Računalni vid primjenom strojnog učenja u Unity razvojnom okruženju

Dumančić, Robert

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAČUNALNI VID PRIMJENOM STROJNOG UČENJA U UNITY RAZVOJNOM OKRUŽENJU

Diplomski rad

Robert Dumančić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 08.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Robert Dumančić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1048R, 06.10.2019.
OIB studenta:	36023142813
Mentor:	Izv. prof. dr. sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr.sc. Josip Job
Član Povjerenstva 1:	Izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 2:	Dr. sc. Hrvoje Leventić
Naslov diplomskog rada:	Računalni vid primjenom strojnog učenja u Unity razvojnom okruženju
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno trenirati računalni vid primjenom umjetne inteligencije u Unity razvojnom okruženju. Koristiti će se ML-Agents paket za treniranje inteligentnih agenata kako bi se pomoću jednostavnog Python API-ja koristeći metode strojnog učenja odredilo određeno ponašanje umjetne inteligencije. Cilj rada je istrenirati agenta koji će vizualno moći prepoznati objekte u svojoj okolini i reagirati na određeni način na njih. Tema rezervirana za: Robert Dumančić
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	08.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 24.09.2021.

Ime i prezime studenta:	Robert Dumančić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1048R, 06.10.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Računalni vid primjenom strojnog učenja u Unity razvojnom okruženju**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Nedavni napredci u umjetnoj inteligenciji vođeni su rastućom dostupnosti sve realističnijih i kompleksnijih simulacijskih okolina. Međutim, većina postojećih okolina učenja ne pruža zadovoljavajuće uvjete, poput nepreciznog ili netočnog modela fizike, ograničenosti agenta ili ograničene interakcije između više agenata. Štoviše, mnoge platforme ne nude mogućnosti podešavanja simulacije što pretvara okolinu učenja u crnu kutiju. Kako su algoritmi dubokog podržanog učenja postali sve sofisticiraniji, postojeće okoline i mjerila učinkovitosti bazirana na njima postaju manje relevantnima. ML-Agents alat napravljen je kao paket za Unity platformu koji pokušava riješiti navedene probleme. Kombiniranjem Unity razvojnog alata s ML-Agents paketom moguće je lagano stvoriti interaktivne, kompleksne i lagano podesive okoliše na kojima će se agenti trenirati.

U ovom radu će se trenirati umjetna inteligencija pomoću računalnog vida unutar Unity razvojnog okruženja. ML-Agents paketom će se pomoću jednostavnog Python API-ja, koristeći metode strojnog učenja, trenirati inteligentni agenti s određenim ponašanjem. Cilj je stvoriti agenta koji će vizualno moći prepoznati objekte u svojoj okolini i primjereno reagirati na njih, obavljajući određenu funkciju.

Započinje se s kratkim osvrtom na područje teme rada te se zatim nastavlja na korištene alate i tehnologije. Nakon toga se prolazi kroz mogućnosti ML-Agents paketa pri treniranju samih agenata. Sljedeće poglavlje je implementacija samog projekta i njegovi rezultati na konkretnom primjeru.

2. PREGLED PODRUČJA RADA

U posljednjem desetljeću, došlo je do značajnog napretka u području dubokog podržanog učenja i dizajna algoritama. Taj napredak je potican razvojem sve kompleksnijih i naprednijih okolina učenja koje su sve dostupnije širem broju ljudi kao i konferencijama poput ICLR (engl. *International Conference on Learning Representations*) koja se održava od svake godine od 2013. gdje akademski i industrijski istraživači, inženjeri i studenti prezentiraju svoja istraživanja bazirana na dubokom učenju [1].

Studenti s odjela za računarstvo sa sveučilišta u Iowa, SAD, su koristeći računalni vid treniran dubokim podržanim učenjem omogućili autonomno kretanje bespilotne letjelice kroz okoliš [2]. Letjelica je vrlo osjetljiva i skupa, stoga nije moguće treniranje izravno na njoj zbog rizika štete tijekom sudara. Prva treniranja su određena unutar simulacija s postepenim povećanjem težine. Na tom primjeru se vidi potreba za većim, realističnijim i modularnijim okolinama učenja. Pošto će letjelica naposljetku koristiti stvarnu kameru koja snima svoju okolinu, potrebno je imati okoliš za treniranje koji je dovoljno realističan kako bi kamera u simulaciji primala slične podatke kao kamera u stvarnosti.

Pregledom postojećih okolina za treniranje može ih se razvrstati u tri skupine. Prva skupina obuhvaća jedinstvenu okolinu koja djeluje kao crna kutija agentu. U tu skupinu spadaju okoline poput CartPole [3] gdje agent mora naučiti balansirati štap kretanjem lijevo i desno. Druga skupina su skupovi okolina koje obično sadrže slična promatranja i prostor djelovanja i zahtijevaju slične vještine kako bi se riješili. Primjer takvih okolina je ALE (engl. *Arcade Learning Environment*) koji je baziran na Atari 2600 igrama [4]. S razvojem metoda dubokog podržanog učenja, agenti trenirani na ALE su davno premašili ljudske sposobnosti i trenutno se koristi samo za usporedbu performansi algoritama. Pri korištenju boljih algoritama razlike su marginalne zbog jednostavnosti testne okoline. U trećoj skupini se nalaze platforme vezane za određenu domenu. Odnosi se na okoline koje sadrže zadatke unutar određene domene poput navigacije u prvom licu. Takve okoline obično sadrže ograničenja u perspektivi, fizičkim ograničenjima okoline ili interakcijama agenta s okolinom. Primjer takvog je VizDoom [5]. VizDoom je sustav za treniranje inteligentnih agenata koji će vizualnim promatranjima navigirati kroz prostor i boriti se s neprijateljima. Baziran na Doom videoigri, limitiran je na pogled iz prvog lica kao i na limitiranu simuliranu fiziku tog vremena. Cilj ML-Agents paketa bio je stvoriti četvrtu kategoriju koja omogućava stvaranje okolina s kompleksnim vizualnim, fizičkim i socijalnim interakcijama i zadatcima. Te okoline obuhvaćaju i proširuju sve navedene u prethodne tri kategorije [6]. Unityjev fokus na razvoj

univerzalnog sustava koji će podržavati veliki broj platformi, biti pristupačan razvojnim inženjerima raznog iskustva i podržavati različite tipove videoigara ga čini idealnom platformom za istraživanje umjetne inteligencije.

Koristeći Unity i ML-Agents stvoren je Obstacle Tower [7]. Cilj projekta je stvoriti novu naprednu okolinu za treniranje u 3D, trećem licu s proceduralno generiranom okolinom. Agent mora naučiti rješavati kontroliranje lika kao i rješavanje razine sve dok uči iz piksela i prima oskudne nagrade. Za razliku od ALE, evaluira se agentovo snalaženje u nepoznatoj okolini. Pri izradi Obstacle Tower-a testirani su svi suvremeni algoritmi dubokog podržanog učenja i svi su bili neuspješni u reproduciranju rezultata približnom ljudskom mjerilu. Iako izgled agenta ostaje isti, uz nasumičnost generiranja oblika prostorija, nasumično se generira i tekstura zidova što dodatno otežava agentima prolaz pošto se baziraju na vizualnim promatranjima.

3. KORIŠTENI ALATI I TEHNOLOGIJE

3.1. Unity

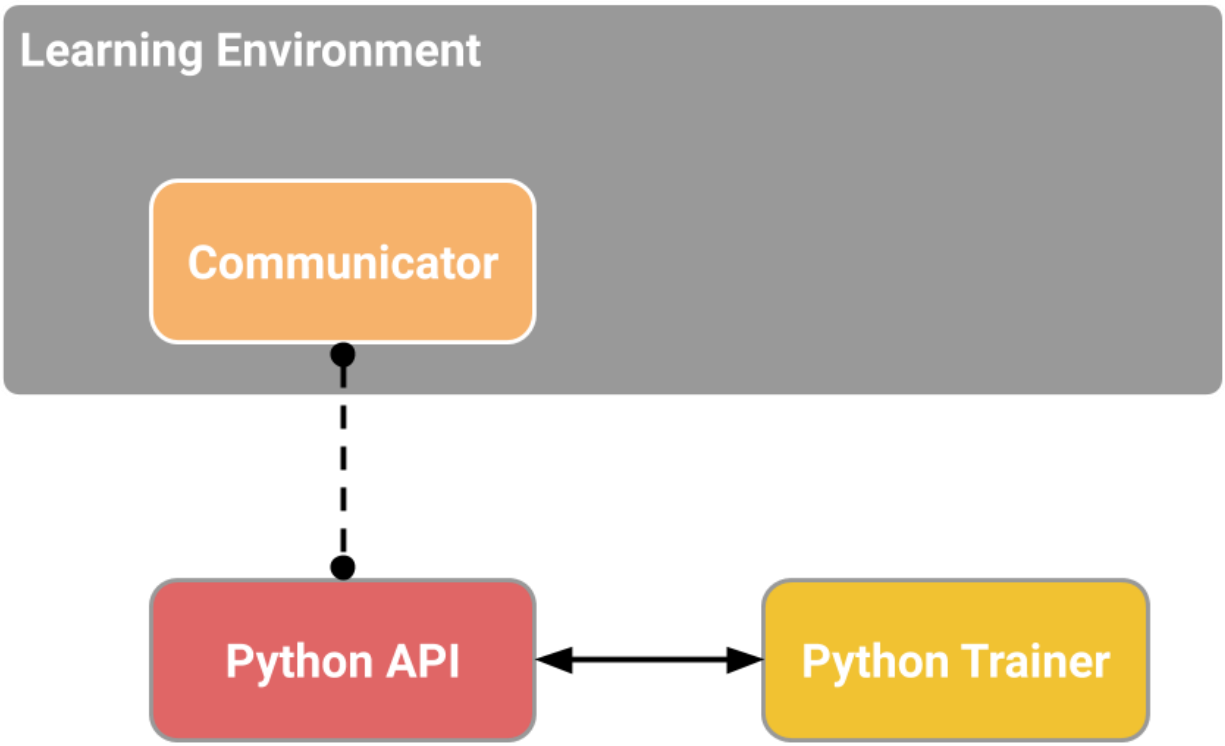
Unity je program za razvoj i upravljanje videoigrama na više platformi objavljen 2005. godine. Originalno dostupan isključivo na Mac OS X operacijskom sustavu, kasnije mu je podrška proširena na platforme za razvoj virtualne stvarnosti, igrače konzole, mobilne platforme i stolna računala. Unity se može koristiti za stvaranje trodimenzionalnih i dvodimenzionalnih videoigara kao i interaktivnih simulacija. Time mu se uporaba proširila i na automobilsku industriju u svrhe simulacije razvoja i performansi kao i razvoju sučelja unutar vozila, filmsku industriju u svrhe animacije i kinematografije kao i arhitektonsku industriju [8].

Unity pruža primarno pisanje skripti u C# programskom jeziku. Prije C#, podržavao je Boo programski jezik koji je izbačen u verziji 5 Unity-a, kao i verziju JavaScript-a zvanu UnityScript koju je u kolovozu 2017 nasljedio C#. Unity projekt se sastoji od skupa sredstava. Sredstva su obično datoteke unutar projekta (slike, skripte, zvučne datoteke i slično). Scene su poseban tip sredstava koje definiraju okolinu ili razinu projekta. Scene sadrže definiciju hijerarhijske kompozicije *GameObject*-a čiju funkciju opisuju komponente koje su mu dodijeljene. Unity uređivač dolazi s brojnim komponentama poput kamera i fizikalnih elemenata, ali omogućuje korisnicima i pisanje vlastitih komponenti koristeći C# skripte.

Unity uređivač je grafičko sučelje unutar kojega se nalaze alati potrebni za direktno stvaranje 2D i 3D scena, animacija i kinematike. Za dodatne funkcionalnosti unutar uređivača se nalazi upravitelj paketima. Paketi sadrže dodatne funkcije i mogućnosti koje su prilagođene potrebama korisničkog projekta. To podrazumijeva bilo koju ključnu funkcionalnost Unity-a koja je instalirana uz uređivač, kao i ostale pakete koje je moguće instalirati po potrebi poput onih koje su napravili drugi korisnici Unity-a i postavili na digitalnu trgovinu [9]. Ovaj rad se bazira na paketu ML-Agents kojega je moguće preuzeti direktno s upravitelja paketima.

3.2. ML-Agents

Unity ML-Agents je projekt otvorenog koda koji omogućava stvaranje okoline za treniranje umjetne inteligencije unutar Unity-a [10]. Istrenirani agenti imaju višestruke namjene poput kontroliranja likova kojima ne upravlja igrač, automatizirano testiranje verzija videoigara i evaluaciju odluka o aspektima igre. ML-Agents paket je obostrano koristan i za razvojne programere videoigara kao i istraživače umjetne inteligencije pošto pruža platformu na kojoj se



Slika 3.1. *Pojednostavljen blok dijagram ML-Agents paketa*

umjetna inteligencija može procijeniti na raznovrsnim okolinama i tada postati dostupna daljnem istraživanju i zajednicama razvojnih programera.

Na slici 3.1. [11] prikazan je pojednostavljeni blok dijagram ML-Agents projekta. ML-Agents projekt čine četiri glavne komponente koje se ne nalaze sve unutar ML-Agents Unity paketa već su podijeljene na više paketa, od kojih dio nije dostupan unutar upravitelja paketima.

- ML-Agents Python paket sadrži dvije komponente, Python aplikacijsko programsko sučelje(engl. *application programming interface, API*) niske razine i ulaznu točku za treniranje *mlagents-learn* koja omogućava treniranje agenata u Unity okolini koristeći predodređene implementacije podržanog učenja ili učenja imitacijom. Navedena ulazna točka će se isključivo koristiti za komunikaciju s Python API-jem. Navedeni paket je potrebno odvojeno instalirati i ne dolazi kao dio Unity paketa.
- Unutar Unity paketa se nalazi okolina učenja koja sadrži Unity scene i likove unutar videoigre. Unity scena pruža modularnu okolinu u kojoj se agent promatra i trenira. ML-Agents zapravo uključuje ML-Agents Unity set razvojnih alata (engl. *software development kit, SDK*) koji omogućavaju preobrazbu bilo koje Unity scene u okolinu učenja tako što se unutar navedene definiraju agenti i njihovo ponašanje. Ujedno se u

paketu nalazi i vanjski komunikator koji povezuje Python API s okolinom učenja i nalazi se unutar same okoline učenja.

- Odvojeni neobavezni paket (stoga nije prikazan na blok dijagramu) se zove *gym-unity*, koji služi kao omotač za način na koji strojno učenje komunicira s okolinom. Paket se koristi samo u slučaju kada se žele koristiti postojeći algoritmi strojnog učenja koji implementiraju dvoranu.

Dvije glavne komponente koje čine okolinu učenja su agent i ponašanje. Agenti su spojeni na Unity *GameObject* (koji podrazumijeva bilo koji lik unutar scene) i upravlja njegovim akcijama i dodjeljuje mu nagrade po potrebi čime mu određuje njegovo ponašanje. Ponašanje definira specifične atribute agenta poput broja koraka. Ponašanje se može prikazati kao funkcija koja prima agentova opažanja i nagrade i po tome mu dodjeljuje akciju. Može biti tri tipa ponašanja: učenja, heurističko i zaključivanjem. Ponašanje učenja je ono koje nije još definirano, ali će ga se istrenirati. Heurističko ponašanje kodirano je nizom pravila implementiranih programskim kodom. Ponašanje zaključivanjem je ono koje koristi već istreniranu neuronsku mrežu. Unutar jedne okoline učenja može postojati više agenata u isto vrijeme koji mogu, ali i ne moraju, imati isto ponašanje.

3.3. Programski jezici

Python API niske razine ML-Agents paketa se može koristiti izravno s Unity okolinom [12]. Takva okolina se može koristiti za razvoj novih algoritama za strojno učenje. Navedeni API koriste algoritmi za učenje unutar ML-Agents paketa, ali dozvoljava i pisanje vlastitih Python aplikacija. Unutar ovog rada se neće ulaziti u stvaranje vlastitih algoritama za učenje pa pisanje Python aplikacija nije potrebno, ali je Python svejedno potreban za uporabu *mlagents-learn* pristupne točke. Ujedno, preporučeno je i koristiti Python virtualnu okolinu pri korištenju paketa kako on ne bi utjecao na druge projekte na računalu.

C# je objektno orijentirani programski jezik korišten za izradnju aplikacija bazirane na .NET sustavu [13]. Mogućnosti C# pojednostavljuju programiranje u odnosu na C obitelj programskih jezika, na kojima je C# baziran. Kolektor smeća koji automatski oslobađa memoriju zauzetu nekorištenim objektima, *null* tipovi podataka za varijable koje ne referiraju objekte, upravljanje iznimkama, lambda izrazi, LINQ sintaksa i podrška za asinkrone operacije su samo neki od dostupnih mogućnosti C#. Ključne organizacijske strukture unutar C# su programi, prostor imena (engl. *namespace*), tipovi, članovi i sklopovi. Programi definiraju tipove, koji sadrže članove i mogu biti organizirani u prostore imena. Klase, strukture i sučelja su primjeri tipova. Polja, metode

i događaji su primjeri članova. Pri prevođenju C# programa, pakiraju se u sklopove. Ti sklopovi (engl. *assembly*) obično imaju .exe ili .dll nastavke ovisno o tome radi li se o aplikaciji ili biblioteci (engl. *linked library*).

U ovom radu C# će se koristiti za pisanje skripti za agente koji će određivati njegovo ponašanje i računati mu nagrade.

3.4. Dodatni alati

Mnogi algoritmi koji se koriste unutar ML-Agents paketa koriste neki oblik dubokog učenja, pogotovo *mlagents-learn*, koji je izrađen na bazi biblioteke PyTorch [14]. PyTorch je biblioteka otvorenog koda za izvođenje proračuna koristeći dijagrame protoka podataka. Dodjeljuje treniranje i zaključivanje na CPU i GPU unutar stolnog računala, servera ili mobilnog uređaja. Unutar ML-Agents paketa, pri treniranju agenta, izlazni model je .onnx datoteka koja se kasnije može dodijeliti agentu. Ukoliko se ne vrši vlastita implementacija algoritma za učenje, korištenje PyTorch-a je skriveno od korisnika i odvija se u pozadini.

Dodatna komponenta treniranja modela s PyTorch-om uključuje namještanje hiperparametara. Određivanje ispravnih vrijednosti hiperparametara može potrajati kroz nekoliko iteracija treniranja. Za pripomoć se koristi alat za vizualizaciju TensorBoard [15]. On omogućava prikaz određenih atributa agenta, poput nagrade, tijekom treniranja što može pomoći pri odabiru boljih vrijednosti hiperparametara.

4. METODE TRENIRANJA

U podržanom učenju krajnji je cilj agenta otkriti ponašanje koje će mu maksimizirati nagradu. Time se već zna kako je potrebno agentu dati nekakvu nagradu kao motivaciju za pronalaženje željenog ponašanja. Obično je nagrada definirana okolinom i podudara se s pronalaskom cilja. Takve nagrade se nazivaju vanjskima ili ekstrinzičnima pošto su definirane izvan algoritma za učenje. Nagrade se mogu definirati i izvan okoline, kako bi se potaklo određeno ponašanja agenta ili kako bi se ubrzalo učenje i navođenje agenta prema ekstrinzičnoj nagradi. Takve nagrade se nazivaju unutarnjima ili intrinzičnima. Ukupna nagrada koju agent primi je uobičajeno mješavina tih dviju nagrada.

ML-Agents omogućava definiranje modularnih signala nagrada kao i četiri signala nagrada koji se mogu kombinirati po potrebi:

- *extrinsic* – predstavlja nagradu definiranu unutar okoliša.
- *gail* – predstavlja intrinzičnu nagradu koju specifira GAIL (*Generative Adversarial Imitation Learning*) okvir za učenje.
- *curiosity* – predstavlja intrinzičnu nagradu koja potiče istraživanje u okolišu s oskudnim nagradama koju definira Curiosity modul.
- *rnd* – predstavlja intrinzičnu nagradu koja potiče istraživanje u okolišu s oskudnim nagradama koju definira Curiosity modul kao i prethodna, ali koristi drukčije izračune.

4.1. Duboko podržano učenje

Agent koristeći podržano učenje uči na pokušajima i greškama tijekom interakcije s okolinom tijekom kojeg pokušava pronaći optimalan put za maksimiziranje nagrade [16]. Duboko učenje podrazumijeva niz skrivenih slojeva neuronskih čvorova koji repliciraju strukturu ljudskog mozga. Na svakom sloju izuzevši ulazni, ulaz se računa iz sume ulaznih jedinica koji stižu iz prethodnog sloja. Tada se koristi nelinearna funkcija ili aktivacijska funkcija poput tanh ili ReLU (*Rectified Linear Unit*) kako bi se dobio novi prikaz ulaza prethodnog sloja. Nakon svih izračuna, greška se prenosi unatrag kroz mrežu kako bi se izmijenila funkcija gubitka i optimizirala mreža. Kombiniranjem podržanog učenja i dubokog učenja omogućava algoritmima primanje iznimno velike količine podataka, poput svakog piksela ekrana što je ključno za ovaj rad, i određivanje sljedeće akcije agenta iz nestrukturiranih podataka. Duboko podržano učenje se postiže primjenom dubokih neuronskih mreža na aproksimaciju bilo koje od sljedećih komponenti podržanog učenja:

model (funkciju promjene stanja i funkciju nagrade), funkciju vrijednosti stanja (trenutnog stanja) i ponašanje.

ML-Agents pruža implementaciju dva algoritama podržanog učenja:

- *Proximal Policy Optimization* (PPO)
- *Soft Actor-Critic* (SAC)

Algoritmi sadržavaju mnogo promjenjivih varijabli zbog kojih je teško ispravljati greške i zahtijevaju znatna podešavanja kako bi se postigli dobri rezultati [17]. PPO postiže dobar balans između jednostavnosti implementacije, kompleksnosti uzorka, i lakoće podešavanja. Uključuje skupljanje iskustava i njihovo korištenje za ažuriranje ponašanja agenta. Nakon toga se iskustva odbacuju i stvara se novi skup iskustava i postupak se ponavlja. Ključni dio PPO je da se ažuriranje ponašanja ne udaljava previše od prethodnog ponašanja što smanjuje varijancu u treningu [18].

SAC, za razliku od PPO, može učiti na osnovi prijašnjih iskustava. Kako se iskustva prikupljaju tako se stavljaju u spremnik iz kojega se mogu izvući u bilo kojemu trenutku tijekom treniranja agenta. To ga čini učinkovitijim od PPO na osnovi broja uzoraka, ali SAC zahtijeva češće dopune modela. Time je bolji izbor za teže i sporije okoline.

U okolinama gdje agent prima oskudne nagrade, može doći do slučaja gdje agent uopće ne primi nagradu što ne pridonosi procesu treniranja. U takvim slučajevima se preporuča koristiti intrinzične nagrade. *Curiosity* je jedan od prethodno navedenih signala nagrade koji se može iskoristiti. Koristeći ga, treniraju se dvije mreže, inverzni model koji uzima trenutno i sljedeće promatranje agenta, nkodira ih i predviđa akciju koja se izvršila između ta dva opažanja i model koji uzima trenutno nkodirano opažanje i akciju i predviđa sljedeće nkodirano promatranje. Razlika između predviđenog i stvarnog nkodiranog promatranja se koristi kao intrinzična nagrada, pa što je model iznenađeniji to je bolja nagrada.

Analogno s *curiosity* nagradom, *rnd* (*Random Network Distillation*) se koristi kao nagrada u okolinama s oskudnim nagradama pošto potiče agenta na istraživanje. Također koristi dvije mreže. Prva mreža sadrži nasumične vrijednosti koji se kodiraju zajedno s agentovim opažanjima. Druga mreža slične strukture je trenirana kako bi predvidjela izlazne veličine prve mreže i koristi agentova opažanja kao podatke za treniranje. Gubitak treniranog modela se koristi kao intrinzična nagrada. S povećanjem učestalosti agenta na određenom stanju to je točnije predviđanje i smanjuje se nagrada za to stanje što potiče agenta da istraži nova stanja s većim stupnjem pogreške pri predviđanju.

4.2. Učenje imitacijom

Učenje imitacijom koristi parove opažanja i akcija iz demonstracije kako bi naučio agenta određeno ponašanje. Kada se koristi samo po sebi, upotrebljava se kako bi se naučilo specifično ponašanje, tj. kako bi se naučilo agenta jednom specifičnom od mnogih načina rješavanja zadatka. Kada se koristi zajedno s podržanim učenjem ono znatno smanjuje vrijeme potrebno agentu da riješi zadatak. Pogotovo u okolinama s oskudnim nagradama.

ML-Agents pruža način treniranja direktno kroz demonstraciju kako bi se ubrzalo podržano učenje. Uključuje dva algoritma zvana BC (*Behavioral cloning*) i GAIL (*Generative Adversial Imitation Learning*). U većini slučajeva moguće je kombinirati algoritme. Demonstracije se mogu izravno snimiti koristeći Unity uređivač i spremiti u projekt.

GAIL direktno nagrađuje agenta za ponašanje slično onome iz seta demonstracija. Može se koristiti sa ili bez nagrada iz okoliša i radi najbolje na ograničenom broju demonstracija. Sadrži drugu neuronsku mrežu, diskriminatora, koji promatra agentove postupke i demonstraciju i dodjeljuje nagradu ovisno o vlastitoj procjeni koliko je dobro agent pratio demonstraciju. Svakim korakom treninga agent će se učiti maksimizirati nagradu dok diskriminator trenira bolje prepoznati razlike između demonstracije i agenta.

BC trenira agenta direktno oponašati akcije prikazane unutar demonstracija. BC ne može generalizirati izvan opsega demonstracija pa je idealno kada demonstracije obuhvate sva moguća stanja koja agent može iskusiti, ili u kombinaciji s GAIL ili ekstrinzičnim nagradama.

Sve tri metode BC, GAIL i RL (PPO ili SAC) se mogu koristiti samostalno ili u kombinaciji s drugima.

4.3. Učenje specifično okolini

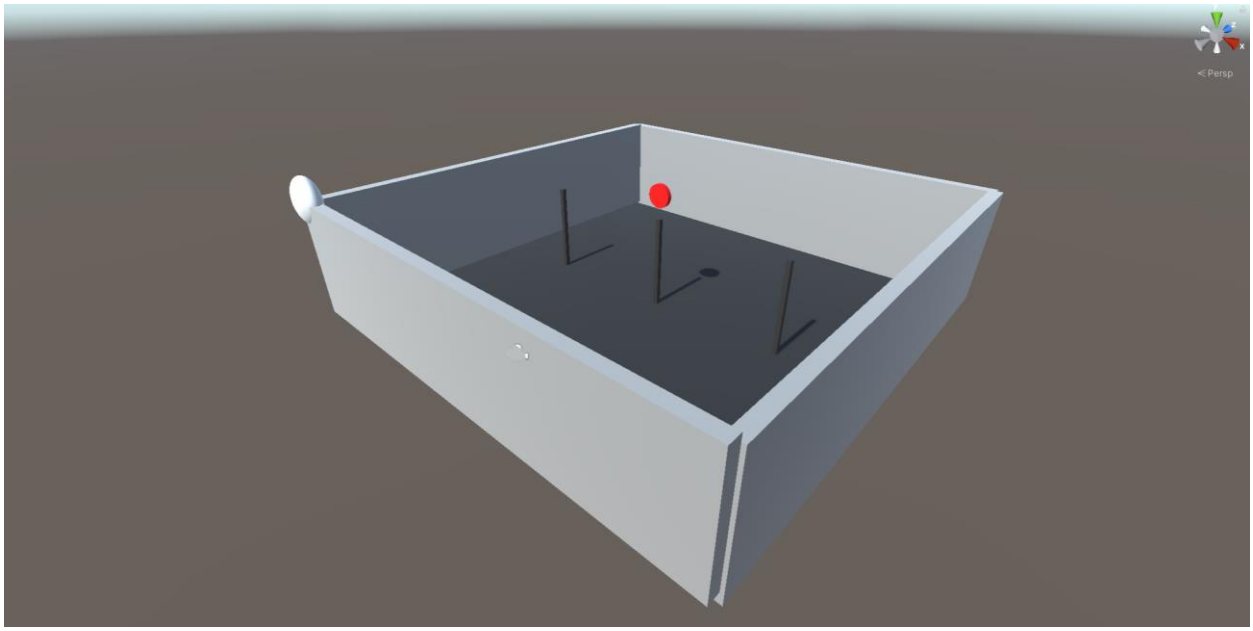
Uz prethodno navedene metode treniranja, ML-Agents nudi dodatne metode i alate treniranja agenata koje pomažu pri učenju na specifičnim tipovima okoline. Jedan takav trener je MA-POCA, koji predstavlja centraliziranu neuronsku mrežu koja služi kao trener za grupu agenata. Dodjeljuje nagrade cijelom timu i/ili individualnim agentima kako bi svi radili zajedno prema postizanju cilja. Tijekom treniranja agenti se mogu dodavati i micati iz grupe i svejedno primaju podatke o svom doprinosu timu, što omogućava agentima donošenje odluka koje doprinose cijelom timu čak i ako to dovede do agentovog micanja iz igre.

Ujedno veliki doprinos treniranju agenata koristeći ML-Agents dovodi učenje kurikulumom i nasumični odabir parametara. Učenje kurikulumom omogućava postepeno otežavanje okoline na način da je pred agentom uvijek optimalan izazov. Primjerice, agent počinje učenjem kako dostići cilj, tada se na put prema cilju uvedu prepreke i na kraju dodaju dodatni uvjeti za postizanje cilja. Uz to, nasumičnim odabirom parametara omogućava se agentu prilagoditi na promjene u okolišu. Ako se agent pomiče naprijed kako bi došao do cilja, on tehnički može shvatiti samo kretanje naprijed kao uvjet za pobjedu, umjesto dostizanje cilja, te ako se promijeni cilj na lokaciju iza agenta, agent će svejedno otići naprijed.

5. IZRADA PROJEKTA

5.1. Implementacija okoline

Prvotno je potrebno stvoriti okolinu u kojoj će agent obavljati svoj zadatak. U ovu svrhu stvorena je zatvorena prostorija u kojoj se nalaze 3 stupa i disk koji će predstavljati agentov cilj kao što se vidi na slici 5.1. Zidovi su svjetlije boje, a meta žarko crvene kako bi agent lakše prepoznao cilj. Na rub prostorije smješten je dodatni objekt, koji se nalazi u sloju kojeg agentova kamera ne vidi, kako bi se vizualizirala agentova uspješnost obavljanja zadatka. Mijenja boju u crvenu ako nije uspio i mijenja se u zelenu ako je agent uspio dostići cilj. Stupovi služe kao i cilj, ali i za ometanje agenta. Nasumično će se meta stvoriti na jednom stupu i taj stup agent mora dotaknuti kako bi uspješno riješio zadatak. Pri početku epizode, stupovi se postavljaju na nasumičnu poziciju, nasumične visine. Zidovima i stupcima koji ne sadržavaju metu dodana je prazna skripta nazvana Wall. Pri detekciji kolizije između agenta i objekta, ako objekt kojega je agent dotakao ima navedenu komponentu, agent odmah gubi i epizoda se resetira. Ujedno i cilju je dodana prazna skripta zvana Target i analogno se detektira agentov uspjeh u zadatku.



Slika 5.1. *Izgled okoline učenja*

Linija ***Kod***

```
1:      transform.localPosition = new Vector3(0,0,-8f);
2:      transform.rotation = Quaternion.identity;
3:      rigidbody.velocity = Vector3.zero;
4:
5:      pillar1.localScale = new Vector3(0.2f, Random.Range(1f, 4f), 0.2f);
        pillar1.localPosition = new Vector3(Random.Range(-9f, -3f),
6:      (pillar1.transform.localScale.y / 2), Random.Range(0f, 9f));
7:
8:      pillar2.localScale = new Vector3(0.2f, Random.Range(1f, 4f), 0.2f);
        pillar2.localPosition = new Vector3(Random.Range(-2.75f, 2.75f),
9:      (pillar2.transform.localScale.y / 2), Random.Range(0f, 9f));
10:
11:     pillar3.localScale = new Vector3(0.2f, Random.Range(1f, 4f), 0.2f);
        pillar3.localPosition = new Vector3(Random.Range(3f, 9f),
12:     (pillar3.transform.localScale.y / 2), Random.Range(0f, 9f));
13:
14:     int pillarRNG = Random.Range(0, 3);
15:
16:     if(pillarRNG == 0)
17:     {
            targetTransform.localPosition = new Vector3(
        pillar1.localPosition.x, pillar1.localPosition.y * 2f +0.5f,
18:     pillar1.localPosition.z);
19:     } if(pillarRNG == 1)
20:     {
            targetTransform.localPosition = new Vector3(
        pillar2.localPosition.x, pillar2.localPosition.y * 2f +0.5f,
21:     pillar2.localPosition.z);
22:     } if(pillarRNG == 2)
23:     {
            targetTransform.localPosition = new Vector3(
        pillar3.localPosition.x, pillar3.localPosition.y * 2f +0.5f,
24:     pillar3.localPosition.z);
25:     }
        targetDistance = new Vector3(targetTransform.localPosition.x, 0,
26:     targetTransform.localPosition.z);
        distance = Mathf.FloorToInt(Vector3.Distance(targetDistance,
27:     transform.localPosition));
```

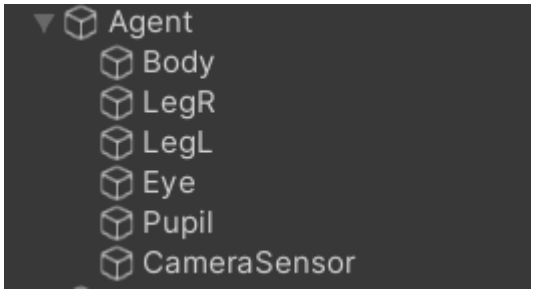
Slika 5.2. *Funkcija OnEpisodeBegin*

Inicijaliziranje okoline je obavljeno unutar agentove skripte u funkciji OnEpisodeBegin koju je potrebno nadjačati i dio je Agent klase koja dolazi s ML-Agents paketom. Navedena funkcija se poziva na početku treniranja, kao i nakon svakog poziva funkcije EndEpisode. Zato se u OnEpisodeBegin postavljaju sve vrijednosti potrebne za početak treniranja, kao i vraćanje agenta u početnu poziciju što je prikazano isječkom koda sa slike 5.2.

Pozicioniranje objekata se odvija koristeći lokalnih pozicija umjesto globalnih kako bi skripta radila neovisno o globalnoj poziciji okoline učenja što će se pokazati potrebnim pri treniranju. Ujedno se računa i početna udaljenost agenta od cilja kako bi se pomoću te vrijednosti kasnije uspostavila nagrada agentu. Za udaljenost se koriste cijeli brojevi kako se ne bi poticalo agenta da radi sitne korake za maksimiziranje nagrade.

5.2. Stvaranje agenta

Agent je ključni dio okoline učenja koji predstavlja člana okoline kojega se pokušava istrenirati obavljanje određene radnje. Stvoren je enkapsulacijom više Unity objekata u jedan *GameObject* u Unity uređivaču kao što je prikazano na slici 5.3. Enkapsulacija omogućava upravljanje svim objektima kao cjelinom te će se sve skripte dodavati na Agent objekt koji se vidi na slici 5.4. Prvo je potrebno agentu dodati *Rigidbody* komponentu koja se koristi za kontroliranje pozicije objekta kroz simuliranje fizike. Zatim se dodaje detektor kolizija. Pri dodavanju detektora kolizija



Slika 5.3. *Prikaz agent GameObject-a*

Slika 5.4. *Izgled agenta*

potrebno ga je dodati samo na enkapsulirajući objekt i maknuti ga sa svih drugih članova kako bi se spriječili konflikti i neželjeno ponašanje. U ovom slučaju korišten je običan detektor u obliku pravokutnika koji obuhvaća tijelo i noge agenta, gdje noge nisu specifično definirane već okvirno zajedno s tijelom. Agentu se potom stvara CharacterController skripta koja će biti zaslužna samo za pomicanje agenta. Skripta je prikazana na slici 5.5. Kada se bude vršilo treniranje skripta zaslužna za agentovo ponašanje će pozivati navedenu skriptu za pretvaranje svojih vrijednosti u agentovo kretanje.

Varijable za brzinu kretanja i okretanja su stavljene kao javne kako bi se njihova vrijednost mogla podešavati izravno iz uređivača. TurnInput i ForwardInput će sadržavati vrijednosti kojima će se obavljati kretnja i njihove vrijednosti se dodjeljuju tijekom treniranja. Njihove očekivane vrijednosti su od minus jedan do jedan, gdje minus jedan označava kretanje unazad punom

Linija ***Kod***

```
1:      public float moveSpeed = 2f;
2:      public float rotateSpeed = 300;
3:      new private Rigidbody rigidbody;
4:
5:      public float TurnInput {get; set; }
6:      public float ForwardInput {get; set; }
7:
8:      private void Awake()
9:      {
10:         rigidbody = GetComponent<Rigidbody>();
11:     }
12:
13:     private void FixedUpdate()
14:     {
15:         HandleMovement();
16:     }
17:
18:     private void HandleMovement()
19:     {
20:         if(TurnInput != 0f)
21:         {
22:             float angle = Mathf.Clamp(TurnInput, -1f, 1f) * rotateSpeed;
23:             transform.Rotate(Vector3.up, Time.fixedDeltaTime * angle);
24:         }
            Vector3 move = transform.forward * Mathf.Clamp(ForwardInput, -
25:     1f, 1f) * moveSpeed * Time.fixedDeltaTime;
26:         rigidbody.MovePosition(transform.position + move);
27:     }
```

Slika 5.5. *CharacterController skripta*

brzinom i okretanje na desnu stranu dok jedan označava kretnju punom brzinom naprijed i okretanje na lijevo. Nula oboma označava da se ne kreću. Awake je funkcija koja se koristi za inicijaliziranje varijabli prije početka aplikacije te se u njoj pridružuje lokalnoj varijabli *Rigidbody* komponenta agenta. Druga funkcija je FixedUpdate, koja se poziva svakih 0.02 sekundi. Ta se funkcija preporuča za dodjeljivanje sile i rad s *Rigidbody* komponentom pošto se ažuriranje simulirane fizike odvija u mjerenim koracima koji se ne podudaraju s ažuriranjem okvira igre. FixedUpdate se poziva trenom prije svakog ažuriranja fizike kako bi se bilo kakve promjene trenutno procesirale. HandleMovement funkcija upravlja samom kretnjom. Provjerava TurnInput i ForwardInput i primjenjuje pomicanje i rotaciju. Ulazne vrijednosti su ograničene na raspon od minus jedan do jedan kako bi se spriječilo prebrzo kretanje. Kretanje se vrši promjenom pozicije objekta bez primjene sile te se može promatrati kao niz teleportacija. Uzevši u obzir da se ne dodjeljuje sila, trenje se zanemaruje. Vrijednosti se množe s fixedDeltaTime koji predstavlja interval u sekundama u kojima se fizika ažurira s čime se postiže kretanje objekta po sekundi umjesto po pozivu FixedUpdate funkcije.

Nakon CharacterController skripte potrebno je napraviti novu skriptu koja će kontrolirati agenta. Potrebno joj je promijeniti nasljeđivanje tako da nasljeđuje Agent klasu. Agent klasa sadrži brojne funkcionalnosti vezane za treniranje agenata i strojno učenje. Prva funkcija u skripti je Initialize koju Agent klasa automatski poziva i koristi se za inicijalizaciju agenta kao što se vidi na slici 5.6. Za razliku od druge funkcije koju je potrebno dodati, OnEpisodeBegin koja se vidi na slici 5.2. Initialize se poziva samo jednom po treniranju, i vrijedi za više epizoda treniranja. U OnEpisodeBegin se još dodatno vraća agent na početnu poziciju i rotaciju kao i namješta brzina na nulu kako bi se spriječilo neželjeno ponašanje poput agentovog propadanja kroz pod.

Linija ***Kod***

```
1:      public override void Initialize()
2:      {
3:          characterController = GetComponent<CharacterController>();
4:          rigidbody = GetComponent<Rigidbody>();
5:      }
```

Slika 5.6. *Funkcija Initialize*

Potom se dodaje funkcija kojom će se definirati kraj epizode. Navedena funkcija nije dio Agent klase već se poziva pri detekciji kolizije *GameObject-a* s okidačem. Okidači su u ovom slučaju zidovi i stupci za negativan rezultat i cilj za pozitivan. Pri detekciji kolizije s objektom koji ima komponentu Wall, agentu se dodjeli negativna nagrada minus jedan, promijeni se statusni objekt

Linija ***Kod***

```
1:      private void onTriggerEnter(Collider other)
2:      {
3:          if(other.TryGetComponent<Wall>(out Wall wall))
4:          {
5:              AddReward(-1f);
6:              status.material = loss;
7:              EndEpisode();
8:          } if(other.TryGetComponent<Target>(out Target target))
9:          {
10:             AddReward(+1f);
11:             status.material = win;
12:             EndEpisode();
13:         }
14:     }
```

Slika 5.7. *Funkcija detektora kolizija*

u crvenu boju kako bi simbolizirao neuspješni rezultat i završava se epizoda. Pri detekciji kolizije s ciljem, slično prethodnome, agentu se dodaje pozitivna nagrada, status se mijenja u zeleno i završava se epizoda kao što je prikazano na slici 5.7. ML-Agents paket sadrži dvije funkcije kojima se može dodavati nagrada, AddReward i SetReward. SetReward namješta baznu nagradu na određenu vrijednost dok AddReward mijenja nagradu u odnosu na baznu. Pošto je bazna nagrada nula, nju se neće mijenjati, već će se koristiti AddReward za kumulativnu nagradu.

Kako bi se isprobao kod agenata potrebno je nadjačati funkciju Agent klase zvanu Heuristic. Ta funkcija omogućava ručnu kontrolu agenta kako bi se okolina ispitivanja mogla ručno ispitati prije nego se kontrola prepusti neuronskoj mreži. Kao što se vidi na slici 5.8., u funkciji je samo potrebno mapirati dvije osi kretanja agenta s ulaza tipkovnice i poslati vrijednosti

Linija ***Kod***

```
1:      private override void Heuristic(in ActionBuffers actionsOut)
2:      {
3:          ActionSegment<int> actions = actionsOut.DiscreteActions;
4:          int vertical = Mathf.RoundToInt(Input.GetAxisRaw(„Vertical“));
            int horizontal =
5:      Mathf.RountToInt(Input.GetAxisRaw(„Horizontal“));
6:
7:          actions[0] = vertical >= 0 ? vertical : 2;
8:          actions[1] = horizontal >= 0 ? horizontal : 2;
9:      }
```

Slika 5.8. *Funkcija Heuristic*

CharacterController-u. ActionSegment je struktura unutar Agent klase u kojoj se nalaze vrijednosti koje agent može primiti. Pošto se u ovom slučaju radi s diskretnim vrijednostima, agent prima nulu, jedinicu ili dvojku. Kretanje tipkovnicom uzima vrijednosti od minus jedan do jedan tako da se vrijednosti ispod nule dodjeljuju dvojci ActionSegment-a.

Zadnja funkcija je OnActionReceived. Ta funkcija definira ponašanje agenta tijekom treniranja, to jest, određuje djelovanje agenta u odnosu na vrijednost ActionSegment-a koju primi. Također, pošto se funkcija izvršava nakon svake agentove akcije, u nju se dodaju dodatne kumulativne nagrade agentu. Prvo se pretvaraju ulazi nula, jedan i dva u raspon od minus jedan i jedan za kretanje agenta. Tada se te vrijednosti predaju CharacterController-u i vrši se pomicanje. Nakon toga se računa trenutna udaljenost agenta od cilja i uspoređuje s onom određenom na početku epizode. Za svaki metar koji se približi cilju dobije sitnu nagradu kako bi se agent brže istrenirao pravilnom ponašanju. Nagrada se dodjeljuje samo prvi puta kako agent ne bi išao naprijed natrag u cilju maksimiziranja nagrade. Funkcija je prikazana na slici 5.9.

Linija ***Kod***

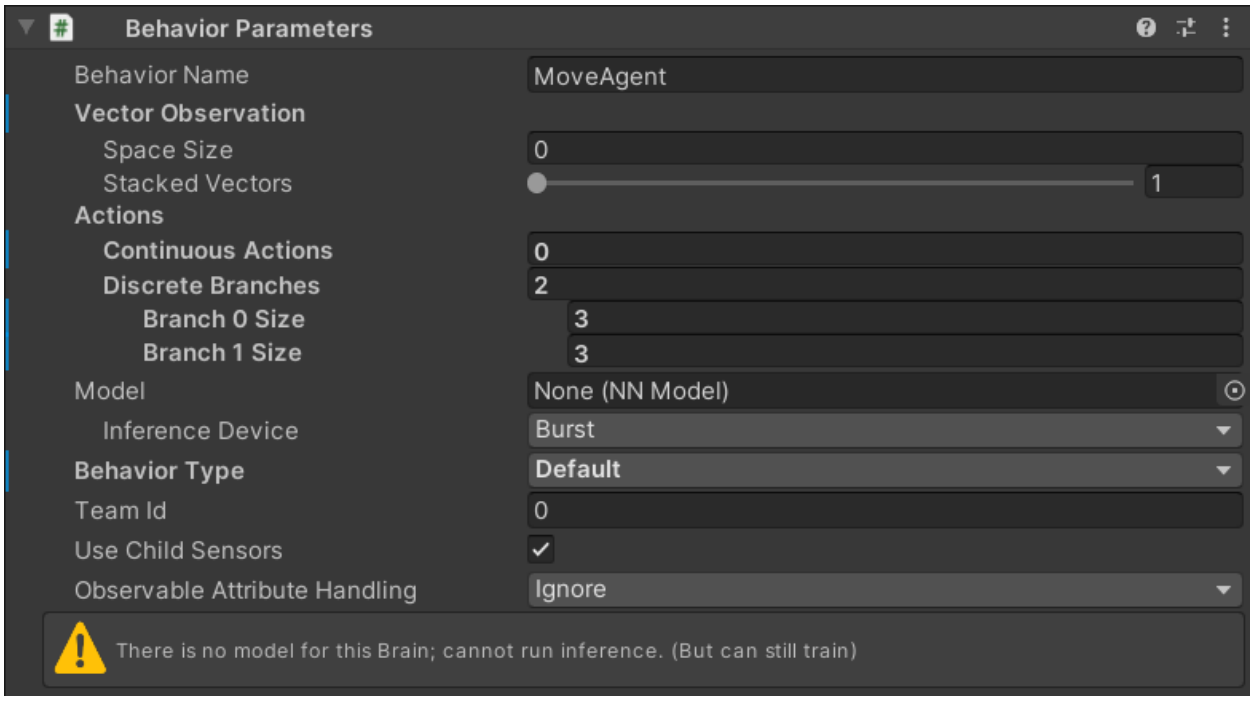
```
1:      private override void OnActionReceived(ActionBuffers actions)
2:      {
            float vertical = actions.DiscreteActions[0] <= 1 ?
3:      actions.DiscreteActions[0] : -1;
            float horizontal = actions.DiscreteActions[1] <= 1 ?
4:      actions.DiscreteActions[1] : -1;
5:
6:          characterController.ForwardInput = vertical;
7:          characterController.TurnInput = horizontal;
8:
            distanceBuffer =
       Mathf.FloorToInt(Vector3.Distance(targetDistance,
9:     transform.localPosition))
10:         if(distanceBuffer == distance - 1)
11:         {
12:             AddReward(0.1f);
13:             distance = distanceBuffer;
14:         }
15:     }
```

Slika 5.9. *Funkcija OnActionReceived*

Nakon što su obje skripte dodane agentu, potrebno je napraviti dodatne izmjene komponenti. Kada se agentu doda skripta koja nasljeđuje Agent klasu, u uređivaču se pojavljuje dodatna varijabla Max Step na skripti. Izmjenom vrijednosti te varijable se mijenja maksimalan broj koraka epizode prije nego što se automatski pozove EndEpisode. Ujedno, dodaje se i nova komponenta

Behaviour Parameters. Unutar te komponente se namještaju parametri agenta kao što se vidi na slici 5.10. Unutar te komponente se nalazi:

- *Behavior name* – određuje ime komponente kao i ime modela koji će se stvoriti treniranjem.
- *Vector Observations* – koristi se za definiranje vektorskih promatranja agenata
- *Actions* – određuje tip i količinu agentovih akcija. Postoje dvije vrste, kontinuirane koje daje realne brojeve između minus jedan i jedan i kontinuirane koje daju cjelobrojne vrijednosti koje kreću od nule i završavaju ovisno o veličini zadanoj pod veličinom grane. Kod korištenja diskretnih akcija potrebno je dati broj grana i veličinu grana
- *Model* – ovdje se stavlja istrenirani model kako bi se agent ponašao po modelu. Moguće je pokrenuti treniranje s modelom aktivnim kako bi se vršilo učenje kurikulumom
- *Behavior Type* – određuje način ponašanja agenta. Moguće opcije su *Default* gdje se vrši treniranje, *Heuristic* za ručno upravljanje agentom i *Inference* gdje se agent ponaša po istreniranom modelu
- *Team Id* – opcija za treniranje grupa agenata, agenti koji dijele isti identifikator se promatraju kao agenti u istom timu
- *Use Child Sensors* – Opcija za korištenje drugih komponenti pridruženih objektu na kojem se ova komponenta nalazi. Koristi se kada agent koristi kameru ili *ray casting* za promatranje okoline



Slika 5.10. *Postavke Behavior Parameters komponente agenta*

Sljedeće je potrebno dodati komponentu zaslužnu za zahtijevanje odluka za agenta u određenom intervalima zvanu Decision Requester. Moguće je i izostaviti Decision Requester, ali je onda potrebno ručno pozivati funkciju koja zahtjeva odluku u nekom intervalu. Sadrži samo dvije postavke, jedna određuje unutar koliko agentovih koraka će se poslati zahtjev za novom odlukom i opcija koja određuje smije li agent samostalno izvršavati akcije između zahtjeva.

Zadnja komponenta je sama kamera kojom će agent „gledati“. Komponenta se naziva Camera Sensor. Ona sadrži postavke:

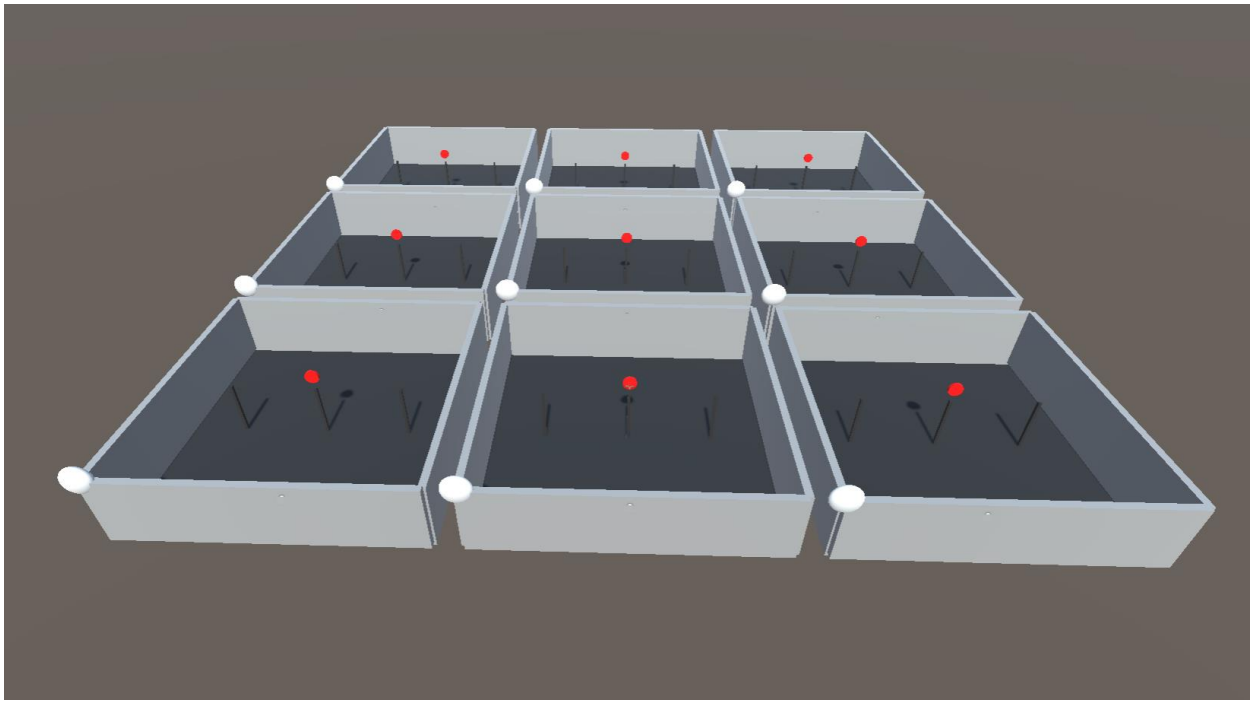
- *Camera* – prazno polje u koje se stavlja Unity *GameObject* kamera.
- *Sensor Name* – ime senzora
- *Width* – zajedno s visinom određuje u pikselima rezoluciju kamere
- *Height* – zajedno s širinom određuje u pikselima rezoluciju kamere
- *Grayscale* – opcija za pretvaranje slike kamere u crno bijelu sliku. Koristi se kako bi se smanjio broj promatranja po pikselu i kompleksnost obrade podataka
- *Observation Stacks* – Određuje broj promatranja koji se uzimaju u obzir pri donošenju odluke. Može se koristiti za određivanje smjera objekta u pokretu
- *Compression* – tip kompresije slike

5.3. Treniranje agenta

Prije početka treniranja potrebno je namjestiti sve parametre komponenti agenta. Unutar Behavior Parameters potrebno je odrediti dvije grane diskretnih akcija veličine tri. Jedna grana će predstavljati rotaciju agenta, a druga kretanje naprijed i nazad. Obavezno odabrati opciju korištenja vanjskih senzora kako bi agent znao da treba vršiti očitavanja s kamere. Pod agentovom skriptom namjestiti maksimalan broj koraka na 4000. Ta vrijednost se pokazala dovoljnom kako bi agent stigao do cilja, a u slučaju da pogriješi, ima dovoljno vremena da se ispravi. Unutar Camera Sensor dodjeliti agentovu kameru komponenti. Ova kamera može biti jednaka glavnoj kameri koja se stvori pri otvaranju Unity uređivača, ali u ovom slučaju će se koristiti odvojena kamera koja se stavlja u agentov *GameObject*. Rezoluciju kamere postaviti na 64 sa 64 piksela. Na agentovoj kameri je moguće mijenjati i vidno polje kamere, što u ovom slučaju neće biti potrebno, ali će se kut pogleda kamere podići za 20 stupnjeva kako bi agent bolje uočio metu kada stoji neposredno ispod viših stupaca. Dodatna neobavezna opcija je dubina kamere, ona određuje koja će se kamera koristiti kao glavna kamera u igri. U ovom slučaju pošto scena sadrži glavnu kameru kojom će se promatrati, dubina agentove kamere se stavlja na dubinu ispod dubine glavne kamere. Bitno je

dodijeliti objekte koje kamera mora vidjeti u poseban sloj i omogućiti kameri pogled samo nad tim slojem. U ovom slučaju to podrazumijeva zidove, pod, stupce i metu.

Kako bi se dodatno ubrzalo treniranje, svi dosada stvoreni objekti će se zajedno enkapsulirati u novi objekt. Taj objekt je tada moguće dodati kao dio projekta u datoteke. Mijenjanjem novo nastalog objekta mijenjaju se sve stvorene instance tog objekta. Sada je moguće stvoriti više kopija okoline učenja na kojima će se paralelno izvoditi treniranje kao što se vidi na slici 5.11. Zbog toga je u funkciji prikazanoj na slici 5.2. bilo potrebno koristiti lokalne vrijednosti za poziciju kako se pri ponovnom generiranju epizode okoline ne bi preklapale u jednoj globalnoj poziciji.



Slika 5.11. *Izgled dupliciranih okolina učenja*

Hiperparametri treniranja se nalaze u posebnoj yaml datoteci koju se treba referencirati pri pokretanju treniranja. U slučaju korištenja virtualne okoline pri treniranju, datoteka s navedenim hiperparametrima se mora nalaziti unutar direktorija projekta. Prikaz hiperparametara koji se koriste u radu na slici 5.12.

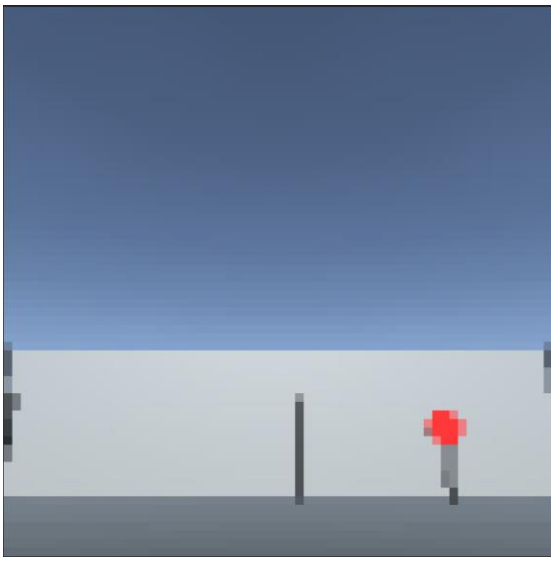
U ovom radu se trenira samo jedan agent (iako su dupli cirani radi bržeg treniranja, svejedno se trenira samo jedan model kojega dijele svi agenti) i nastali model će se nazvati MoveAgent. Korišteni algoritam je PPO. Pošto se koristi PPO u hiperparametrima je namješteno da se mijenja ponašanje agenta nakon 2048 skupljenih iskustava s 128 u svakoj iteraciji gradijentnog spusta. Početna stopa učenja gradijentnog spusta je postavljena na 0.0003 s linearnom promjenom kroz

```yaml
behaviors:
  MoveAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 2048
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 256
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 2000000
    time_horizon: 128
    summary_freq: 20000
    threaded: true
```

Slika 5.12. *Hiperparametri treniranja*

vrijeme. Beta određuje snagu entropijske regulacije, što čini učenje više nasumičnim i potiče agenta da istražuje nepoznato. Epsilon određuje stopu kojom se promjene odvijaju tijekom treniranja. Odnosi se na odstupanje između stare i nove metode tijekom ažuriranja gradijentnog spusta. Lambda je generalizacijski parametar koji se koristi kao vrijednost koja opisuje koliko se agent oslanja na pretpostavljene vrijednosti kada računa sljedeći korak i obično se koristi vrijednost u rasponu između 0.9 i 0.95. Broj epoha opisuje broj prolaza kroz spremnik iskustava kada se izvodi gradijentni spust. Ujedno se u konfiguracijskoj datoteci nalaze i postavke neuronske mreže kao i nekolicina ostalih. Za neuronsku mrežu koristi se mreža s 2 skrivena sloja koja sadržavaju 256 čvorova bez korištenja normalizacije. Broj čvorova je povećan kako bi se kompenziralo treniranje koristeći veliki broj ulaznih vrijednosti (slika u boji). Nagrada agenta je isključivo definirana unutar okoline stoga je snaga navedenih maksimalna. Treniranje se izvodi kroz dva milijuna koraka sa statusnim porukama svakih 20000 koraka. Svakih 128 koraka se gleda kao jedno iskustvo agenta. Okolinama je dozvoljeno odrađivati korake dok se ažurira model. ML-Agents pruža opširnu dokumentaciju [19] svih opcija i mogućnosti konfiguracijske datoteke.

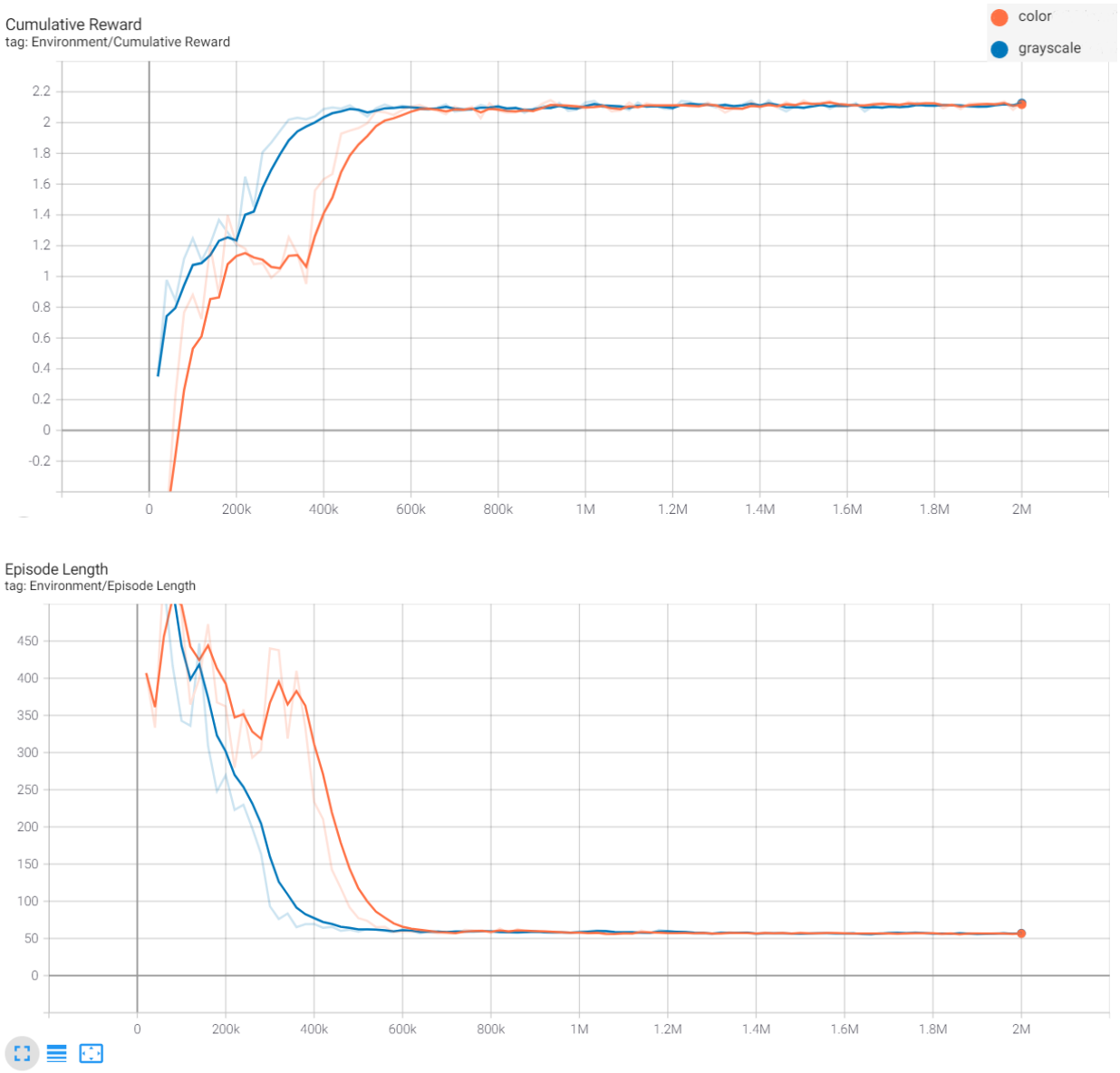
Izvršila su se dva testiranja, jedno sa slikom u boji i drugo monokromatskom slikom. Nije bilo potrebno mijenjati rezoluciju kamere sa 64 s 64 piksela, pošto je s manjim vrijednostima teže uočiti cilj iz udaljenosti, a veće vrijednosti samo usporavaju treniranje i povećavaju veličinu treniranog modela. Po slici 5.13. se vidi kako je cilj na zadanim postavkama dovoljno vidljiv. Pri treniranju s monokromatskom slikom promijenjene su boje na: bijelu za cilj, nebo u crnu i zidovi na tamniju nijansu sive.



Slika 5.13. *Pogled agentove kamere*

5.4. Rezultati treniranja

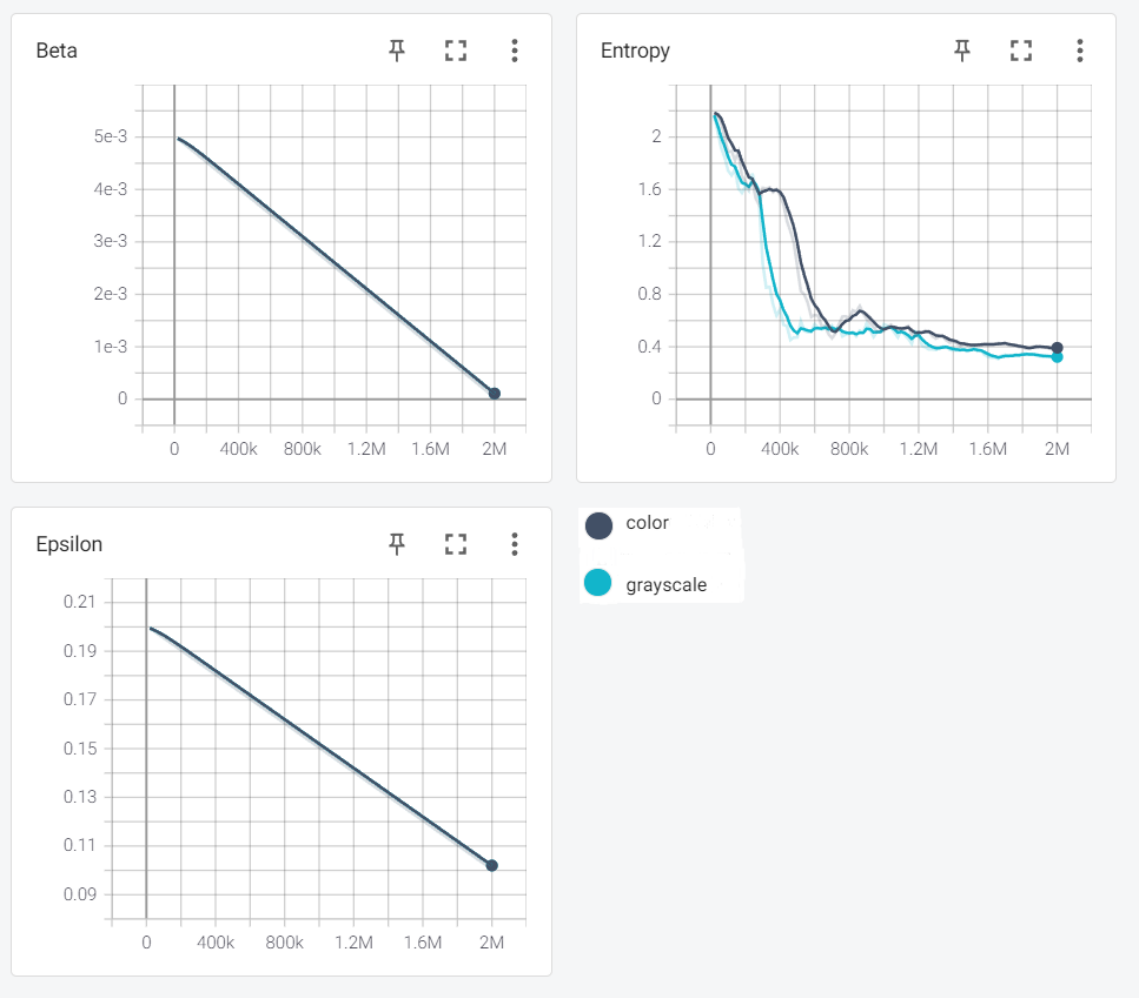
Pri završetku treniranja, rezultati su spremljeni u poseban direktorij unutar projekta. Stvoreni direktoriji sadrže kopiju konfiguracijske datoteke koja se koristila pri treniranju, TensorBoard prikaz rezultata i sam model. Navedeni model je moguće iskoristiti na agentu kako bi otada agent djelovao na osnovi naučenog. Na slici 5.14. narančastom bojom je označeno treniranje s kamerom u boji, a plavom monokromatskom slikom. Zbog veće količine ulaznih podataka pri treniranju sa slikom u boji uočava se sporija stopa učenja nego kod monokromatske slike. Monokromatski model se stabilizirao nakon 450 tisuća koraka dok je drugom modelu trebalo 650 tisuća koraka. Maksimalna kumulativna nagrada koju agent može postići ovisi o udaljenosti agenta od cilja te zato modeli „trepere“ oko vrijednosti 2 i 2.2. Zbog funkcije nagrade agent se brzo naučio kretati u pravilnom smjeru, iako mu nije jasno što mu je cilj, pa završi epizodu dodirom sa zidom ili preprekom. Nakon otkrića pravca kretanja uočava se stagnacija kumulativne nagrade gdje je agent pokupio sve nagrade dodijeljene tijekom približavanja cilju i tada stoji pred ciljem bez da dolazi u kontakt s njim. Nakon što je agent otkrio da mu i meta dodjeljuje nagradu, brzo se prilagođava i



Slika 5.14. *TensorBoard prikaz rezultata treniranja*

model je tehnički istreniran. Zbog nasumičnosti epizode nema problema od prevelikog prilagođavanja modela specifičnoj situaciji.

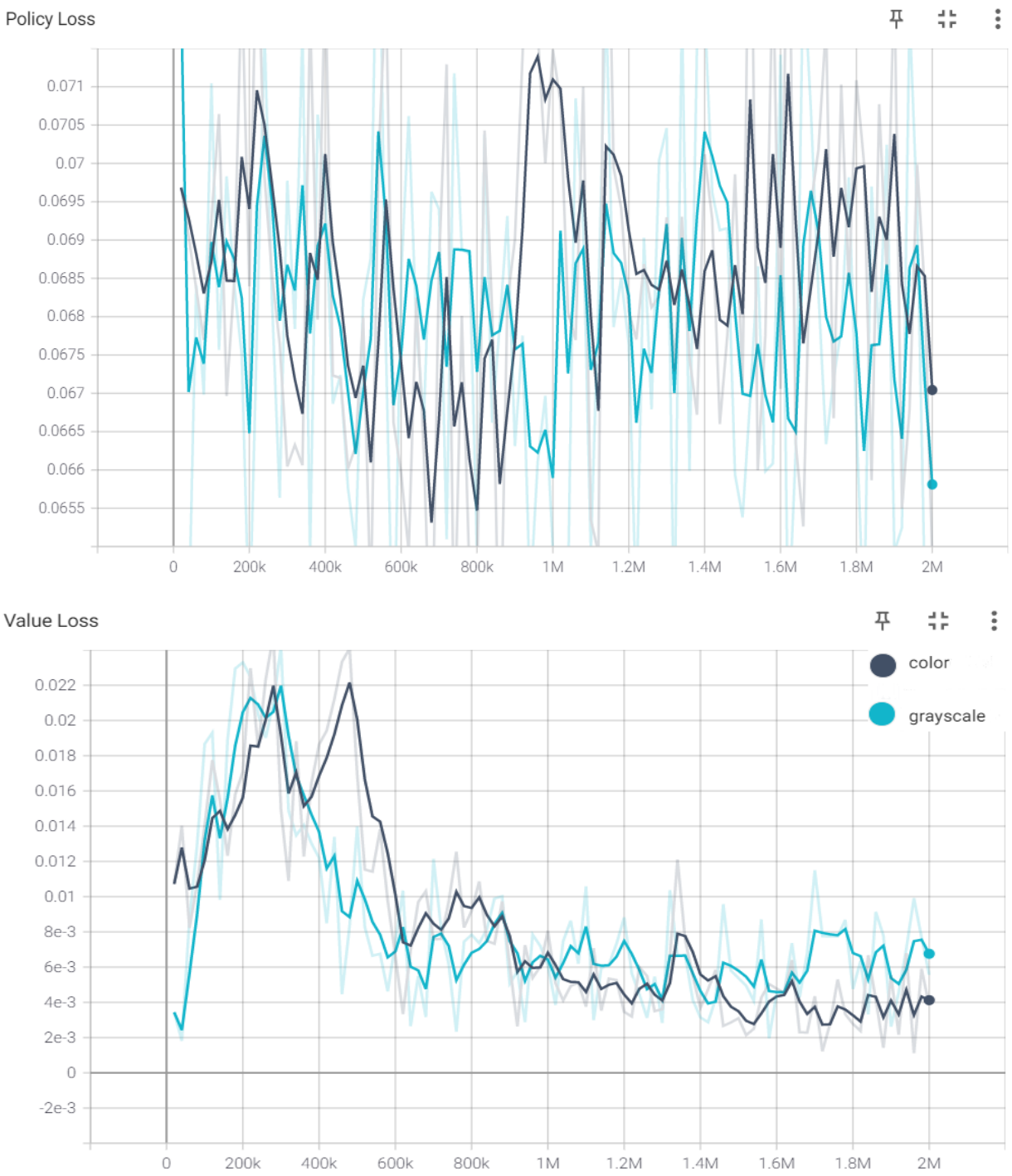
Nakon treniranja agent uspješno detektira cilj i kreće se punom brzinom prema njemu. U slučaju da ne vidi cilj iz startne pozicije, lagano se kreće unaprijed i okreće dok ga ne uoči. Ujedno, agent se naučio umjereno izbjegavati prepreke. U slučaju da se prazan stup nalazi neposredno ispred ciljanog stupa, zbog razlučivosti kamere agent nije u stanju detektirati pogrešan stupac i zabija se u njega. U drugim slučajevima gdje je pogrešan stup na putu, ali ne neposredno ispred cilja, agent uspješno zaobiđe prepreku i ispravi se prema cilju.



Slika 5.15. *TensorBoard prikaz promjene hiperparametara tokom treniranja*

Slika 5.15. prikazuje promjenu prethodno objašnjenih hiperparametara tokom treniranja dok slika 5.16. prikazuje gubitke algoritma. *Policy loss* predstavlja veličinu funkcije gubitaka. Odnosi se na količinu promjene procesa za odlučivanje agentove akcije. Kao što je i prikazano, veličina treba oscilirati tijekom treniranja i obično je reda veličine ispod jedinice. *Value loss* predstavlja gubitak vrijednosti promjene funkcije. Odnosi se na agentovu sposobnost predviđanja vrijednosti svih stanja. Vrijednost navedene treba rasti dok agent uči, te se smanjivati nakon što se nagrada stabilizira. Kao što se vidi sa grafa, vrijednost mu raste do 650 tisuća koraka, ili 450 tisuća koraka u slučaju monokromatske slike, te onda naglo opada pošto je agent u suštini već istreniran za odrađivanje svog zadatka pa mu promjena funkcije nije od daljnje koristi.

Iako se okolina učenja čini prilično oskudnom, rezultat je moguće primijeniti na veći opseg problema zbog prirode prilagodljive okoline. Agent je naučen detektirati i dostići crveni disk. Zbog prethodno navedenih postavki kamere moguće je u kompleksan objekt enkapsulirati skriveni



Slika 5.16. *TensorBoard prikaz promjene gubitaka algoritma*

objekt koji je spomenutog oblika i prikazati ga samo u određenom sloju. Tada je samo potrebno postaviti agentovu kameru da detektira taj sloj i prepreke i agent će uspješno detektirati objekt bez obzira na stvarnu kompleksnost objekta. Istom metodom je moguće i ukloniti ili pojednostaviti sve objekte koji nisu potrebni agentu za snalaženje kroz prostor i time olakšati detekciju prepreka i cilja.

6. ZAKLJUČAK

U ovom radu istražena je primjena dubokog podržanog učenja za treniranje autonomnih agenata u virtualnom okruženju koristeći Unity i ML-Agents. Korištenje virtualnih okruženja olakšava treniranje agenata pošto ne dovodi do stvarne materijalne štete tijekom treniranja i omogućava treniranje u kontroliranoj okolini. Nedostatak takvih je što često agent ne može primijeniti naučene tehnike u stvarnom svijetu zbog drukčijeg ponašanja, uvjeta i izgleda virtualne okoline od stvarnosti. Na primjeru autonomnog kretanja bespilotne letjelice, ako agent pogriješi pri treniranju u stvarnosti, može doći do trajnog oštećenja letjelice i obustave treniranja. Treniranje u stvarnim okruženjima ima prednost u kontroliranim okruženjima, gdje nema mogućnosti štete jer se agent odmah obavlja zadatak dok ga uči, pa nije potrebno prebacivati iskustva iz simulacije u stvarnost. Kako bi se agenta brže navelo na dolazak do cilja, korištena je funkcija nagrade. Ona se može daljnje optimizirati kako bi se ubrzalo treniranje i obuzdalo nepoželjno kretanje agenta. Trenutno agent dobiva nagradu kada se približava cilju što ovisno o varijabilnoj udaljenosti mijenja iznos maksimalne nagrade koju agent može dobiti.

Unatoč većoj kompleksnosti rada s kamerom nego ray-tracing metodom, pruža mogućnosti koje su teže ostvarive s ray-tracing metodom poput lakšeg pronalaženja objekata zbog većeg vidnog polja. Također, implementacijom unutar Unity razvojnog okruženja, može se umanjiti najveći problem računalnog vida u stvarnom svijetu, što je šum u slici. Za ostvarivanje samostalnih inteligentnih agenata, dok obje metode imaju svoje prednosti i mane, idealno bi bilo koristiti obje. Kamerom se može postići opće saznanje o okolini i objektima oko agenta dok se korištenjem ray-tracing metode postiže detekcija u neposrednoj blizini agenta.

Okolinu treniranja je moguće dodatno nadograditi kako bi se poboljšala funkcija agenta dodavanjem cilja koji se kreće, skrivanjem cilja od agenta kako bi ga poticalo na istraživanje okoline ili dodavanjem dodatnih akcija poput skakanja agentu.

LITERATURA

[1] ICLR 2021 Virtualna konferencija, https://iclr.cc/Conferences/2021 ,posjećeno 12.7.2021.

[2] J. Roghair, K. Ko, A.E.N. Alsi, A. Jannesari, „A Vision Based Deep Reinforcement Learning Algorithm for UAV Obstacle Avoidance“, odjel za računarstvo, Iowa sveučilište, IA, SAD, ožu. 2021, https://arxiv.org/abs/2103.06403

[3] G. Surma, „Cartpole – Introduction to Reinforcement Learning“, Medium, ruj. 2018., https://gsurma.medium.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288#:~:text=Cartpole%20%2D%20known%20also%20as%20an,forces%20to%20a%20pivot%20point. ,posjećeno 15.07.2021.

[4] A. Lazaridis, A. Fachantidis, I. Vlahavas, „Deep Reinforcement Learning: A State-of-the-Art Walkthrough“, Journal of Artificial Intelligence Research 69, str. 1421-1471, ruj. 2020., Thessaloniki, Greece

[5] VizDoom github paket, https://github.com/mwydmuch/ViZDoom, ožu. 2017., posjećeno 17.8.2021.

[6] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, D. Lange, „Unity: A General Platform for Intelligent Agents“, San Francisco, SAD, svi. 2020., https://arxiv.org/abs/1809.02627

[7] A. Juliani, V.-P. Berges, E. Teng, A. Crespi, J. Harper, J. Togelius, H. Henry, A. Khalifa, D. Lange, „Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning“, New York, SAD, srp. 2019., https://arxiv.org/abs/1902.01378

[8] Unity Game Engine, https://unity.com/ , posjećeno 12.7.2021.

[9] Unity lista paketa, https://docs.unity3d.com/Manual/PackagesList.html, posjećeno 12.7.2021.

[10] ML-Agents paket, https://github.com/Unity-Technologies/ml-agents, posjećeno 12.7.2021.

[11] Pregled ML-Agents Unity paketa, https://github.com/Unity-Technologies/ml-agents/blob/main/docs/ML-Agents-Overview.md, posjećeno 12.7.2021.

[12] Python aplikacijsko programsko sučelje ML-Agents paketa, https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Python-API.md, posjećeno 12.7.2021

[13] C# programski jezik, https://docs.microsoft.com/en-us/dotnet/csharp/, posjećeno 12.7.2021.

[14] PyTorch, https://pytorch.org/, posjećeno 12.7.2021.

[15] TensorFlow, https://www.tensorflow.org/guide, posjećeno 12.7.2021.

[16] R.R. Torrado, P. Bontrager, J. Togelius, J. Liu, D. Perez-Liebana, „Deep Reinforcement Learning for General Video Game AI“, 2018 IEEE Conference on Computational Intelligence and Games, doi:10.1109/cig.2018.8490422

[17] U. Tewari, „Which Reinforcement learning- RL algorithm to use where, when and in what scenario?“, Medium, tra. 2020., https://medium.datadriveninvestor.com/which-reinforcement-learning-rl-algorithm-to-use-where-when-and-in-what-scenario-e3e7617fb0b1, posjećeno 17.7.2021.

[18] C. Trivedi, „Proximal Policy Optimization Tutorial (Part 1/2: Actor-Critic Method)“, Towards dana science, kol. 2019., https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6, posjećeno 17.8.2021.

[19] ML-Agents Training Configuration File, https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md, posjećeno na 18.8.2021.

SAŽETAK

U radu se istražuje primjena i korištenje virtualnih okolina za treniranje agenata koristeći računalni vid. Cilj je stvoriti okolinu u kojoj će se agent kretati vizualnim opažanjima u svrhu obavljanja zadatka. Za stvaranje virtualne okoline korišten je Unity program za razvoj i upravljanje video igrama zajedno s ML-Agents paketom kao osnovom za treniranje agenta. Stvorena je okolina s tri stupca od kojih će jedan biti nasumično odabran kao cilj s metom na sebi dok su druga dva mamac agentu. Stupci su također generirani nasumične visine i pozicije. Agent je uspješno istreniran za prepoznavanje cilja i navigiranje prema njemu, izbjegavajući prepreke na putu.

Ključne riječi: duboko učenje, podržano učenje, računalni vid, Unity, virtualna okolina

ABSTRACT

AI Vision using machine learning in Unity game engine

This work researches and applies usage of virtual environments in purpose of training agents using visual observations. It's purpose is to create an enviromnent that the agent will traverse using visual observations to complete his task. Unity game engine was used to create the environment alongside ML-Agents package that handles training the agent. The learning environment contained three pillars out of which one was randomly chosen to be the agent's goal with a visual target on top of it, whilst the other two were decoys. The pillars were also randomized in height and position. The agent was successfully trained to spot his target as well as to navigate towards it while avoiding obstacles.

Keywords: deep learning, reinforcement learning, AI vision, Unity, virtual environment