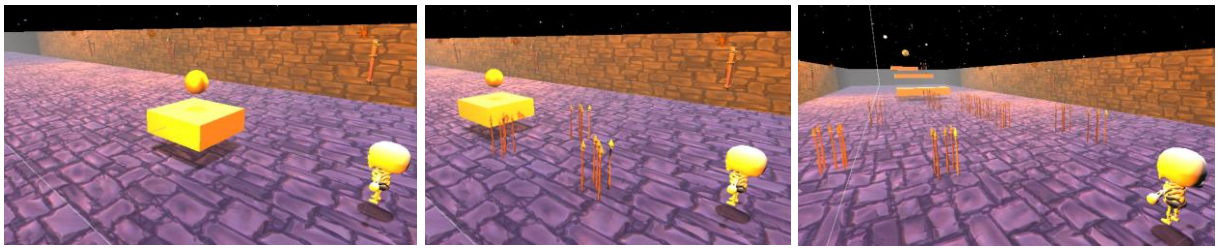
Unutar razine nalazi se nekoliko vrsta objekata. Glavni objekt je agent koji se nalazi na podnoj platformi. Agent je okružen s 4 zida koji ga sprječavaju da padne s poligona. Na razini se još nalaze i manje platforme, a na jednoj od njih nalazi se novčić koji agent mora skupiti. Po podnoj platformi nasumično su postavljeni šiljci koje agent mora izbjegavati. Svaki objekt ima svoju oznaku pomoću koje agent prepoznaje kojoj vrsti pripada.



Slika 7 – razina igre u programu Unity

4.2. Zadatak

Zadatak agenta je skupiti novčić koji se nalazi na jednoj od platforma. Najjednostavniju verziju zadatka predstavlja slučaj u kojem je platforma s novčićem blizu agenta te na podu nema šiljaka. Zadatak srednje težine dobit ćemo ako platformu udaljimo od agenta te na pod stavimo šiljke. Najteži zadatak dobili bismo kada bi platforme postavili tako da agent prvo mora skočiti na jednu ne bili došao na drugu koja sadrži novčić pritom zadržavajući šiljke na podnoj platformi.



Slike 8, 9 i 10. – razine igre različite kompleksnosti

Postoje tri načina da agent ne izvrši zadatak: zabijanjem u zidove razine, zabijanjem u šiljke te istekom dopuštenog vremena.

5. Agentov *Unity* objekt

Agent je temeljni i najkompleksniji objekt u sceni. Predstavljen je animiranim modelom te sadrži mnoge komponente koji mu omogućavaju treniranje podržanim učenjem i testiranje znanja. Glavna skripta kojom su definirana opažanja, akcije i nagrade zove se *AgentBehavior.*

5.1. Kretanje modela po razini

Agent je sposoban za kretanje u smjeru naprijed-nazad ovisno o trenutnoj rotaciji te ska kanje. Detekcija dodira i simulacija fizike temelji se na *Collider* komponenti pridruženoj svakom objektu te *RigidBody3D* komponenti na agentu.

Detaljnije funkcionalnosti definirane su u skripti *SimpleCharacterController* preuzetoj s interneta.

5.2. Akcije

Pri prvom pokušaju definiranja akcija agenta koristile su se kontinuirane vrijednosti za rotaciju i kretanje te diskretna vrijednost (0 ili 1) za skakanje. Kontinuirane vrijednosti odabrane su u nadi da će agent u zadnjim fazama učenja krenuti koristiti veću brzinu i snažniju rotaciju ne bi li brže riješio zadatak. Nakon treniranja ipak se ispostavilo da jednostavnost diskretnih akcija bolje odgovara modelu te su odabrane sljedeće tri diskretne akcije:

1. **kretanje**

   0 – kretanje unazad

   1 – kretanje unaprijed

   2 – stajanje na mjestu

2. **rotacija**

   0 – rotacija udesno

   1 – rotacija ulijevo

   2 – bez rotacije

3. **skakanje**

   0 – bez skoka

   1 – skok

5.3. Opažanja

Kroz razvijanje agenta isprobano je mnogo pristupa sakupljanju opažanja. U početku je dana prednost numeričkim opažanjima raznih tipova podataka kao što su:

- udaljenost agenta od novčića, platformi ili zidova (float)
- pozicija agenta (Vector3)
- rotacija agenta (Quaternion)
- vrijednost koja označuje je li agent u zraku (boolean)

- broj proteklih koraka prije kraja epizode treniranja (int)

Ovakva opažanja rezultirala su suboptimalnim ponašanjem agenta te se odlučilo koristiti i vizualna opažanja.

Jedan od načina na koji agent može vizualno pratiti stanje svoje okoline je izbacivanje zraka. Ideja je u svakom koraku odaslati nekoliko zraka u smjeru pogleda agenta te koristiti povratne informacije (objekt koji je pogođen i udaljenost) kao opažanja. ML-Agents alat ima ugrađenu komponentu *RayPerceptionSensor3D* koja radi upravo to tako da ne moramo ručno pisati kod u glavnoj skripti.

Uz izbacivanje zraka korišteno je i jedno numeričko opažanje - udaljenost između agenta i platforme s novčićem.

5.4. Nagrade

Tijekom treniranja agenta preporuča se nagrade držati što jednostavnijima te u rasponu od –1 do 1.[10] Imajući to na umu odabrane su sljedeće nagrade i kazne:

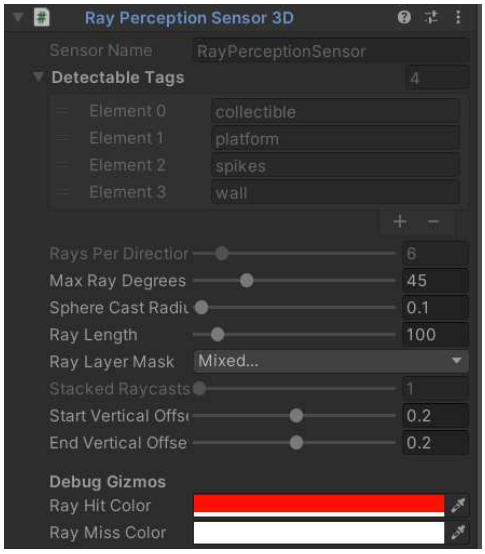
- **Dodir s novčićem (+1)** - najviše nagrađujemo obavljanje samog zadatka.
- **Dodir sa zidovima (-1)** - kažnjavamo kretanje agenta u krivom smjeru.
- **Dodir sa šiljcima (-0.5)** - kazna je manja jer ne želimo agenta previše odvući od šiljaka koji su nekada direktno na putu prema novčiću.
- **Svaki korak treninga agent dobiva malu kaznu (-0.000125)** – cilj ove kazne je spriječiti agenta da izbjegava akcije te u strahu od negativnih nagrada čeka kraj epizode skakutajući na mjestu.

5.5. Komponente objekta agenta

U ovom potpoglavlju objašnjenja je uloga svake komponente koja je pridružena objektu agenta. Pojašnjenje su opcije komponenti relevantne za ovaj rad.

- *Transform* – sadrži informacije o poziciji, rotaciji i skali objekta
- *Mesh filter* i *Mesh renderer* – definiraju izgled modela
- *Capsule collider* – služi za detekciju dodira s drugim objektima
- Rigidbody – simulira fizikalne pojave poput gravitacije i trenja

- *SimpleCharacterController* skripta - definira ponašanje agenta na terenu s nagibom, provjerava je li uzemljen te svaki korak provjerava unose za kontrolu agenta. Skripti tijekom treniranja i testiranja šaljemo tri unosa: rotaciju (lijevo-desno), pomak (naprijed-nazad) te skok.
- *RayPerceptionSensor3D* – određuje način odašiljanja zraka koje koristimo kao vizualna opažanja agenta

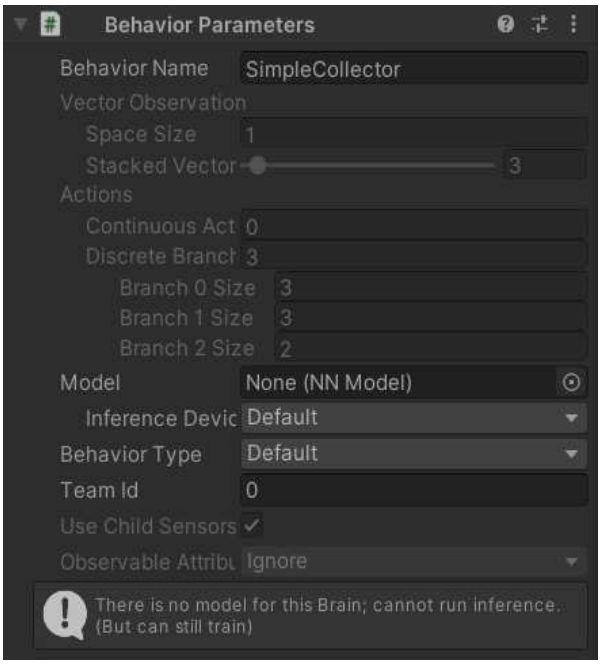


Slika 11 – RayPerceptionSensor3D komponenta

- *Detectable tags* – oznake objekata koje će zrake prepoznavati
- *Rays Per Direction* – broj zraka
- *Max Ray Degrees* – širina snopa
- *Ray Length* – maksimalni domet zraka

- *Behavior Paramaters* – određuje neke od pojedinosti pri treniranju agenta

  - *Behavior Name* – ime modela koji treniramo



Slika 12 – Behavior Parameters komponenta

- *Vector Observation* – određuje broj opažanja, a opcija *Stacked Vectors* omogućava spremanje nekoliko uzastopnih vrijednosti što modelu omogućuje korištenje pamćenja između koraka
- *Actions* – definira broj kontinuiranih i diskretnih akcija te broj mogućih vrijednosti za svaku od njih
- *Model* – jednom naučeni model možemo postaviti ovdje kako bismo testirali što je agent naučio
- *Behavior Type* – može biti postavljen na *Inference* što znači da agent uči ili koristi naučeni model te na *Heuristic* što znači da možemo sami kontrolirati model
- *Use Child Sensors* – ukoliko je opcija odabrana, model će bilježiti i opažanja objekata koji su pod njim u *Hierarchy* prozoru

- *Decision Requester* – određuje koliko često tjeramo agenta da napravi odluku. Korak je postavljen na vrijednost 5.

5.6. *AgentBehavior* skripta

Ovom centralnom skriptom detaljno su definirani svi parametri za treniranje agenta te njegova interakcija s okolinom. U nastavku objašnjenja je uloga svake funkcije u skripti relevantne za treniranje agenta.

- Funkcija *OnEpisodeBegin()* – ova funkcija poziva se pri početku epizode treniranja. Svi se objekti postavljaju u svoje početne pozicije sljedećim naredbama.

```
transform.position = startPosition;
transform.rotation = Quaternion.Euler(Vector3.up * 0f);
rigidbody.velocity = Vector3.zero;
foreach (GameObject obj in platforms) obj.transform.rotation = Quaternion.identity;
foreach (Transform t in spikes.transform){
    t.localPosition = new Vector3(t.localPosition.x,t.localPosition.y,Random.Range(-8f,8f));
}
```

Isječak koda 1 – OnEpisodeBegin funkcija

Zatim se dohvaća varijabla trenutne lekcije u kurikulumu. Ovisno o njenoj vrijednosti postavlja se okolina odgovarajuće razine kompleksnosti. U nastavku su navedene sve lekcije i njihove motivacije:

- **Lekcija 1** – ispred agenta je jedna platforma na kojoj se nalazi novčić. Ovime agent uči da je cilj pokupiti novčić te da to ne može postići samo kretanjem već treba skočiti na platformu.
- **Lekcija 2** – ispred agenta nalaze se dvije platforme po kojima se treba popeti ne bi li se pokupio novčić. Agent ovu lekciju nije mogao naučiti bez prethodnog znanja jer je odašiljanjem zraka rijetko kada pogađao novčić. Ovako postavljenu lekciju agent je u stanju naučiti jer iz prethodne lekcije zna da treba skočiti na platformu.
- **Lekcija 3** – ispred agenta nalazi se kompleksni poligon od tri platforme na vrhu kojega se nalazi novčić. Koristeći znanje iz prethodne lekcije agent je nakon podužeg treniranja naučio popeti se po ovoj platformi.
- **Lekcija 4** – postavljamo šiljke na putu do platforma. Agent koji već zna popeti se do novčića sada treba ukomponirati izbjegavanje šiljaka u svoje ponašanje.

- **Lekcije 5, 6 i 7** – postepeno udaljavamo platforme i novčića od agenta. Agentu postaje teže orijentirati se te mora izbjegavati sve više šiljaka ne bi li uspješno riješio zadatak.

U nastavku je kao primjer dan kod četvrte lekcije. Objekt koji sadrži sve šiljke postavlja se u aktivno stanje, a od platforma palimo samo najkompliciraniju treću platformu. Njoj ujedno mijenjamo poziciju i rotaciju kako agent ne bi učio samo jedan specifičan slučaj.

```
case 0.75f:

    spikes.gameObject.SetActive(true);
    platforms[0].SetActive(false); platforms[1].SetActive(false); platforms[2].SetActive(true);
    platforms[2].transform.localPosition = new Vector3(30, 0, Random.Range(-10f, 10f));
    platforms[2].transform.Rotate(Vector3.up, Random.Range(-15f, 15f), Space.Self);
    break;
```

Isječak koda 2 – postavljanje lekcije u funkciji OnEpisodeBegin

- Funkcija *CollectObservations (VectorSensor sensor)* – ova funkcija poziva se svaki korak treniranja te sakuplja agentova opažanja. Kao argument dobiva podatak tipa *VectorSensor* u koji spremamo jedno eksplicitno opažanje udaljenosti između agenta i platforme koja je trenutno u razini igre. Naredba *AddReward* svaki korak daje negativnu nagradu u iznosu od *1 / maksimalni broj koraka*. Time kažnjavamo agenta za „okolišanje“ i tjeramo ga da bude proaktivan u rješavanju problema.

```
if (envParams.GetWithDefault("level", 3.0f) == 0f)
{
    distToPlatform = Vector3.Distance(transform.localPosition, platforms[0].transform.localPosition);
}
else if (envParams.GetWithDefault("level", 3.0f) == 0.25f)
{
    distToPlatform = Vector3.Distance(transform.localPosition, platforms[1].transform.localPosition);
}
else
{
    distToPlatform = Vector3.Distance(transform.localPosition, platforms[2].transform.localPosition);
}
sensor.AddObservation(distToPlatform);
AddReward(-1f / MaxStep);
```

Isječak koda 3 – CollectObservations funkcija

- Funkcija *OnActionReceived(ActionBuffers action)* – ova funkcija kao argument prima međuspremnik s vrijednošću svih akcija te ih prosljeđuje skripti *SimpleCharacterController* koja kontrolira kretanje agenta. Postoje tri vrijednosti koje se prosljeđuju: pomak naprijed-nazad (varijabla *vertical*), rotacija lijevo-desno (varijabla *horizontal*) te skok (varijabla *jump*).

```
float vertical = actions.DiscreteActions[0] <= 1 ? actions.DiscreteActions[0] : -1;
float horizontal = actions.DiscreteActions[1] <= 1 ? actions.DiscreteActions[1] : -1;
bool jump = actions.DiscreteActions[2] > 0;

characterController.ForwardInput = vertical;
characterController.TurnInput = horizontal;
characterController.JumpInput = jump;
```

Isječak koda 4 – OnActionReceived funkcija

- Funkcija *Heuristic*(*in ActionBuffers actionsOut*) – funkcija se poziva kada odaberemo ručnu kontrolu agenta. Ova funkcija uzima vrijednosti strelica na tipkovnici te *space* tipke te ih sprema u varijablu tipa *ActionBuffers* temeljem koje agent izvodi svoje akcije. Funkcija *GetAxisRaw(string axisName)* uzima kao argument jednu od dvije osi te vraća njenu diskretnu vrijednost. Na primjeru horizontalne osi funkcija će vratiti -1 ako je pritisnuta lijeva strelica, 1 ako je pritisnuta desna te 0 ako nijedna strelica nije pritisnuta. Te vrijednosti prije prosljeđivanja agentu moramo prebaciti iz skupa vrijednosti [-1, 0, 1] u skup vrijednosti [0, 1, 2] koje odgovaraju diskretnim akcijama.

```
int vertical = Mathf.RoundToInt(Input.GetAxisRaw("Vertical"));
int horizontal = Mathf.RoundToInt(Input.GetAxisRaw("Horizontal"));
bool jump = Input.GetKey(KeyCode.Space);

ActionSegment<int> actions = actionsOut.DiscreteActions;
actions[0] = vertical >= 0 ? vertical : 2;
actions[1] = horizontal >= 0 ? horizontal : 2;
actions[2] = jump ? 1 : 0;
```

Isječak koda 5 – Heuristic funkcija

- Funkcija *OnTriggerEnter(Collider other)* – funkcija se poziva pri dodiru s bilo kojim objektom koji sadrži *Collider* komponentu označenu kao okidač.

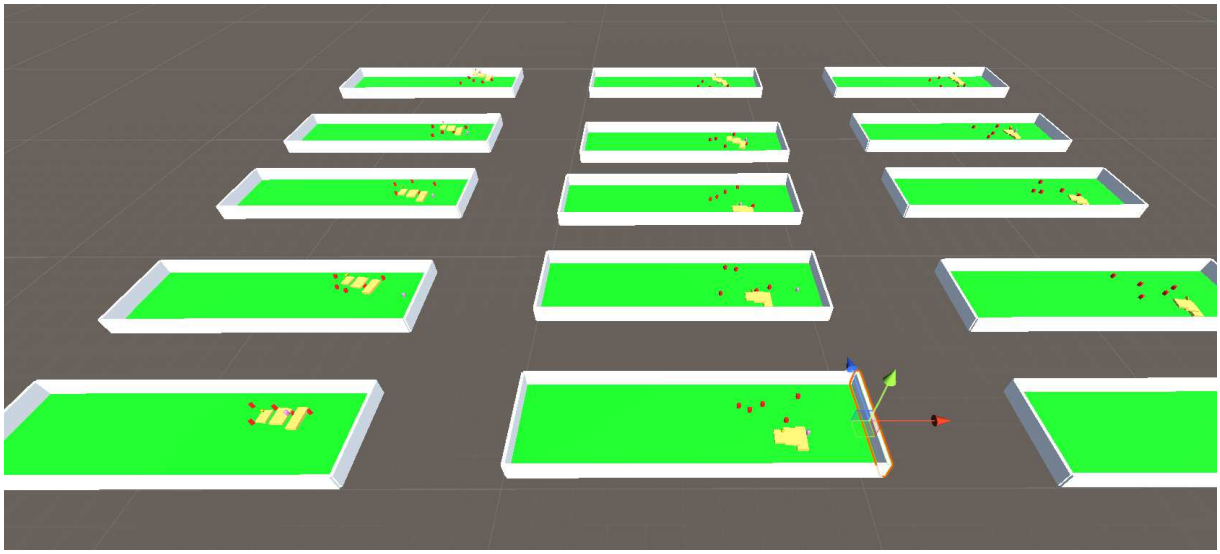
Uvjetnom naredbom *if* provjeravamo oznaku objekta te u skladu s time nagrađujemo ili kažnjavamo agenta funkcijom *AddReward(float amount)* te završavamo epizodu treniranja funkcijom *EndEpisode().* Naredbom *baseMeshRenderer.material = color;* mijenjamo boju poligona za lakšu preglednost tijekom treniranja.

```
if (other.TryGetComponent(out Collectible collectible))
{
     AddReward(1f);
     baseMeshRenderer.material = green;
     EndEpisode();
}
else if (other.TryGetComponent(out Spikes spikes))
{
     AddReward(-0.5f);
     baseMeshRenderer.material = red;
     EndEpisode();
}
else if (other.TryGetComponent(out Walls wall))
{
     AddReward(-1f);
     baseMeshRenderer.material = red;
     EndEpisode();
```

Isječak koda 6 – OnTriggerEnter funkcija

6. Treniranje agenta

Za treniranje agenta odabran je pristup paralelnog učenja s petnaest kopija agenta koji koriste ista opažanja, akcije i nagrade u odvojenim okolinama čime je uvelike skraćeno vrijeme treniranja. Problem se pokazao kao dovoljno kompleksan te je zahtijevao korištenje kurikuluma pri učenju.



Slika 13 – paralelno treniranje

6.1. Pokretanje treniranja

Treniranje pozivamo iz komandne linije. U virtualnom okruženju instalirane su sve Python biblioteke potrebne za uspješno treniranje agenta. Prvo je potrebno pozicionirati se u mapu gdje se nalazi virtualno okruženje te ga pokrenuti pozivanjem skripte *activate*:

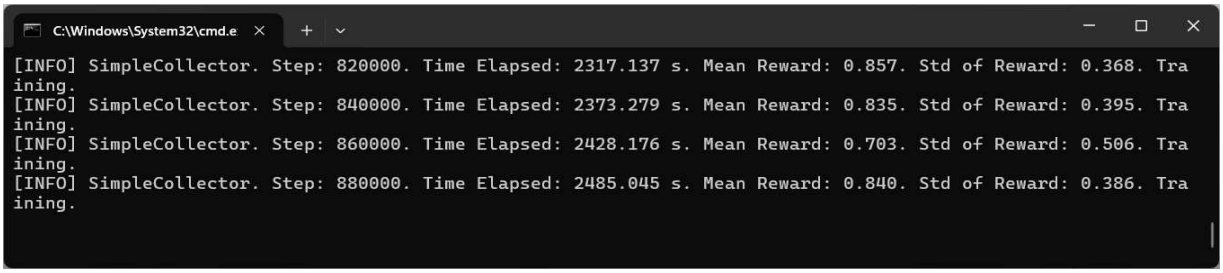
venv\scripts\activate

Sljedeći je korak pozvati naredbu za treniranje agenta:

mlagents-learn configuration.yaml --run-id="ime_treninga" --force

U datoteci *configuration.yaml* postavljene su sve varijable za treniranje te ćemo je detaljno proći u nastavku. Polje *run-id* označuje ime treninga, a zastavica *force* služi tome da novi trening istog imena prebriše staru inačicu.[11]

Pod uvjetom da je sve dobro postavljeno, agent će krenuti trenirati te periodički ispisivati svoj napredak. Pri konfiguraciji treniranja možemo odrediti koliko često želimo dobiti informacije o prosječnoj nagradi i standardnoj devijaciji iste.[11]



Slika 14 – ispis tijekom treniranja

6.2. Konfiguracijska datoteka

U ovoj datoteci ekstenzije .yaml određujemo konfiguraciju za treniranje agenta. Datoteka je dana u nastavku, a pojašnjenji su najrelevantniji parametri.[9]

```yaml
behaviors:
  SimpleCollector
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 2048
      learning_rate: 0.0001
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedul : linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
```

Isječak koda 7 – konfiguracijska datoteka

- *trainer_type* – odabir algoritma za treniranje
- *learning_rate* – početna stopa učenja za gradijentni spust. Ako je učenje nestabilno potrebno ju je smanjiti.
- *beta* – veće vrijednosti čine učenje više nasumičnim što rezultira boljim istraživanjem prostora
- *epsilon* – određuje koliko brzo agent smije mijenjati svoje ponašanje
- *lambd* - predstavlja regulacijski parametar tj. koliko se agent oslanja na trenutnu procjenu vrijednosti pri izračunu sljedeće procjene
- *hidden_units* – broj skrivenih slojeva u neuronskoj mreži. U slučajevima kada su odnosi između vrijednosti obzervacija kompeksni poželjno je povećati ovaj parametar.

```
reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 20000000
    time_horizon: 128
    summary_freq: 20000
    threaded: true
```

Isječak koda 7 – konfiguracijska datoteka

- *reward_signals* – svaki tip nagrade množi se parametrom *strength,* a parametrom *gamma* određuje se gubi li nagrada vrijednost s vremenom.
- *max_steps* – maksimalni broj koraka učenja
- *summary_freq* – koliko često želimo ispis napretka učenja

U sekciji *environment_paramaters* definiran je kurikulum za treniranje agenta. Problem se pokazao prekompleksnim za rješavanje klasičnim pristupom te su uvedene razne lekcije koje postepeno uvode sve teže probleme. Postepenim uvođenjem kompliciranijih platformi, postavljanjem šiljaka tek nakon naučenog penjanja po platformama te postepenim udaljavanjem platformi postignuti su zadovoljavajući rezultati treniranja. Relevantni parametri jedne lekcije u kurikulumu dani su u nastavku.

```yaml
environment_parameters:
  level:
    curriculum:
      - name: Lesson0 #Platforma1
        completion_criteria:
          measure: reward
          behavior: SimpleCollector
          signal_smoothing: true
          min_lesson_length: 300
          threshold: 0.95
        value: 0
      - name: Lesson0.25 #Platforma2
        completion_criteria:
          measure: reward
          behavior: SimpleCollector
          signal_smoothing: true
          min_lesson_length: 300
          threshold: 0.95
        value: 0.25
      .
      .
      .
      - name: Lesson1 #PlatformaDalje1
        completion_criteria:
          measure: reward
          behavior: SimpleCollector
          signal_smoothing: true
          min_lesson_length: 300
          threshold: 0.95
        value: 1
      .
      .
      .
      - name: Lesson3 #ZadnjaLekcija
        value: 3
```

Isječak koda 8 – konfiguracijska datoteka (environment paramaters)

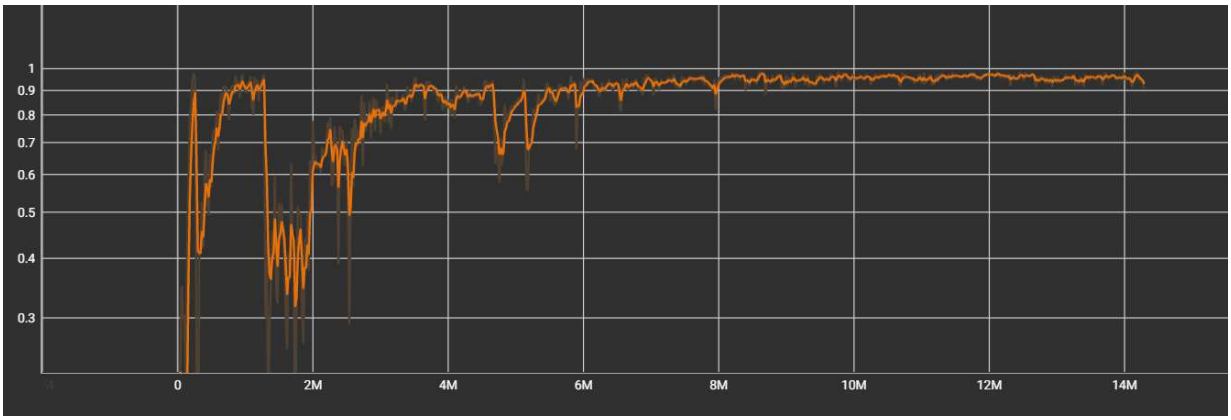
- *name* – ime lekcije
- *measure* – koji kriterij uzimamo za završetak lekcije (koristimo prosjek nagrade)
- *min_lesson_length* – koliko najrecentnijih vrijednosti uzimamo pri računanju prosjeka nagrade
- *threshold* – prag koji prosjek nagrade mora doseći ne bi li prešli na iduću lekciju
- *value* – vrijednost varijable koju dohvaćamo u glavnoj *Unity* skripti pri postavljanju lekcije

7. Rezultati treniranja i usporedba pristupa

U ovom poglavlju analizirat ćemo rezultate najuspješnijeg treniranja koristeći alat *TensorBoard* koji generira razne grafove koji nam daju detaljan uvid u proces učenja. Nakon što iz više kutova sagledamo najuspješnije treniranje usporedit ćemo ga s učenjem bez kurikuluma.

7.1. Analiza najuspješnijeg treniranja

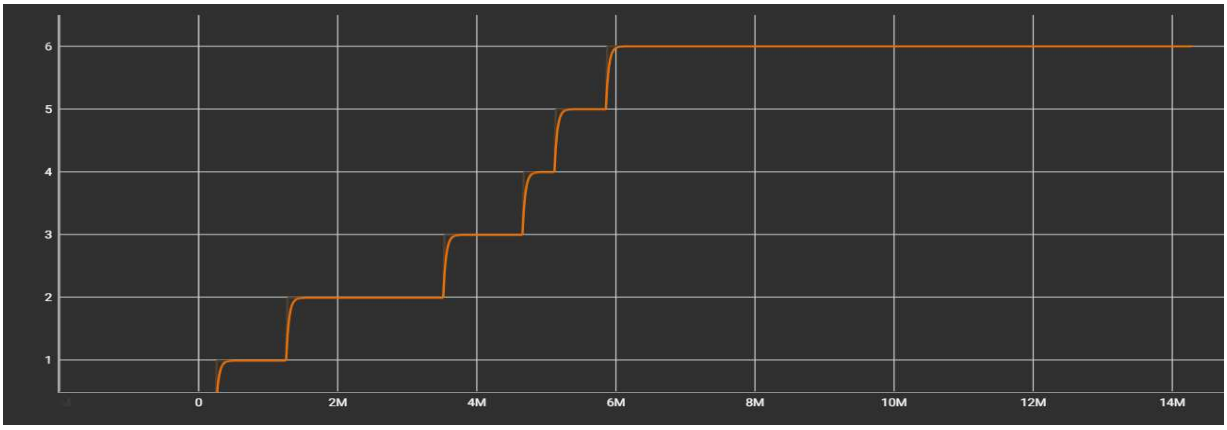
Pristup učenju s kurikulumom rezultirao je zadovoljavajućim rezultatima nakon cca. 6 milijuna koraka, no algoritam nije prekidan sve do cca. 14 milijuna koraka. Prvi graf koji ćemo sagledati je graf ukupne nagrade:



Graf 1 – kumulativna nagrada (učenje kurikulumom)

Apscisa predstavlja broj koraka treniranja, a ordinata ukupnu nagradu (prikazana u logaritamskoj skali radi bolje preglednosti). Na prvu primjećujemo više mjesta na kojima vrijednost grafa naglo pada. Ta mjesta predstavljaju točke u kojima je algoritam usvojio trenutnu lekciju te je krenuo na iduću. Prve tri lekcije predstavljaju uvođenje kompleksnijih platformi te graf u njima značajno pada. Kasnije lekcije sastoje se uglavnom od udaljavanja platforme od agenta. Takva promjena ne mijenja mnogo bit problema što smanjuje pad vrijednosti grafa u tim trenucima. Nakon zadnje lekcije vrijednost ukupne nagrade stagnirala je veoma blizu maksimalne vrijednosti. Valja napomenuti da nagrada vrijednosti 1 nije moguća jer agenta minimalno kažnjavamo za svaki korak koji radi kako bi spriječili okolišanje.

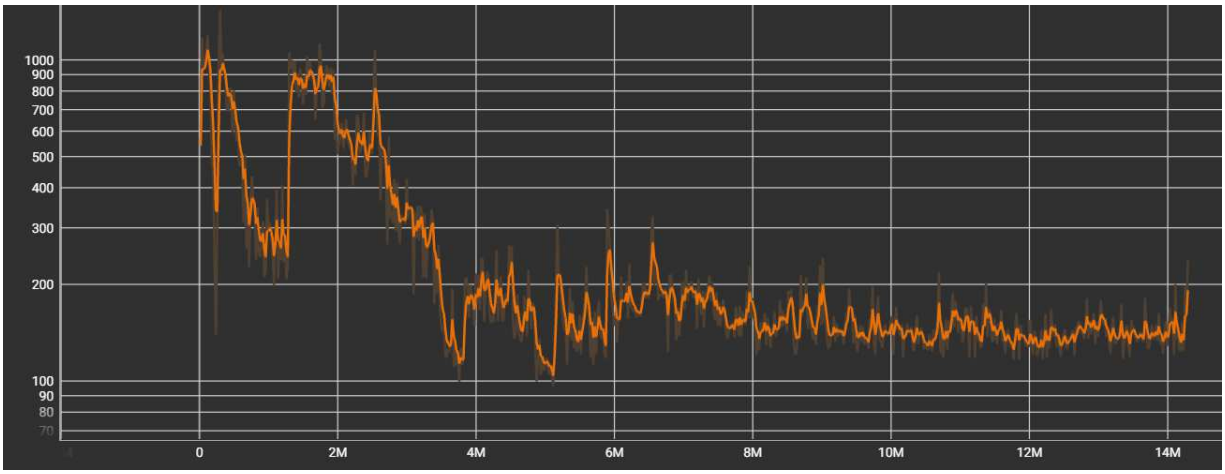
Pogledajmo sada graf koji prikazuje koja je lekcija aktivna u kojem periodu treniranja:



Graf 2 – aktivne lekcije (učenje kurikulumom)

Os ordinata ovaj put predstavlja broj lekciju u datom koraku treniranja. Na grafu je vidljivo da je najdulje trebalo za usvajanje treće lekcije koja predstavlja zadatak penjanja na najkompliciraniju platformu. Agent je najlakše usvojio prvu lekciju tj. skok na jednostavnu platformu s novčićem.

Zanimljivo je promatrati i trajanje jedne epizode treniranja kroz cjelokupni proces učenja:

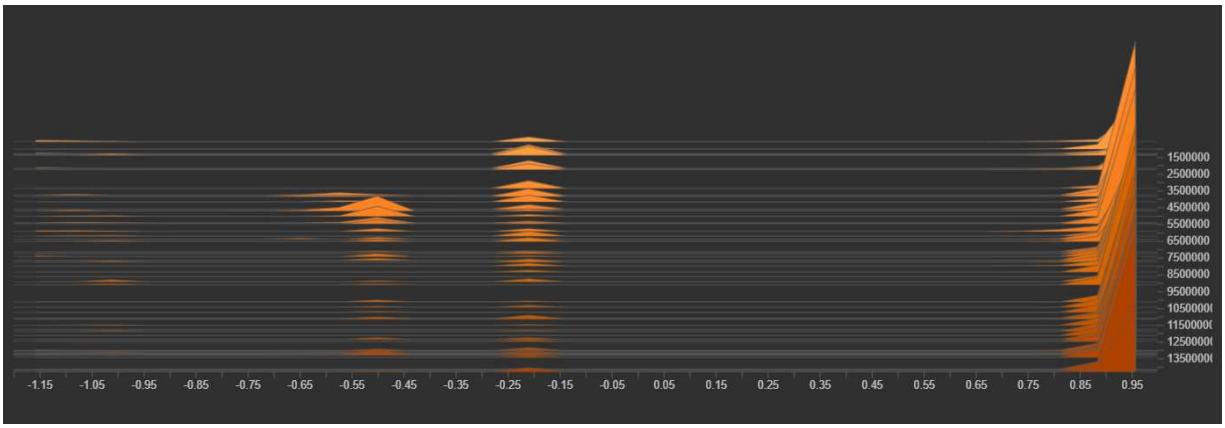


Graf 3 – trajanje epizode (učenje kurikulumom)

Ordinata prikazuje koliko je puta agent donio odluku u jednoj epizodi učenja što ne mora biti istovjetno broju koraka. Primjećujemo da su trajanja epizode pri početku treniranja duža. Razlog tome je što agent troši dosta vremena pokušavajući skočiti na

razne vrste platforma dok zabijanjem u šiljke pri kasnijim lekcijama ranije završava epizodu treniranja.

Naposlijetku ćemo sagledati histogram ukupne nagrade tijekom treniranja:

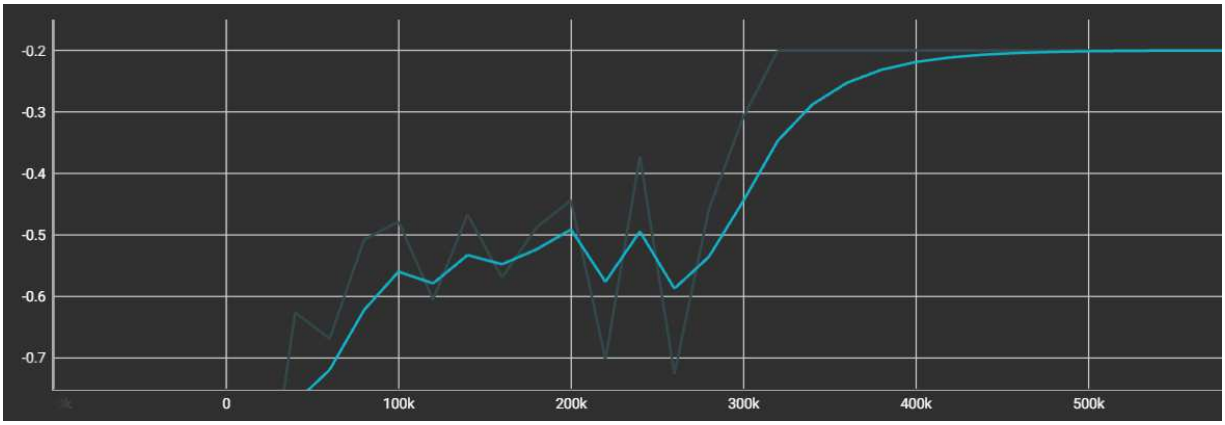


Graf 4 – histogram nagrade (učenje kurikulumom)

Apscisa prikazuje moguće vrijednosti nagrade, ordinata učestalost tih vrijednosti, a z-os predstavlja njihovo mijenjanje kroz vrijeme. Ističu se tri učestale vrijednosti nagrade. Najčešća je nagrada oko vrijednosti 0.95 koja predstavlja naučenu lekciju. Primjećujemo čestu negativnu nagradu u vrijednosti –0.5 koja se javlja zbog zabijanja agenta u šiljke i to vrlo naglo u trenu kada ih uvodimo u proces treniranja. Još jedna izražena vrijednost nagrade je –0.2 koja nastupa ako agentu istekne dozvoljeno vrijeme za jednu epizodu učenja.

7.2. Usporedba s treniranjem bez kurikuluma

Kako bismo ilustrirali važnost učenja s kurikulumom u ovom ćemo potpoglavlju usporediti taj pristup s klasičnim podržanim učenjem. U ovom primjeru dali smo agentu da proba riješiti zadatak bez korištenja kurikuluma.



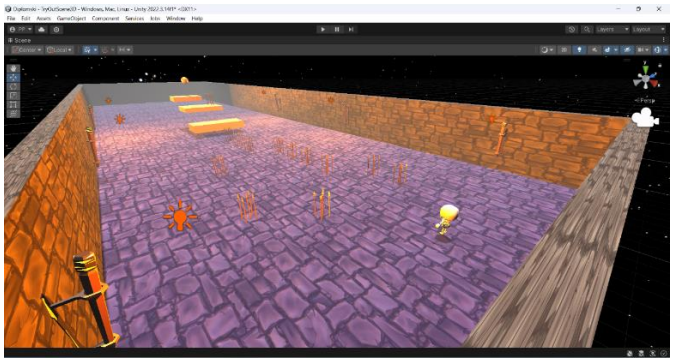
Graf 5 – kumulativna nagrada (klasični pristup)

Primjećujemo da agent u nijednom trenu nije postigao pozitivnu ukupnu nagradu. Nakon 500 tisuća koraka dolazi do stagnacije od vrijednosti –0.2 što znači da je agent odustao od pronalaženja veće nagrade te smatra da je –0.2 maksimalna nagrada koju može postići. Ta nagrada odgovara stajanju agenta na mjestu dok mu ne istekne vrijeme jedne epizode učenja.

8. Izrada ostatka Unity projekta

8.1. Izrada scene za testiranje agenta

Scena za testiranje agenta sastavljena je od raznih objekata koji su preuzeti s repozitorija *Unity Asset Store*. Korišteno je jedno glavno usmjereno svjetlo te nekoliko točkastih izvora svjetla u obliku baklja na zidovima. Zvjezdano nebo postavljeno je kao nebeski okvir (*skybox)* te je korištena opcija suptilnog dodavanja magle u sceni.

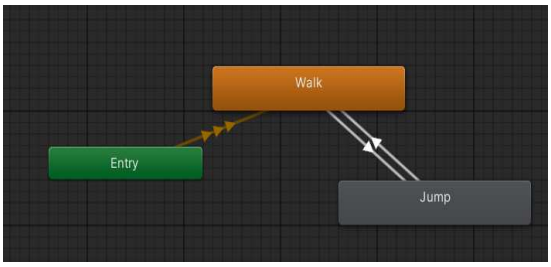


Slika 15 – scena za testiranje agenta

8.2. Model agenta i animacija kretanja

Za model agenta odabran je jednostavni kostur preuzet s repozitorija *Unity Asset Store*. Na agentov objekt dodana je komponenta *Animator* u kojoj se nalaze animacijska stanja objekta.

Slika 16 – model agenta



Slika 17 – animator komponenta agenta

Uočavamo dva stanja: *Walk* i *Jump* koja su povezana. Aktivacijom okidača imena „Jump“ iz skripte za kretanje pokrećemo tranziciju između stanja *Walk* u stanje *Jump*, a nakon završetka animacije skakanja model se vraća u svoju glavnu animaciju hodanja.

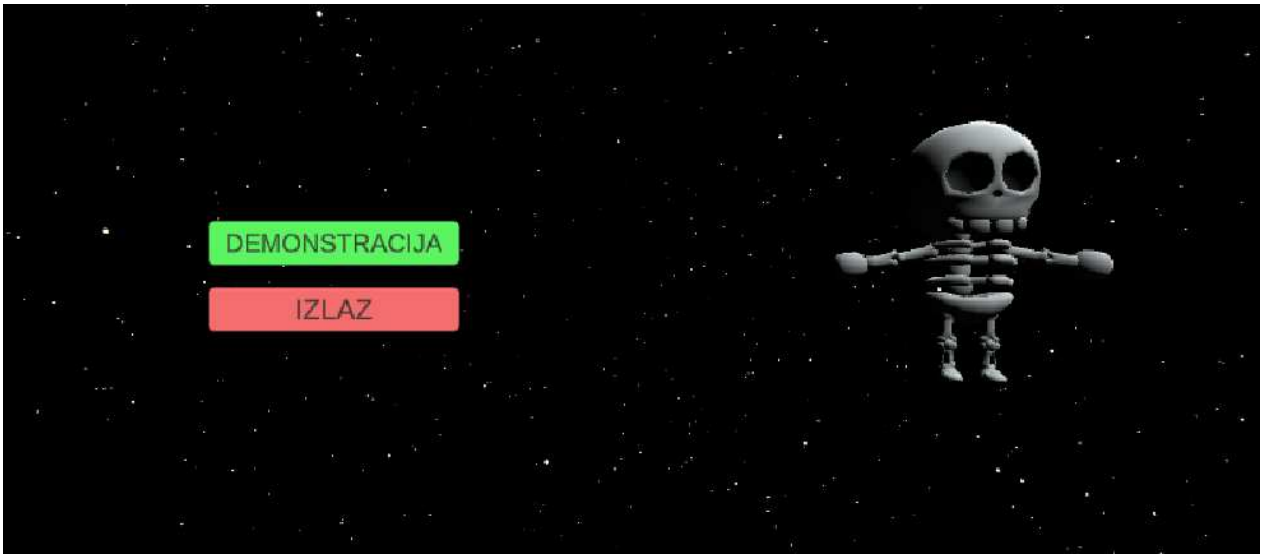
8.3. Testiranje agenta

Testiranje agenta možemo promatrati u sceni koja predstavlja kompleksan problem. Tri su platforme po kojima se agent mora penjati do novčića, a na putu do njih nalazi se duplo više šiljaka nego na poligonu za treniranje. Na jednoj od platformi također se nalaze šiljci. Agentova glavna skripta modificirana je tako da u lijevom gornjem dijelu ekrana možemo pratiti broj uspjelih i broj neuspjelih pokušaja te postotak uspješnosti agenta. Testiranje možemo prekinuti gumbom u desnom gornjem kutu koji igrača vraća u glavni izbornik.

Slika 18 – proces treniranja agenta

8.4. Glavni izbornik

Izrađen je glavni izbornik kroz koji pritiskom na zeleni gumb ulazimo u scenu za testiranje agenta. Pritiskom na crveni gumb gasimo aplikaciju.



Slika 19 – glavni izbornik

9. Zaključak

Ovim radom istraženo je korištenje alata ML-Agents u softveru Unity u svrhu treniranja inteligentnog agenta tehnikom podržanog učenja za prelazak jedne razine igre platformi. Razina igre implementirana je kao 3D scena u softveru Unity kroz koju se agent kreće ne bi li skupio novčić i time prešao razinu. Agent automatski gubi igru ako se zabije u zid ili šiljke koji se nalaze na podu, a treba i skakati po platformama ne bi li pokupio novčić.

Koristeći alat ML-agents isprobano je nekoliko pristupa ovom problemu koristeći razne kombinacije opažanja, akcija, nagrada, algoritama i ostalih postavki treniranja. Najbolje rezultate dalo je korištenje PPO algoritma strojnog učenja te uvođenje kurikuluma u proces treniranja. Razbivši kompleksan problem na više jednostavnijih, agent je postigao zadovoljavajuće rezultate u razumnom vremenu treniranja.

Agentovo naučeno znanje testiramo na kompleksnoj razini igre platformi koja sadrži više prepreka nego poligon za treniranje. Kroz testiranje dobivamo izvještaj o tome koliko je puta agent uspio završiti razinu.

Postoji prostor za daljnji napredak u vidu optimiziranja procesa učenja i uvođenja još kompleksnijih poligona. Daljnjim eksperimentiranjem s postavkama u alatu ML-agents i postavljanjem drugačijih kurikuluma moguće je da bi proces učenja trajao kraće i bio još stabilniji. Moguće je dodavanjem novih lekcija osposobiti agenta za rješavanje još kompleksnijih zadataka. U ovom pristupu korištene su samo rudimentarne akcije, no moguće je i koristiti kompleksnije akcije kao npr. aktivacija funkcije za skok u smjeru novčića ili pametnog kretanja do neke točke u sceni.

10. Literatura

1. „Big Cloud : The evolution of AI in gaming“, svibanj, 2021., Poveznica: https://bigcloud.global/the-evolution-of-ai-in-gaming/, datum pristupa: 7.8.2024.
2. „Unity (Game Engine) – Wikipedia“, lipanj, 2024., Poveznica: https://en.wikipedia.org/wiki/Unity_(game_engine), datum pristupa: : 7.8.2024.
3. „Visual Studio Code – Wikipedia“, listopad, 2023., Poveznica: https://en.wikipedia.org/wiki/Visual_Studio_Code, datum pristupa: 7.8.2024.
4. „ML-Agents Toolkit Overview“, travanj, 2024., Poveznica: https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/#running-example-training-npc-behaviors, datum pristupa: 7.8.2024.
5. „TensorBoard: TensorFlow's visualization kit“, Poveznica: https://www.tensorflow.org/tensorboard, datum pristupa: 7.8.2024.
6. „Machine Learning – Wikipedia“, lipanj, 2024., Poveznica: https://en.wikipedia.org/wiki/Machine_learning, datum pristupa: 7.8.2024.
7. „Nadzirano i nenadzirano učenje“, Poveznica: https://www.unite.ai/hr/supervised-vs-unsupervised-learning/, datum pristupa: 7.8.2024.
8. „Reinforcement Learning – Wikipedia“, svibanj, 2024., Poveznica: https://en.wikipedia.org/wiki/Reinforcement_learning#Exploration, datum pristupa: 7.8.2024.
9. „ML-Agents Toolkit – Training Configuration File“, Poveznica: https://unity-technologies.github.io/ml-agents/Training-Configuration-File/#common-trainer-configurations, datum pristupa: 11.8.2024.
10. „Github – ML-agents – Learning Environment Design Agents, Poveznica: https://github.com/yosider/ml-agents-1/blob/master/docs/Learning-Environment-Design-Agents.md#rewards, datum pristupa: 15.8.2024.
11. „Github – ML-agents – Training, Poveznica: https://github.com/yosider/ml-agents-1/blob/master/docs/Training-ML-Agents.md, datum pristupa: 15.8.2024.

Primjena podržanog učenja u računalnoj igri platformi

Sažetak

Cilj ovog rada je istraživanje korištenja podržanog učenja pri razvijanju računalne igre platformi. U programu Unity izrađena je okolina za treniranje, a samo učenje odvija se pomoću alata ML-Agents. Zadatak inteligentnog agenta je pokupiti novčić na kraju razine pritom se penjajući po platformama i izbjegavanjem šiljaka na podu. Eksperimentiranjem s različitim pristupima podržanom učenju odlučeno je koristiti vizualna opažanja, diskretne akcije te učenje kurikulumom. Rezultat rada je inteligentni agent koji konzistentno rješava spomenuti problem.

Ključne riječi: podržano učenje, platformer, Unity, ML-agents, razvoj igara, strojno učenje

Application of Reinforcement Learning in a Platformer Video Game

Abstract

The aim of this paper is to explore the use of reinforcement learning in the development of a platformer video game. A training environment was created in Unity, with the learning process carried out using the ML-Agents toolkit. The task of the intelligent agent is to collect a coin at the end of the level while climbing platforms and avoiding spikes on the floor. After experimenting with different approaches to reinforcement learning, it was decided to use visual observations, discrete actions, and curriculum learning. The outcome is an intelligent agent that consistently solves the aforementioned problem.

Keywords: reinforcement learning, platformer, Unity, ML-Agents, game development, machine learning