Napredni mehanizmi percepcije u igri traženja na temelju dubokog učenja

Hodak, David

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:399216

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

NAPREDNI MEHANIZMI PERCEPCIJE U IGRI TRAŽENJA NA TEMELJU DUBOKOG UČENJA

Završni rad

David Hodak

Osijek, 2020.

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

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 09.09.2020.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	David Hodak
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3922, 23.09.2019.
OIB studenta:	86918475501
Mentor:	Izv. prof. dr. sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Napredni mehanizmi percepcije u igri traženja na temelju dubokog učenja
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Predložena ocjena završnog 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:	09.09.2020.
Datum potvrde ocjene Odbora:	23.09.2020.

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, 24.09.2020.

Ime i prezime studenta:	David Hodak
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3922, 23.09.2019.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Napredni mehanizmi percepcije u igri traženja na temelju dubokog učenja**

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 *1*

1.1. Zadatak završnog rada **1**

2. PREGLED PODRUČJA RADA *2*

3. KORIŠTENE TEHNOLOGIJE I ALATI *3*

3.1. Unity **3**

3.2. C# programski jezik **4**

3.3. Unity Machine Learning Agents **5**

3.3.1. Metode treniranja 7

3.4. Tensorflow **9**

4. DUBOKO UČENJE *10*

4.1. Duboko učenje s potporom **11**

5. IZRADA PROJEKTA *13*

5.1. Metode percepcije agenta **13**

5.2. Izrada okoline **14**

5.3. Implementacija agenta **16**

5.4. Treniranje agenta **22**

5.5. Rezultati treniranja **23**

5.6. Problemi u implementaciji i mogućnosti poboljšanja **26**

6. ZAKLJUČAK *28*

LITERATURA *29*

SAŽETAK *30*

ABSTRACT *30*

ŽIVOTOPIS *31*

1. UVOD

Strojno učenje je grana umjetne inteligencije gdje se uz korištenje posebno strukturiranih algoritama, računala "uče" izvršavati određene zadatke s dostupnim podacima. Upravo su ti podaci, te njihova sve veća važnost i količina u današnjem svijetu potaknuli razvitak dubokog učenja, grane strojnog učenja koja nastoji što bolje iskoristiti te podatke u rješavanju izazovnih problema umjetne inteligencije.

Testiranje situacija u kojima bi specijalizirani objekti koristili algoritme strojnog i dubokog učenja te izvršavali kompleksne zadatke može biti financijski i vremenski intenzivno. Za velike tvrtke kao što su Google ili Amazon to možda nije problem, ali za manja poduzeća, istraživače umjetne inteligencije ili hobiste korištenje programskog okruženja gdje se kreiraju virtualne okoline koje omogućuju testiranja je efikasnija i ekonomičnija varijanta.

Upravo će se u ovom završnom radu istražiti na koji način agenti kreirani u virtualnoj simulaciji percipiraju objekte oko sebe koristeći mehanizme percepcije i algoritme dubokog učenja kako bi došli do zadanog cilja. Alat za strojno učenje koji će to omogućiti dio je Unity *game enginea*. Unity *game engine* omogućava kreiranje virtualnih simulacija s realnom fizikom. Uz ponešto znanja moguće je kreirati zanimljive okoline u kojima se treniraju agenti. Unity Machine Learning Agents je SDK (engl. *Software Development Kit*), te će se koristiti za izradu projektnog zadatka. Zadatak će se sastojati od scenarija u kojem će agent tražiti objekt koji se nakon svakog pronalaska postavlja na novo, nasumično generirano mjesto. Koristiti će se učenje s potporom.

U nastavku rada opisane su korištene tehnologije s naglaskom na Unity Machine Learning Agents. Nakon toga objašnjavaju se osnove dubokog učenja. U sljedećem poglavlju opisana je izrada projekta, a na kraju rezultati, problemi i komentari rada.

1.1. Zadatak završnog rada

U radu je potrebno kreirati Unity okolinu u kojoj će se trenirati agent koristeći mogućnosti Unity Machine Learning Agentsa. Opisati mehanizme percepcije agenata u Unity okolini te dati osnovnu pozadinu dubokog učenja.

2. PREGLED PODRUČJA RADA

Napredak u istraživanjima umjetne inteligencije ovisi o pronalaženju problema u postojećim okolinama i uspješnom rješavanju istih. Međutim, uvijek postoji potreba za novim okolinama čije kreiranje često zahtijeva mnogo vremena, novaca i specijaliziranog znanja. Koristeći Unity i ML-Agents alat, mogu se stvoriti željene okoline te ih koristiti za istraživanja novih algoritama i metoda. Prva verzija Unity Machine Learning Agents alata objavljena je 17. rujna 2017. „Zadatak je bio jednostavan – omogućiti game developerima i istraživačima umjetne inteligencije da koriste Unity kao platformu za treniranje inteligentnih agenata koristeći najnovija dostignuća u strojnom učenju.“[1]

Jedan takav primjer je Carry Castle, mali studio za izradu igara iz Švedske. U njihovoj još ne objavljenoj igri, Source of Madness, glavni lik kojim upravlja igrač prolazi kroz dinamični svijet i bori se sa proceduralno generiranim neprijateljima koji su kreirani korištenjem strojnog učenja. Na taj način, igrač se može susresti sa mnoštvom varijacija neprijatelja. Upotrijebili su ML-Agents alat i metode dubokog učenja kako bi kreirali model neuronske mreže koja bi im to omogućila i ugradili ju u Unity okruženje.[2]

Unity tim programera je u partnerstvu s JamCity, tvrtkom iz San Francisca koja izrađuje mobilne igre, pokušao trenirati agenta pomoću alata ML-Agents da nauči igrati njihovu igru Snoopy Pop. Poanta igre je prelaziti mnoštvo razina gdje se na svakoj razini nalazi drugačija kombinacija balona u različitim bojama koji se moraju uništiti koristeći balone koje kontrolira igrač. S obzirom da postoji više boja i igrač može kontrolirati samo jedan od balona u trenutku, čija boja je slučajno generirana, takvo okruženje je predstavljalo pravi izazov za agenta s obzirom na količinu informacija koje mora analizirati.[3]

Iako nisu koristili ML-Agents alat, tvrtka OpenAI je uz pomoć vlastitog algoritma za duboko učenje kreirala okruženje gdje su agenti igrali igru skrivača te promatrala njihova ponašanja i načine na koji su se oni pokušavali sakriti, odnosno pronaći.[4]

Alat je dostupan svima pa tako i studentima koji su zainteresirani za strojno učenje. Na ovaj način im je omogućeno da implementiraju i testiraju svoje ideje. Tako je kolega Mateo Bareš u svome diplomskom radu opisao strojno učenje kroz projektni zadatak u kojem je trenirao agenta da izbjegava prepreke krećući se po nasumično generiranoj cesti.[5]

3. KORIŠTENE TEHNOLOGIJE I ALATI

3.1. Unity

Unity je više-platformski *game engine* kreiran od tvrtke Unity Technologies. Objavljen 2005. godine, prvotno ga se moglo koristiti samo na Mac operacijskom sustavu, a kasnije je dodana podrška za Windows sustav. Cilj mu je bio razvoj igara učiniti pristupačnijim široj zajednici, čime je privukao mnoštvo novih developera. Korišten je za kreiranje 2D i 3D video igara, računalnih simulacija, virtualnih i proširenih stvarnosti te stvaranje prigodnih okolina za strojno učenje. Osim industrije igara, njegova korisnost pronašla se i industrijama kao što su filmska, inženjerska, automobilska i arhitektonska. Iako je pisan u C++ programskom jeziku, za korištenje raznih mogućnosti i pisanje skripti koje daju funkcionalnost objektima upotrebljava se C# programski jezik.

Kada je u pitanju grafika URP (engl. *Universal Render Pipeline*) i HDRP (engl. *High Definition Render Pipeline*) nude mnoštvo mogućnosti. URP je lakša verzija koja kombinira perfomanse i optimalan izgled, bez izmjena u vlastitom programskom kodu. HDRP omogućava kreiranje vizualno impresivne računalne grafike. Kako bi ostvario što realističniju fiziku u svojim simulacijama, Unity koristi ugradnje Box2D i NVIDIA PhysX. [6]

Unity uređivač je grafičko sučelje programa u kojem se kreira sadržaj. Korisniku se na raspolaganje stavlja mnoštvo alata za kreiranje i manipulaciju objekata. Jednostavniji oblici objekata se mogu kreirati u samom uređivaču dok se oni sa više detalja kreiraju u programima za modeliranje. U glavnom prozoru za izradu (engl. *scene view*) možemo postavljati objekte, okretati ih, smanjivati ili povećavati, dodavati različita osvjetljenja itd., sve kako bi kreirali jednu razinu u igri, cijeli igru, simulaciju ili pak samo interaktivni tekst.

Da bi se objektima dodijelile kompleksnije funkcionalnosti, primjerice, pomicanje u određenom smjeru, koristimo C# skripte. Uređivač podržava različite video, tekstualne, slikovne i audio formate. Unity se pokazao idealnim za ovaj rad jer podržava dodatak koji koristi strojno učenje za treniranje agenata te je besplatan ukoliko se ne ostvaruje određena financijska dobit.

3.2. C# programski jezik

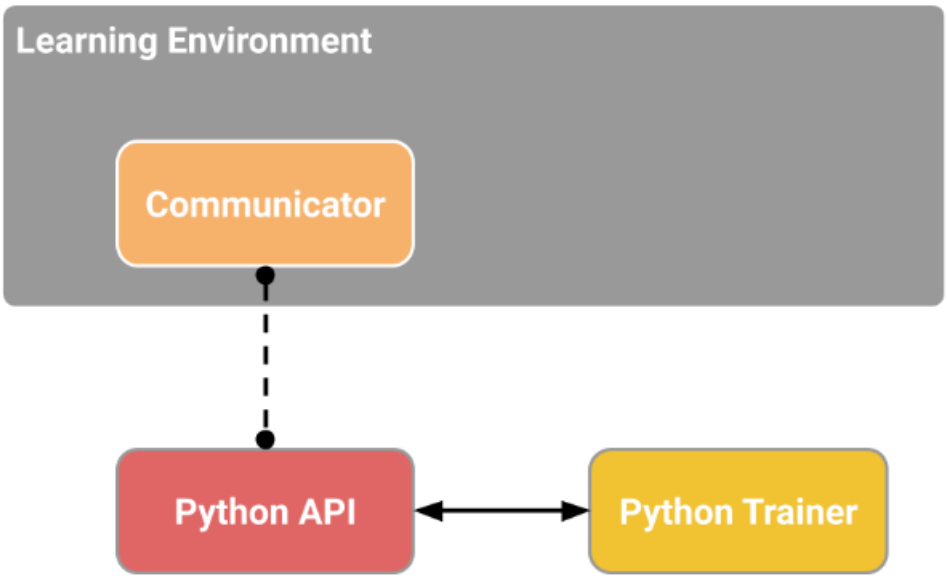
„C# je *type-safe* objektno-orijentirani programski jezik opće namjene. Cilj mu je poboljšati produktivnost programera. Uravnotežuje jednostavnost, izražajnost i performanse. Glavni arhitekt ovog programskog jezika od njegove prve verzije je Anders Hejlsberg.“ [7] Sintaksom je sličan programskim jezicima C, C++ i Java što omogućuje vrlo brzu prilagodbu programerima koji su upoznati s njima na C#. Pojednostavljuje kompleksnost koju ima C++ te pruža značajke kao što su *nullable* vrijednosni tipovi, enumeracije, delegate, lambda ekspresije i direktan memorijski pristup. Podržava i generičke metode i tipove čime utječe na sigurnost i performanse aplikacija. Oslanja se primarno na objektno-orijentiranu paradigmu, što znači da uključuje svojstva enkapsulacije, nasljeđivanja i polimorfizma. Enkapsulacija znači stvaranje granice oko objekta kako bi odvojili njegova javna ponašanja od privatnih implementacijskih detalja. Sve varijable i metode su enkapsulirane u definiciju klase. Klasa može naslijediti samo jednu roditeljsku klasu, ali može implementirati željeni broj sučelja. C# ima sustav jedinstvenog tipa gdje svaki tip varijable dijeli osnovnu funkcionalnost, primjerice, svaki tip se može pretvoriti u *string* tip koristeći metodu *ToString*. Koristi i značajke funkcijskog programiranja, specifično, funkcije (metode) se mogu tretirati kao varijable. Proces izvršenja programa u C# je jednostavniji nego u C i C++, te fleksibilniji nego u Javi. Nema dodatnih datoteka i zahtjeva da metode i atributi moraju biti deklarirani posebnim redoslijedom. C# izvršna datoteka može sadržavati bilo koji broj klasa, struktura, sučelja i događaja.[7]

Programi napisani u C# izvode se na .NET *Frameworku*, ugrađenoj komponenti Windows operacijskog sustava koja uključuje virtualni sustav izvršenja nazvan CLR (engl. *common language runtime*) i jedinstveni skup biblioteka klasa. Biblioteka se sastoji od preko 4000 klasa organiziranih u imenike koji omogućuju korištenje veoma korisnih funkcija u upravljanju tijeka programa. Primjerice, kontroliranje unosa i izlaza iz datoteke ili izrada aplikacije uz pomoć Windows Forms sustava.[8]

3.3. Unity Machine Learning Agents

ML-Agents (engl. *Machine Learning Agents*) je *open-source* projekt koji igrama i simulacijama omogućuje da služe kao okruženje za treniranje inteligentnih agenata. Agenti se mogu trenirati korištenjem učenja s potporom, imitacijskim učenjem, evolucijom ili drugim metodama strojnog učenja kroz jednostavno sučelje Phython API-ja. [9]

ML-Agents zapravo dolazio kao SDK (engl. *software development kit*) koji je kompatibilan s Unity *engineom* i instalira se kao paket. Može se dohvatiti s službenih stranica ili izravno instalirati iz Unity uređivača. Sadrži 15 primjera okolina za učenje, s idejama koje su raznolike i primjenjive u vlastitim okolinama, stoga je njihovo proučavanje i igranje s parametrima preporučljivo. Za treniranje s potporom koristi dva algoritma, PPO (engl. *Proximal Policy Optimization*) i SAC (engl. *Soft Actor Critic*). Ugrađena je i podrška za učenje kroz imitaciju. Mogućnost trening mehanizma za samo-igranje u suparničkim scenarijima. Više instanci okolina u Unity-u se može istovremeno trenirati što drastično ubrzava vrijeme potrebno za treniranje. Kontroliranje treninga kroz Python sučelje. ML-Agents se sastoji od četiri glavne komponente, prikazane na slici 3.1:



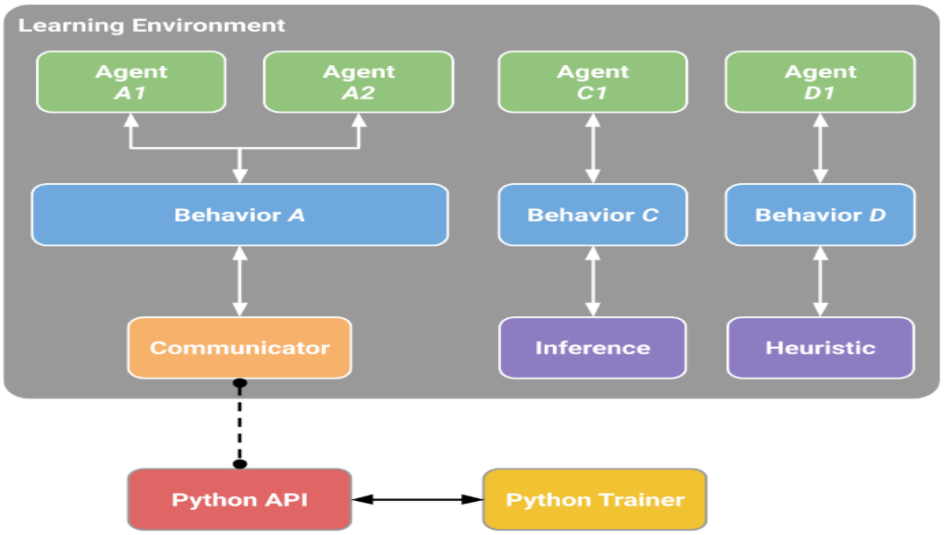
Sl. 3.1. Pojednostavljeni blok dijagram komponenti ML-Agentsa[10]

- **Okolina učenja –** sastoji se od zamišljene scene u Unityu i svih likova (agenata). Scena predstavlja okolinu u kojoj agenti promatraju, djeluju i uče. Izgradnja okoline bi trebala biti postepena kako bi što efikasnije riješili problem ili shvatili da je problem suviše kompleksan da bi ga agent svladao te smanjili kompleksnost.

- **Python API –** nije dio Unitya već se povezuje kroz komunikator. Sadrži jednostavno sučelje koje se koristi za interakciju i rukovanje treningom.
- **Vanjski komunikator -** povezuje Python API s okolinom učenja.
- **Python treneri –** sadrže sve algoritme koji koriste strojno učenje pri treniranju agenata. Algoritmi su implementirani u Pythonu.

Okolina učenja sastoji se od dvije veoma bitne komponente koje sudjeluju u organizaciji okoline:

- **Agenti -** svakom liku u sceni koji je predstavljen Unity *game* objektom može se dodati svojstvo agenta. Primjerice, agent može biti dodan na najjednostavniji objekt kao što je kocka. Izvršava akcije, rukovodi generiranje vlastitih zapažanja te dodjelu nagrade kada je potrebno. Svaki agent je povezan s ponašanjem.
- **Ponašanje –** definira specifične atribute agenta kao što je broj akcija koje smije poduzeti. Ponašanje se može prikazati kao funkcija koja prima zapažanja i nagrade od agenta i vraća akcije koje će agent poduzeti. Postoji tri tipa ponašanja: učenjem, heuristički i zaključivanjem. Ponašanje učenjem je ono koje nije definirano za sada ali je spremno za treniranje. Heurističko ponašanje je definirano pravilima implementiranim u kod. Ponašanje zaključivanjem podrazumijeva datoteku s istreniranom neuronskom mrežom.



Sl. 3.2. Blok dijagram kao primjer više ponašanja i agenata[10]

Kako je prikazano na slici 3.2. svaka okolina učenja može imati više agenata, odnosno onoliko agenata koliko ima likova u sceni. S obzirom da svaki agent mora biti povezan s ponašanjem, ukoliko agenti dijele slična promatranja i akcije, moguće je da će imati isto ponašanje.

Ponašanjem se definira prostor mogućih opažanja i akcija. Iako dijele isto ponašanje agenti mogu imati zasebna opažanja i akcije. Naravno, moguće je imati i više ponašanja s vlastitim povezanim agentima.

3.3.1. Metode treniranja

Da bi se objasnilo treniranje u simulaciji potrebno je definirati tri elementa koja su uvijek prisutna:

- **Opažanja –** što se percipira iz okoline. Opažanja mogu biti numerička i/ili vizualna. Numerička opažanja označuju atribute okoline s gledišta agenta. Vizualna opažanja su slike generirane od kamere koja je postavljena na agenta i predstavljaju ono što agent vidi u određenom vremenu. Ovisno o složenosti simulacije, opažanja mogu biti diskretna ili kontinuirana.
- **Akcije –** koje akcije agent može poduzeti. Ovisno o složenosti simulacije, akcije također mogu biti diskretne ili kontinuirane. Ukoliko želimo da se agent kreće u jedan od četiri smjera, primijenit ćemo diskretne akcije, što znači da će agent primati samo jednu vrijednost ako se kreće, primjerice, ravno. Ako želimo da se agent kreće slobodnije i ako nam je okolina kompleksnija, koristit ćemo kontinuirane akcije.
- **Signali nagrade –** vrijednost koja pokazuje napredak agenta. Nije potreban u svakom trenutku, nego kada agent učini neku akciju koja može biti dobra ili loša, što znači da će kao signal nagrade biti dodijeljena pozitivna ili negativna vrijednost. Uvijek je bolje voditi se manjim pozitivnijim nagradama nego velikim negativnim. Primjerice, bolje je agenta nagraditi malom pozitivnom vrijednošću za svaki učinjeni korak nego negativnom.

Učenje s potporom

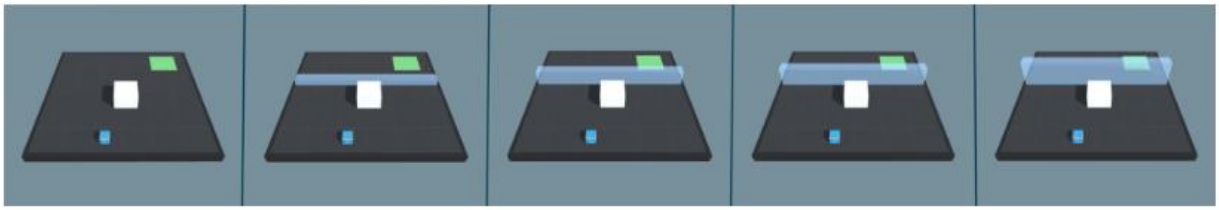
Sastoji se od dva algoritama PPO i SAC. Zadani algoritam je PPO, koji će se koristiti u ovom radu. Pokazao se stabilnijim od mnogih drugih algoritama učenja s potporom. S obzirom da su takvi algoritmi poprilično nezgodni za *debugging* i zahtijevaju dosta namještanja ako se žele dobri rezultati, PPO je inovacija jer je omogućio ravnotežu između jednostavnosti implementiranja, kompleksnosti uzorka i lakšeg namještanja parametara. Koristi arhitekturu dubokih neuronskih mreža. Koristeći SAC algoritam, agent može učiti iz iskustava prikupljenih u prošlim prolascima. Iskustva se spremaju u spremnik (engl. *buffer*) i izvlače nasumično iz njega, što SAC čini učinkovitijim pri skupljanju uzoraka. Dobar je izbor za teže i sporije okoline.

Učenje imitacijom

Uz pomoć demonstracije ponašanja za kojeg želimo da agent nauči možemo drastično ubrzati proces treniranja. Demonstracije se izvode s igračeve strane, uz pomoć zadanih kontrola te će sve aktivnosti, nagrade i zapažanja biti zabilježena. Poželjno je demonstraciju provoditi što je duže moguće. Algoritam učenja imitacijom će zatim koristiti zabilježene aktivnosti i pokušati ih naučiti. Može ga se koristiti kao samostalni tip učenja ili s učenjem s potporom. Samostalno se koristi ako se želi trenirati specifični tip ponašanja.

Učenje s kurikulumom

Nastoji se simulirati način na koji ljudi uče. Prvo se postavljaju najjednostavniji problemi koji se zatim postepeno otežavaju. Ovaj način treniranja koristi se kod složenijih problema gdje se problem postupno uvodi na takav način da je model uvijek optimalno izazvan. Na slici 3.3. je prikazan jednostavni primjer učenja s kurikulumom gdje agent pokušava doći do cilja. Kako nauči zadano, prepreka (zid) između njega i cilja se poveća. Jasno je da je ovdje zadani problem bio naučiti agenta da dođe do cilja tako što će preskočiti zid zadane visine. S obzirom da bi ovdje učenje s potporom bilo vrlo teško za izvesti, pogotovo bez demonstracija, učenje s kurikulumom se pokazalo idealnim za ovakve okoline gdje je agentu pruženo više aktivnosti i veći broj zapažanja.



Sl. 3.3. Demonstracija učenja s kurikulumom [10]

3.4. Tensorflow

TensorFlow je *open-source* platforma za strojno učenje. Ima opsežan ekosustav alata, biblioteka i resursa od zajednice korisnika kojim omogućava istraživačima da razvijaju svoje ideje u strojnom učenju. ML-Agents koristi biblioteku TensorFlowa u svojim implementacijama. Omogućava numeričko računanje koristeći grafikone protoka podataka, kao temeljni prikaz modela dubokog učenja. Može se izvoditi na procesorima i grafičkim procesorima. Nakon završetka treniranja kao izlaz dobijemo TensorFlow datoteku modela sa ekstenzijom (.nn) koja se onda može povezati s agentom.

TensorBoard je vizualizacijski alat unutar TensorFlowa koji omogućuje prikaz određenih atributa kroz trening agenta. S obzirom da treniranje modela s TensorFlowovom zahtjeva postavljanje određenih parametara (engl. *hyperparameters*), postavljanje istih zahtjeva nekakvu referencu, koju dobijemo sa ovim vizualizacijskim alatom. Primjerice, cjelokupna nagrada bi trebala rasti tokom uspješnog treninga. Ukoliko to nije slučaj, moramo podesiti parametre treniranja.[11]

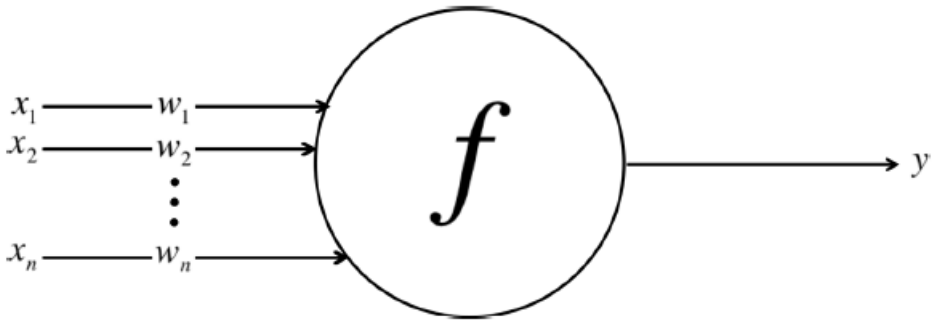
Sl. 3.4. Sučelje TensorBoarda [12]

4. DUBOKO UČENJE

„Duboko učenje je grana strojnog učenja temeljena na predstavljanju podataka složenim reprezentacijama na visokom stupnju apstrakcije do kojih se dolazi slijedom naučenih nelinearnih transformacija.“ [13] Jednostavnije rečeno, duboko učenje je tehnika strojnog učenja koja omogućava računalima da uče iz primjera. Ova tehnologija danas omogućava napredak u mnogim aspektima života. Tako se koristi u autonomnim automobilima, pametnim satovima, mobitelima, televizorima, kamerama itd. Razlog zbog kojeg je duboko učenje nedavno dobilo na značenju je što računalni modeli zahtijevaju velike količine označenih podataka. Podaci se klasificiraju direktno iz slika, teksta ili zvuka. Također se zahtijevaju znatne računalne snage, kao primjerice, grafičke kartice visokih performansi za koje se pokazalo da značajno smanjuju vrijeme treniranja modela. [14]

Većina metoda dubokog učenja koristi arhitekturu neuronskih mreža, zbog čega se na modele dubokog učenja često referira kao na duboke neuronske mreže. Razlog naziva „duboko“ je taj što se modeli sastoje od više skrivenih slojeva između ulaza i izlaza. Modeli se treniraju s označenim podacima i neuronskom mrežom koja uči značajke direktno iz podataka bez potrebe za bilo kakvim izdvajanjem.

Neuronske mreže se sastoje od neurona (slika 4.1.). Isto kao i biološki neuroni, umjetni neuroni primaju određeni broj podataka $(x_1, x_2, \dots x_n)$ koji se množe sa specifičnom težinom$(w_1, w_2 \dots w_n)$. Zatim se takvi podaci zbrajaju kako bi proizveli *logit*[1] $(z = \sum_{i=0}^{n} w_i * x_i)$ neurona, koji se predaje funkciji f čiji je produkt izlaz y = f(z)

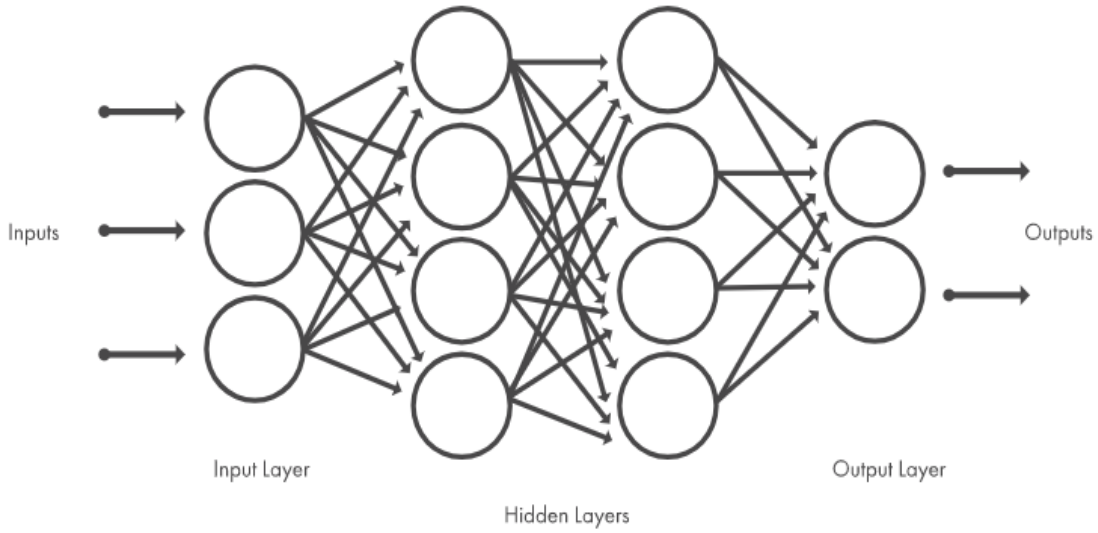


Sl. 4.1. Shema neurona u neuronskom mreži [13]

[1] Funkcija poznata i kao *log-odds* funkcija koja predstavlja vjerojatnostne vrijednosti [0,1] i [-∞, ∞]

Svaka neuronska mreža (slika 4.2.) sastoji se od jednoj ulaznog sloja, kroz koju prolaze ulazni podaci, izvršavaju se proračuni kroz neurone i izlazni podaci se prosljeđuju na sljedeće slojeve.

Izlazni sloj je zadužen za gotovi rezultat. Prima podatke iz slojeva prije njega i izvršava proračune kroz svoje neurone. Skriveni slojevi su vidljivi samo neuronskoj mreži te se nalaze između ulaznog i izlaznog sloja. Moguće je imati nijedan skriveni sloj. Što ih je više, neuronska mreža može riješiti kompleksnije probleme. Duboka neuronska mreža za stvarni problem može imati više od 10



Sl. 4.2. Model neuronske mreže [14]

skrivenih slojeva. Jedna od poznatijih dubokih neuronskih mreža je konvolucijska mreža. Koristi 2D konvolucijske slojeve, što joj arhitekturu čini vrlo prikladnom za obradu 2D podataka pa se koristi za strojni vid.

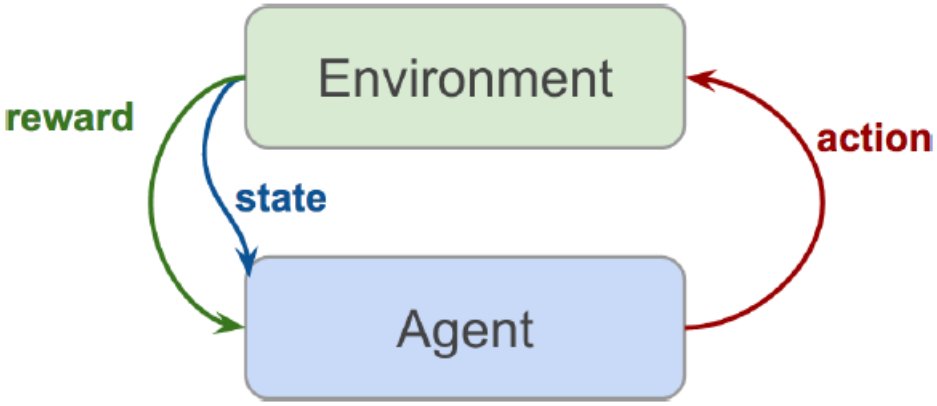
4.1. Duboko učenje s potporom

U nadziranom učenju daju se poznati ulazni i izlazni podaci te se onda vrše predviđanja izlaznih podataka s obzirom na prošle rezultate. U nenadziranom učenju daju se samo ulazni podaci i pokušavaju se pronaći skriveni uzorci i osnove strukture. Kod učenja s potporom nema nikakvih ulaznih ni izlaznih podataka, samo učenje ovisi o nagradama koje agent dobiva.

Ovakvo učenje zahtjeva interakciju s okolinom. Sastoji se od agenta, okoline i signala nagrade. Cilj je proizvesti ponašanje koje će rezultirati najvećom nagradom, odnosno naučiti pravila.

Agent promatra okolinu i odabire akcije koje će poduzeti te prema tome dobiva odgovarajuću nagradu. Nagrada je pokazatelj koliko dobro agent obavlja zadatak, a s obzirom da agent u početku ne zna koji mu je cilj, važno je da su nagrade rijetke. Rijetke nagrade u jednostavnijim okolinama

će najčešće rezultirati uspješnom učenju, dok se kod kompleksnih okolina očekuje dugotrajnije i teže učenje. [13]



Sl. 4.3. Odnosi u učenju s potporom [13]

5. IZRADA PROJEKTA

Projekt će se sastojati od okoline u Unity programskom okruženju kojom će se testirati mogućnosti ML-Agents alata i agenta s pripadajućim elementima koji će naučiti kako da pronađe skriveni predmet.

5.1. Metode percepcije agenta

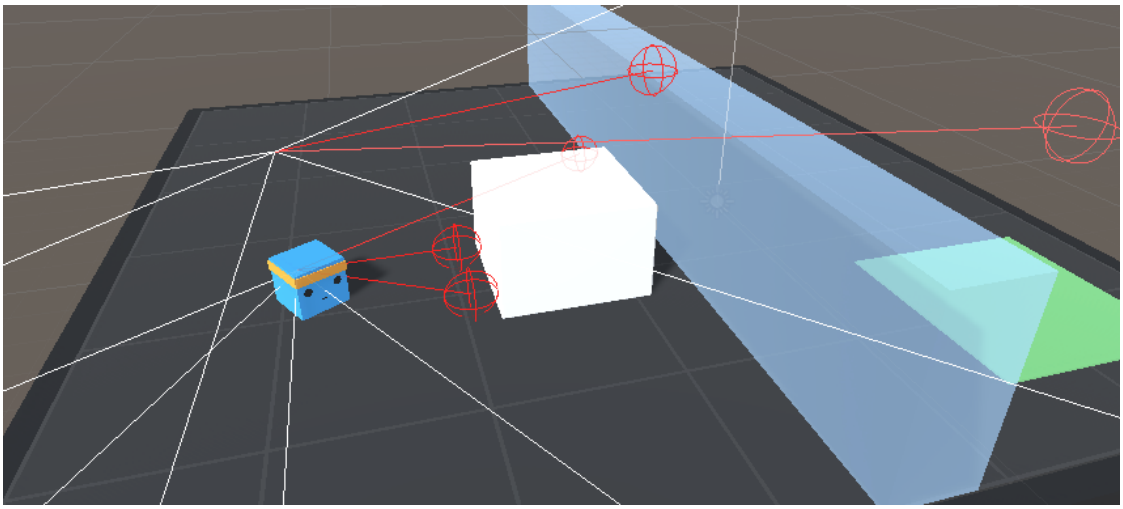
Vizualna opažanja

Jedan od načina na koji agentu možemo omogućiti da percipira okolinu je koristeći *CameraSensor* ili *RenderTextureSensor* komponente koje su dio ML-Agentsa. Informacije o slikama se zatim transformiraju u prihvatljive podatke za konvolucijsku neuronsku *mrežu*, što omogućuje agentu da uči iz pravilnosti na promatranim slikama. Za prikupljanje vizualnih informacija, agentovom *game* objektu dodamo *CameraSensor* ili *RenderTextureSensor* komponentu. Zatim se komponentama dodaju željene kamere ili *render textures*, unose se parametri za visinu i širinu slike te je li slika u boji ili u sivim tonovima. Veličina slike bi trebala biti što manja, do granice gdje se bitni detalji za odlučivanje agenta vide. Također slike bi trebale biti u sivim tonovima, osim ako je boja bitna za odlučivanje. Koristi se u kompleksnijim okolinama gdje se određena stanja ne mogu opisati brojčano. No, često su spora i manje efikasna od vektorskih ili *raycast* opažanja.

Raycast opažanja

Metoda opažanja gdje se agentovom objektu dodaje RayPerceptionSensorComponent3D komponenta, čime se nekoliko zraka pruža u okolinu i dodiruje objekte. Ova komponenta ima nekoliko postavki no najbitnije su:

- **Oznake koje se mogu prepoznati** (engl. *Detectable Tags*) **–** niz oznaka koje odgovaraju vrstama objekata koje bi agent mogao razlikovati
- **Broj usmjerenih zraka** *(engl. Rays Per Direction)* **–** jedna zraka se uvijek pruža ravno i referentna je. Prema ovom broju isti broj zraka se postavlja lijevo i desno od referentne.
- **Maksimalni kut između zraka** (engl. *Max Ray Degrees*) **–** vrijednost koja predstavlja broj stupnjeva raspodijeljenih između zraka.
- **Dužina zrake** (engl. *Ray Length*) **–** željena dužina zraka u prostor.



Sl. 5.1. Zrake u prostoru

Broj zraka i oznaka bi trebao biti reduciran na one najbitnije za odlučivanje kako bi se smanjio broj podataka koje algoritam obrađuje i ubrzao trening. Ova metoda se pokazala najboljom kod okolina gdje se ne zahtjeva cijela slika za uspješno odlučivanje agenta te se može puno jednostavnije postići stabilno i uspješno ponašanje. Iz tog razloga se koristi u ovom projektu.

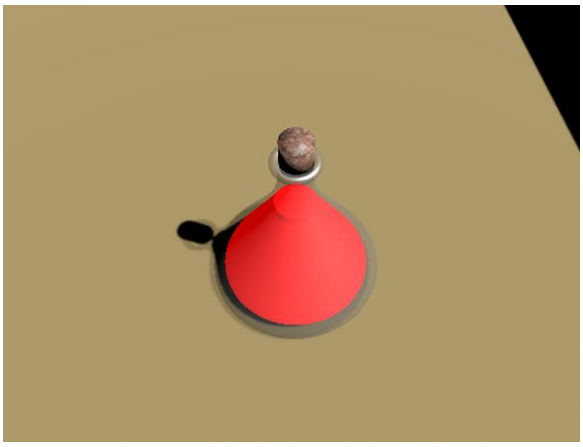
5.2. Izrada okoline

Na slici 5.4. prikazana je scena simulacije unutar Unity uređivača. Okolina je zamišljena kao poligon s osam strana u kojoj se nalazi agent i ostali objekti. Zid i podloga su napravljeni uz pomoć Probuilder alata koji omogućava osnovno modeliranje objekata u Unity uređivaču. Ostali objekti su kupljeni dodaci sa Unity službene stranice. Lik mumije predstavlja agenta (slika 5.2.).

Sl. 5.2. Objekt koji predstavlja agenta

Sl. 5.4. Izgled okoline u Unityu

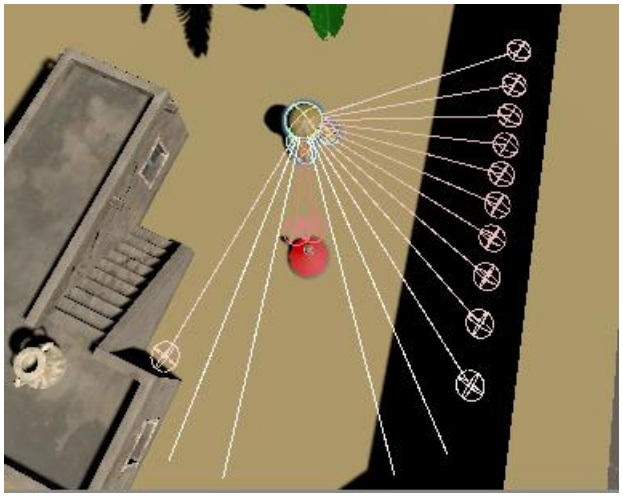
Agent mora pronaći objekt koji je predstavljen crvenim napitkom. Kako bi mu se otežao posao, nakon svakog pronalaska, napitak se postavlja na novo, nasumično generirano mjesto.



Sl. 5.3. Cilj agenta

5.3. Implementacija agenta

Kako bi željeni objekt učinili agentom moramo kreirati novu C# skriptu i postaviti je na *game* objekt. Skripta sadržava klasu željenog imena. Klasa mora naslijediti već predefiniranu klasu *Agent* koja je dio ML-Agents paketa. Postoji nekoliko metoda koje se mogu prepisati no najbitnije su *CollectObservations* i *OnActionReceived*. *CollectObservations* metoda služi za prikupljanje opažanja. *OnActionReceived* služi za definiranje akcija koje će agent raditi, kao primjerice, kretanje i pomicanje drugih objekata. Nagrade se definiraju s metodom *AddReward*. S obzirom da agentu ne dajemo nikakve informacije o okolini u kojoj se nalazi već se oslanjamo na *Raycast* opažanja, metoda *CollectObservations* nije korištena. Parametri koje zahtjeva *RayPerceptionSensorComponent3D* komponenta prikazani su na slici 5.6. Ono što agent može odrediti iz okoline jesu 3 oznake kojima su označeni ostali objekti. Oznakom *Target* označena je meta, oznakom *Object* svi objekti unutar okoline, a oznakom *Wall* zid koji okružuje oktagon. Kada se *collider* zraka (slika 5.5.) dotakne sa *colliderom* drugih objekta podaci se šalju u mozak te s vremenom agent može naučiti određeno ponašanje kada se pojavi očitanje na određenu oznaku.

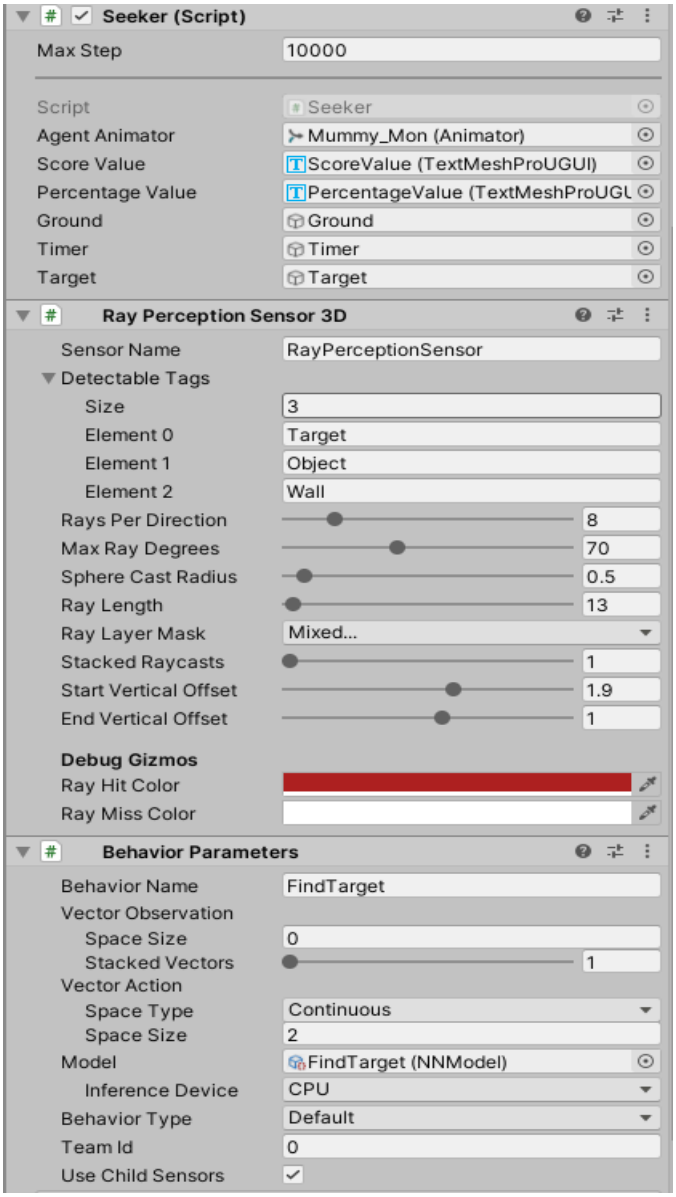


Sl. 5.5. Zrake dodiruju objekte

Parametri postavka ponašanja (slika 5.6.) sastoje se od komponenti:

- **VectorObservation** - veličina se postavlja na nulu jer agentu ne dajemo nikakve informacije o okolini
- **VectorAction** – koristimo kontinuirani tip jer želimo neprekidno prikupljati podatke, u ovom slučaju veličina je 2 jer imamo dvije kontinuirane varijable, rotaciju i kretanje prema naprijed.

- **Model** – ovdje postavljamo datoteku treniranog modela. Ona je generirana nakon svakog završenog treniranja. Trenirati možemo uz pomoć procesora (CPU) ili grafičke procesorske jedinice (GPU).
- **Behavior Type** – možemo postaviti na *inference only* ili *heuristic only*. Ukoliko nema postavljene datoteke treniranja ne možemo koristiti *inference* način dok nam *heuristic* omogućava da sami kontroliramo agenta i testiramo okolinu prije nego pokušamo trenirati agenta u njoj.



Sl. 5.6. Parametri komponenti agenta

Na slikama 5.7. , 5.8. i 5.9. nalazi se C# programski kod korisnički definirane klase agenta. U metodi *OnActionReceived* provjerava se je li isteklo postavljeno vrijeme, ako je iteracija se završava i agent prima negativnu nagradu. Pozivom metode *Move* agent se miče po x i z osi, a rotira po y osi. Na kraju metode agentu se daje mala negativna nagrada kako bi ga se motiviralo da što prije završi zadatak, odnosno da ne stoji u mjestu. Varijabla *MaxStep* predstavlja maksimalan broj koraka po iteraciji, ako se dosegne taj broj iteracija se resetira. U ovom slučaju *MaxStep* je postavljen na 10000 što bi značilo da agent prima negativnu nagradu od -0.001 pri svakom koraku. Kako bi otežali zadatak agentu, nakon svakog resetiranja iteracije postavljamo metu i samog agenta na poziciju koja je nasumično generirana. Za to je zadužena metoda *GetRandomSpawnPos*. Metoda stvara poziciju na x i z osi koja je nasumično generirana te provjerava nalazi li se već koji objekt na toj poziciji. Ukoliko da, generira se nova pozicija. Granice prostora na kojem se generira nasumična pozicija mogu se regulirati sa varijablom *spawnAreaMarginMultiplier*. Prilikom kolizije agenta s objektom poziva se metoda *OnCollision* koja prvo provjerava je li dodirnuti objekt meta koja se traži. Ako je, agent prima veliku nagradu i iteracija se završava, tj. resetira.

```
1  public class Seeker : Agent
2  {
3      public const float MAX_ATTEMPTS = 500.0f;
4
5      public Animator _agentAnimator;
6
7      public TextMeshProUGUI _scoreValue;
8      public TextMeshProUGUI _percentageValue;
9      public TextMeshProUGUI _timeLimitExceededValue;
10
11     public GameObject _ground;
12     public GameObject _timer;
13     public GameObject _target;
14
15     private FindTargetSettings _findTargetSettings;
16
17     private Rigidbody _agentRb;
18
19     private Bounds _areaBounds;
20
21     private Timer _countdownTimer;
22
23     private float _score = 0;
24     private float _countEpisodes = 1;
25     private float _timeExceeded = 2;
26
27     private void Awake()
28     {
29         _findTargetSettings = FindObjectOfType<FindTargetSettings>();
30     }
31
32     public override void Initialize()
33     {
34         _agentRb = GetComponent<Rigidbody>();
35
36         _areaBounds = _ground.GetComponent<Collider>().bounds;
37
38         _countdownTimer = _timer.GetComponent<Timer>();
39     }
40
41     public override void OnActionReceived(float[] vectorAction)
42     {
43         if (_countdownTimer.CheckIfEnds())
44         {
45             _timeExceeded += 1;
46             _timeLimitExceededValue.text = _timeExceeded.ToString();
47
48             AddReward(-1f);
49             EndEpisode();
50         }
51
52         MoveAgent(vectorAction);
53         AddReward(-1 / MaxStep);
54     }
```

Sl. 5.7. Programski kod korisničke klase

```
56      public void MoveAgent(float[] act)
57      {
58          Vector3 _dirToGo = transform.forward * Mathf.Clamp(act[0],-1f,
1f);
59          Vector3 _rotateDir = transform.up * Mathf.Clamp(act[1],-1f,1f);
60
61          transform.Rotate(_rotateDir, Time.deltaTime *
_findTargetSettings._agentTurnSpeed);
62          _agentRb.AddForce(_dirToGo *
_findTargetSettings._agentMoveSpeed, ForceMode.VelocityChange);
63          _agentAnimator.SetFloat("Speed", Mathf.Abs(Mathf.Clamp(act[0],
-1f, 1f)));
64      }
65
66      public override void Heuristic(float[] actionsOut)
67      {
68          actionsOut[1] = Input.GetAxis("Horizontal");
69          actionsOut[0] = Input.GetAxis("Vertical");
70      }
71
72      public override void OnEpisodeBegin()
73      {
74          _countdownTimer.ResetTimer();
75          ResetTarget();
76          ResetAgent();
77
78          _countEpisodes += 1;
79      }
80
81      public Vector3 GetRandomSpawnPos()
82      {
83          bool _foundNewSpawnLocation = false;
84          Vector3 _randomSpawnPos = Vector3.zero;
85
86          while (_foundNewSpawnLocation == false)
87          {
88              var _randomPosX = UnityEngine.Random.Range(-
_areaBounds.extents.x * _findTargetSettings._spawnAreaMarginMultiplier,
89                      _areaBounds.extents.x *
_findTargetSettings._spawnAreaMarginMultiplier);
90
91              var _randomPosZ = UnityEngine.Random.Range(-
_areaBounds.extents.z * _findTargetSettings._spawnAreaMarginMultiplier,
92                  _areaBounds.extents.z *
_findTargetSettings._spawnAreaMarginMultiplier);
93
94              _randomSpawnPos = _ground.transform.position + new
Vector3(_randomPosX, 1f, _randomPosZ);
95
96              if (Physics.CheckBox(_randomSpawnPos, new Vector3(1.5f, 0f,
1.5f)) == false)
97              {
98                  _foundNewSpawnLocation = true;
99              }
100         }
101         return _randomSpawnPos;
102     }
```

Sl. 5.8. Programski kod korisničke klase

```
104    public void ResetTarget()
105    {
106        _target.transform.position = GetRandomSpawnPos();
107    }
108
109    public void ResetAgent()
110    {
111        transform.position = GetRandomSpawnPos();
112        transform.rotation = Quaternion.Euler(new Vector3(0f,
UnityEngine.Random.Range(0, 360)));
113
114        _agentRb.velocity = Vector3.zero;
115        _agentRb.angularVelocity = Vector3.zero;
116    }
117
118    private void OnCollisionEnter(Collision collision)
119    {
120        if (collision.collider.CompareTag("Target"))
121        {
122            AddReward(5f);
123            EndEpisode();
124            ShowSuccessfulAttempts();
125
126            _percentageValue.text =
CalculateSuccesRate().ToString("0.0") + "%";
127        }
128    }
129
130    public void ShowSuccessfulAttempts()
131    {
132        if (_countdownTimer.CheckIfEnds() == false)
133        {
134            if(_countEpisodes == MAX_ATTEMPTS)
135            {
136                Time.timeScale = 0;
137            }
138
139            _score += 1;
140            _scoreValue.text = _score.ToString();
141        }
142    }
143
144    public float CalculateSuccesRate()
145    {
146        return (_score / MAX_ATTEMPTS) * 100;
147    }
148}
```

Sl. 5.9. Programski kod korisničke klase

5.4. Treniranje agenta

Za treniranje agenta zaslužan je eksterni Python proces. On komunicira s agentom unutar scene da bi generirao iskustva. Ta iskustva koristi neuronska mreža kako bi optimizirala pravila koja su zadana agentu. Za stabilno i uspješno treniranje potrebno je postaviti hiperparametre koji se nalaze u .yaml datoteci koja sadrži konfiguracijske detalje svih modela koji se žele trenirati. Hiperparamteri (slika 5.10. koji su izmijenjeni za ovaj projekt su sljedeći:

- **batch_size –** odgovara broju iskustava u svakoj iteraciji gradijentnog spusta. Trebao bi biti višestruko manji od *buffer_sizea*. Tipični raspon vrijednosti za kontinuirani prostor je od 512-5120, a za diskretni prostor je od 32-512.
- **buffer_size –** odgovara broju iskustava koje treba prikupiti prije nego što se započne bilo kakvo učenje ili ažuriranje modela. Treba biti višekratnik *batch_sizea*. Tipični raspon vrijednosti za PPO je od 2048 do 409600.
- **summary_freq –** odgovara broju iskustava koje je potrebno prikupiti prije stvaranja i prikazivanja statistike treniranja. Određuje izgled grafova u Tensorboardu.
- **hidden_units –** odgovara broju skrivenih slojeva neuronske mreže. Za jednostavne probleme trebala bi biti manja vrijednost, gdje za složenije veća. Tipčni raspon vrijednosti od 1 do 3.
- **time_horizon –** odgovara broju koraka za prikupljanje iskustva po agentu prije nego se doda u spremnik iskustava. Kada se ta granica dosegne prije kraja epizode, koristi se procjena vrijednosti za predviđanje ukupne očekivane nagrade iz trenutnog stanja agenta. Tipični raspon vrijednosti je od 32 do 2048.
- **num_epoch –** odgovara broju prolazaka kroz spremnik iskustava tijekom optimizacije gradijentnog spusta. Tipčni raspon vrijednosti od 3 do 10.
- **max_steps –** odgovara broju koraka koji se moraju poduzeti u okolini prije završetka procesa treninga. Tipični raspon vrijednosti je od $5e^5$ do $1e^7$
- **extrinsic strength –** odgovara faktoru pomoću kojeg se množi nagrada koju daje okolina. Tipčni raspon vrijednosti je 1.
- **extrinsic gamma –** odgovara faktoru popusta za buduće nagrade koje dolaze iz okoline. Koliko u budućnosti bi agent trebao brinuti o mogućoj nagradi. Tipčni raspon vrijednosti od 0,8 do 0,995.
- **curiosity strength –** odgovara veličine nagrade za radoznalost koju generira intrinzični modul znatiželje. Tipčni raspon vrijednosti od 0,001 do 0,1.

- **curiosity gamma –** odgovara faktoru popusta za buduće nagrade. Tipčni raspon vrijednosti od 0,8 do 0,995.
- **curiosity encoding_size –** odgovara veličini kodiranja koju koristi intrinzični model znatiželje. Tipični raspon vrijednosti od 64 do 256.

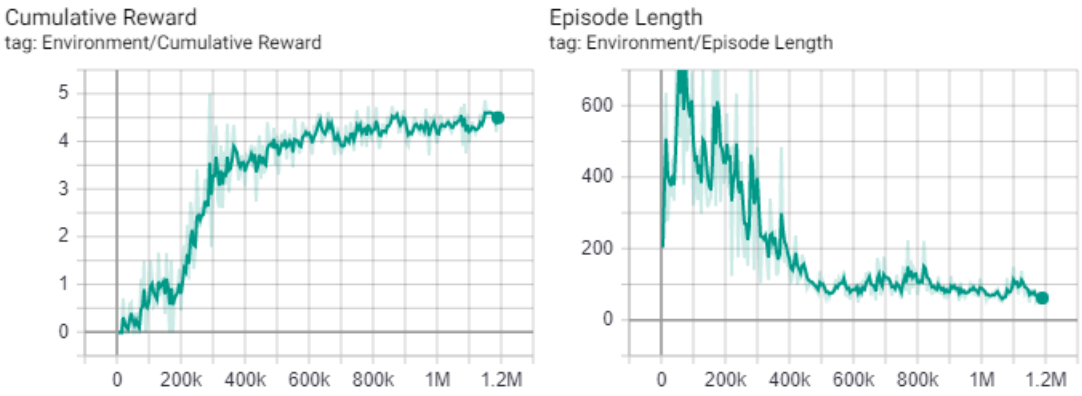
```
FindTarget:
    summary_freq: 5000
    time_horizon: 128
    batch_size: 512
    buffer_size: 5120
    hidden_units: 512
    num_layers: 2
    max_steps: 1.0e7
    num_epoch: 3
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.02
            gamma: 0.99
            encoding_size: 128
```

Sl. 5.10. Hiperparametri za treniranje

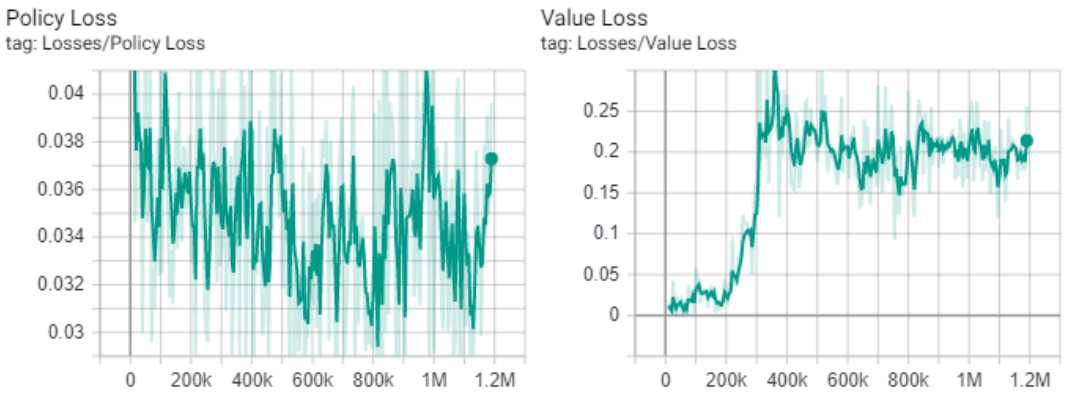
5.5. Rezultati treniranja

Nakon završenog treninga prikupljene podatke možemo pogledati koristeći TensorBoard koji ih vizualizira kroz grafove. Rezultati se mogu pratiti i tijekom treniranja. Na slikama 5.12. 5.11. 5.13. prikazane su vrijednosni grafovi rezultata treniranja agenta za ovaj projekt. Graf *Cumulative Reward* prikazuje prosječnu kumulativnu vrijednost nagrade kroz svaku iteraciju treninga. Ukoliko je trening uspješan, ona će se povećavati do najveće vrijednosti nagrade koja je dana agentu. U ovom slučaju, agentu je dana nagrada u vrijednosti od 5 kada pronađe metu te na grafu možemo vidjeti kako se grafička linija približava vrijednosti 5. Zbog utjecaja okoline, izgradnje okoline, brzine kretanja koja je dana agentu, negativnih nagrada itd., ova vrijednost će rijetko kada biti pri vrhu najveće zadane nagrade. Izuzetak su iznimno jednostavne okoline. U ovom projektu vrijednost se stabilizirala na iznosu od 4.5 s manjim varijacijama pri svakom sljedećem očitavanju. Graf *Episode Length* prikazuje srednju vrijednost trajanja svake iteracije u okolini za sve agente. Početno će iteracije trajati duže dok agent ne nauči pravilno ponašanje. Graf *Policy Loss* prikazuje srednju vrijednost funkcije *policy loss*. *Policy* je proces za odlučivanje koje

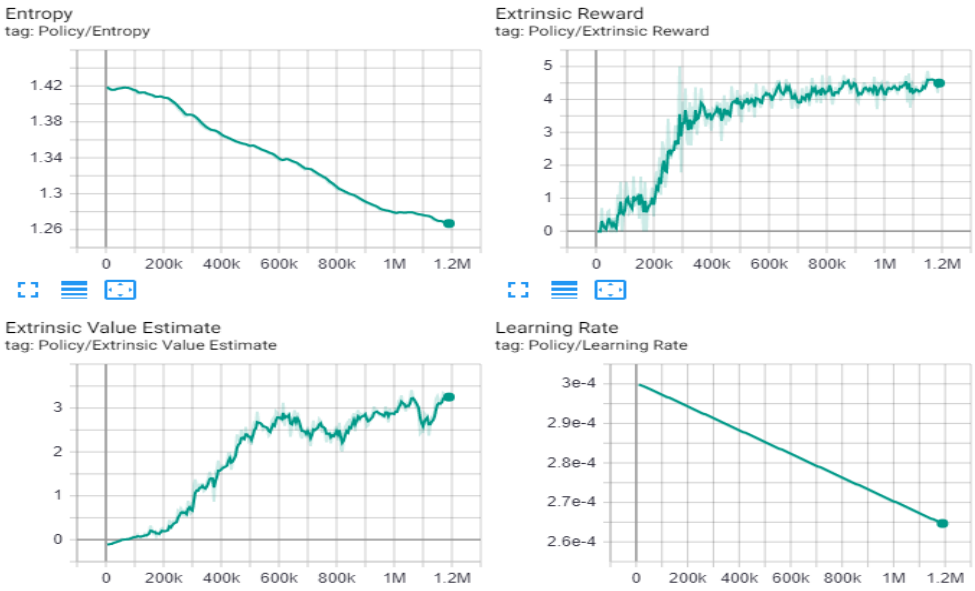
akcije poduzeti. Vrijednost bi trebala padati tokom uspješnog treninga. S obzirom da ova okolina sadrži nepredvidivo ponašanje (nasumično generiranje pozicija) donošenje odluka neće biti olakšano s vremenom. Graf *Value Loss* odgovara prikazu koliko je model u stanju predvidjeti vrijednost svakog stanja. Trebalo bi se povećavati dok agent uči, a zatim smanjiti kad se nagrada stabilizira. Graf *Entropy* prikazuje koliko su nasumične odluke agenta tijekom treninga. Vrijednost bi trebala postepeno opadati kod uspješnog treniranja. Graf *Extrinsic Reward* odgovara srednjoj kumulativnoj nagradi primljenoj od okoline po svakoj iteraciji. Graf *Learning Rate* odnosi se na veličinu koraka kojeg poduzima algoritam treniranja kako bi pronašao optimalan *policy*. Trebalo bi se smanjivati s vremenom. Posljednji graf *Extrinsic Value Estimate* prikazuje srednju vrijednost za sva stanja koja agent posjećuje te bi se vrijednost trebala povećavati tokom uspješnog treninga. Za treniranje agenta u ovom projektu bilo je potrebno 1.2 milijuna koraka. Već pri 800 tisuća koraka primijetilo se da je agent dostigao maksimum te je nagrada stagnirala oko vrijednosti 4,5. ML-Agents podržava treniranje više okolina odjednom. U ovom slučaju korišteno je 16 okolina u isto vrijeme čime je vrijeme treniranja drastično smanjeno (20 minuta) što se i vidi po broju koraka.



Sl. 5.12. Grafički prikaz rezultata treniranja – 1.dio



Sl. 5.11. Grafički prikaz rezultata treniranja – 2.dio



Sl. 5.13. Grafički prikaz rezultata treniranja – 3.dio

Nakon treniranja datoteku sa modelom postavimo u simulaciju. U simulaciji je dodan prikaz statistika koja prati uspješnost naučenih pravila. U svrhu testiranja postavljeno je najviše 500 iteracija kroz koje će agent proći te se bilježe one koje su uspješne, odnosno one u kojima je agent pronašao metu u zadanom vremenu. Uspješnost agenta se automatski računa i prikazuje na ekranu. Na slici 5.14. prikazane su statistike. Od 500 ukupnih iteracija agent je u 483 slučajeva uspješno pronašao metu. Prekoračio je postavljeno maksimalno vrijeme 17 puta. Uspješnost agenta prema ovoj statistici iznosi 96.6%. Simulacija je pokrenuta nekoliko puta te je zaključeno da rezultati variraju što je očekivano zbog prirode problema. Varijacije uspješnosti agenta su između 93% najlošijem slučaju i 96.6% u najboljem slučaju. Istrenirani model nije savršen, no ovo su ipak zadovoljavajući rezultati.

Sl. 5.14. Statistika prikazana u simulaciji

5.6. Problemi u implementaciji i mogućnosti poboljšanja

Tijekom dizajna pojavili su se određeni problemi koji su utjecali na sposobnost agentovog učenja. Trebalo je uskladiti brzinu kretanja s veličinom okoline. Premala brzina i agent neće stići pretražiti cijelu okolinu u zadanom vremenu. Naravno, jedno od rješenja je da se produži zadano vrijeme no parametar vremena postavio se kao referentna točka oko koje su se namještali ostali parametri kako bi se dobila smislena cjelina gdje nećemo čekati predugo vremena da agent pronađe predmet. Prevelika brzina bi uzrokovala poteškoće pri kretanju gdje je čak i umjetna inteligencija imala probleme, a testiranje kao igrač je bilo iznimno teško. Broj objekata je također bitan. Veći broj objekata smanjuje površinu gdje bi se predmet mogao pojaviti i znatno otežava pronalaženje jer je mnogo objekata koji bi se potencijalno nalazili između agenta i predmeta. Agent bi često prošao par puta pokraj predmeta jer mu je, primjerice, kutija zaklanjala predmet pod nezgodnim kutom. Mali broj objekata opet ne predstavlja dovoljan izazov da bi cjelina imala smisao i bila zanimljiva za promatrati. FOV (engl. *field of view*) predstavlja veličinu opažanja agenta (slika 5.15.). Povećanje duljine *raycast* zraka poboljšalo bi rezultat za male margine u slučaju gdje bi agent brže pronašao predmet na otvorenijoj površini, odnosno „vidio“ bi dalje pa bi mu predmet dolazio ranije u FOV, što pomaže kod slučajeva gdje odluči potražiti predmet na nekoj drugoj lokaciji iako mu je skoro ušao u FOV. No često je predmet sakriven iza nekog drugog objekta pa ova komponenta treba biti postavljena u skladu sa smislenom cjelinom i ipak pružiti agentu dodatni izazov.

Najveći problem predstavlja namještanje hiperparametara kojih ima mnogo i proces pronalaska onih zadovoljavajućih je dugotrajan. Srećom ML-Agents sadrži mnogo primjera problema čije konfiguracije mogu poslužiti kao početna točka ukoliko je problem sličan onom u vlastitom projektu. U ovom projektu su namješteni samo oni na koje je agent pokazao najveću osjetljivost.

Sl. 5.15. FOV agenta

6. ZAKLJUČAK

Ovaj završni rad poslužio je kao prikaz mogućnosti algoritama strojnog učenja u kombinaciji sa programskim okruženjem Unity. Simulacijski dio ovog projekta sadržava relativno jednostavan problem gdje se treniranom modelu najviše otežava s nasumičnom generacijom pozicije mete. Bilo je potrebno isprobati nekoliko vrijednosti parametara treniranja te korigirati okolinu kako bi agent uspješno riješio zadatak. Osim rezultata fascinira i vrijeme koje je bilo potrebno agentu da riješi problem, a da se pritom koristi računalo prosječnih performansi.

Iako je ML-Agents noviji dodatak za Unity, privlači sve više pozornosti i ima konstantu podršku za nadogradnje. Omogućava rješavanje jednostavnih ali i kompleksnih problema. Potencijal da se testira model strojnog učenja u simulaciji koja kasnije može biti implementirana u računalo ili stroj, prepoznata je od strane mnogih stručnjaka umjetne inteligencije. Treba naglasiti da je Unity besplatan te je time otvorena mogućnost svima koji žele testirati vlastite projekte. ML-Agents će se nesumnjivo razvijati i dalje te ubrzati proces razvoja novih tehnologija.

LITERATURA

[1] Unity blog, https://blogs.unity3d.com/2020/05/12/announcing-ml-agents-unity-package-v1-0/ , posjećeno 15.06. 2020.

[2] ML-agents, https://unity.com/products/machine-learning-agents#millions-natural-feeling-procedurally-generated-monsters--2, posjećeno 15.06.2020.

[3] Unity blog, https://blogs.unity3d.com/2019/04/15/unity-ml-agents-toolkit-v0-8-faster-training-on-real-games/, posjećeno 15.06.2020.

[4] OpenAI, https://openai.com/blog/emergent-tool-use/, posjećeno 15.06.2020.

[5] M. Bareš (2018) Strojno učenje u Unityu, Diplomski rad, Osijek: Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija.

[6] Unity, https://unity.com/, posjećeno 18.06.2020.

[7] J. Albahari i B. Albahari, *C# in nutshell*, str. 1-5, O'Reilly Media Inc., Sjedinjene Američke Države, 2018.

[8] Introduction to C# programming language, https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework, posjećeno 19.06.2020.

[9] ML-Agents Github, https://github.com/Unity-Technologies/ml-agents, posjećeno 22.06.2020.

[10] ML-Agents Overview, https://github.com/Unity-Technologies/ml-agents/blob/release_3_docs/docs/ML-Agents-Overview.md, posjećeno 22.06.2020.

[11] Tensorflow, https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Background-TensorFlow.md, posjećeno 24.06.2020.

[12] Using TensorBoard, https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md, posjećeno 24.06.2020.

[13] N. Buduma, *Fundamentals of Deep Learning*, str.7 i str.245-248, O'Reilly Media Inc., Sjedinjene Američke Države, 2017.

[14] How deep learning works, https://www.mathworks.com/discovery/deep-learning.html, posjećeno 10.07.2020.

[15] Agent visual observations, https://github.com/Unity-Technologies/ml-agents/blob/release_3_docs/docs/Learning-Environment-Design-Agents.md#visual-observations, posjećeno 13.07.2020.

SAŽETAK

Cilj ovog završnog rada je pokazati mogućnosti koje donose strojno i duboko učenje kada ih se kombinira s moćnim alatima za stvaranje virtualnih simulacija. Ovo je omogućeno kroz Unity *game engine* koji se koristi za izradu simulacija i razvoj 2D i 3D video igara te ML-Agents dodatak koji sadrži algoritme za treniranje inteligentnih agenata. Opisani su osnovni principi i algoritmi dubokog učenja uz pomoć kojih agent percipira okolinu oko sebe i uči rješavati određeni zadatak. U sklopu rada izrađena je simulacija u kojoj agent pokušava pronaći skriveni predmet u određenom vremenu koji se postavi na novo nasumično generirano mjesto nakon pronalaska. Agent uči korištenjem strojnog učenja s potporom. Primjenjujući odgovarajuću implementaciju i parametre agent je uspješno naučio svladati dani problem.

Ključne riječi: duboko učenje, ML-Agents, agent, učenje s potporom, opažanje

ABSTRACT

Purpose of this bachelor's thesis is to show possibilities of machine learning and deep learning in combination with powerful tools which are used in making virtual simulations. This is enabled through Unity game engine, a software which is used for developing 2D and 3D video games and simulations, and ML-agents plugin which contains algorithms for training intelligent agents. Basic principles and algorithms of deep learning are described. Agent uses those methods for perception of environment and solving certain tasks. Within thesis, simulation is created in which agent is trying to find hidden item by using reinforcement machine learning. If agent finds item, new place for item is randomly generated. Also, he must complete task in certain time. With the right implementation and parameters, agent solved the task successfully.

Keyword: deep learning, ML-Agents, agent, reinforcement learning, perception

ŽIVOTOPIS

David Hodak rođen je 20.04.1997 u Puli. Od 2004. do 2012. pohađa OŠ Ivane Brlić Mažuranić. Nakon toga upisuje Gimnaziju Matije Antuna Reljkovića u Vinkovcima. Za vrijeme srednjoškolskog obrazovanja sudjeluje u natjecanjima iz osnova informatike te osvaja treće mjesto na županijskom natjecanju u trećem razredu. Gimnaziju završava 2016. godine. kada upisuje preddiplomski smjer računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek koji i dalje pohađa.