UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

NASPROTNIŠKO SPODBUJEVALNO UČENJE V RAČUNALNIŠKIH IGRAH ZA DVA IGRALCAA

MATIC STARE

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Nasprotniško spodbujevalno učenje v računalniških igrah za dva igralcaa

(Adversarial reinforcement learning in playing video games for two players)

Ime in priimek: Matic Stare
Študijski program: Računalništvo in informatika
Mentor: izr. prof. dr. Matjaž Kljun
Somentor: asist. dr. Domen Šoberl

Koper, 2023

Zahvala

Zahvaljujem se svojima mentorjema za strokovno pomoč, spodbudo in potrpežljivost pri nastajanju tega diplomskega dela.
Hvala tudi vsem ostalim, ki so mi vsa leta pomagali pri študiju in mi stali ob strani.

Ključna dokumentacijska informacija

Ime in PRIIMEK: Matic STARE

Naslov zaključne naloge: Nasprotniško spodbujevalno učenje v računalniških igrah za dva igralcaa

Kraj: Koper

Leto: 2023

Število strani: 32 Število slik: 11

Število referenc: 8

Mentor: izr. prof. dr. Matjaž Kljun

Somentor: asist. dr. Domen Šoberl

Ključne besede: spodbujevalno učenje, Q-učenje, umetne nevronske mreže, arkadno učno okolje, algoritem DQN

Izvleček:
Področje strojnega učenja se v zadnjih letih pospešeno razvija. Del tega je tudi spodbujevalno učenje, ki se uporablja za učenje kompleksnih okolij, kot so na primer igre na simulaciji igralne konzole Atari 2600. Namen diplomske naloge je ugotoviti, kako uspešno se dva agenta spodbujevalnega učenja uspeta naučiti igrati igro Pong v nasprotniški postavitvi. S pomočjo programskega jezika Python in knjižnic PettingZoo ter Tensorflow smo implementirali algoritem globokega spodbujevalnega učenja (DQN) brez konvolucijskih plasti, saj smo namesto zaslonske slike igre izvedli vpogled v pomnilnik in od tam prebrali vhodne vrednosti. Za eksperimentalni del smo dve nevronski mreži pustili igrati igro Pong v nasprotniški postavitvi. Ugotovili smo, da po zadostnem številu učnih korakih nevronski mreži razvijeta napadalno strategijo, ki jima pomaga pri doseganju zadetkov. Rezultate smo prikazali na dva načina, in sicer z merjenjem dolžine posamezne epizode in s štetjem odbojev posameznega agenta v vsaki epizodi.

Key document information

Name and SURNAME: Matic STARE

Title of the final project paper: Adversarial reinforcement learning in playing video games for two players

Place: Koper

Year: 2023

Number of pages: 32 Number of figures: 11

Number of references: 8

Mentor: Assoc. Prof. Matjaž Kljun, PhD

Co-Mentor: Assist. Domen Šoberl, PhD

Keywords: reinforcement learning, Q-learning, artificial neural network, The Arcade Learning Environment, DQN Algorithm

Abstract:
The field of machine learning has been rapidly advancing in recent years. Part of it includes reinforcement learning, which is used to learn complex environments, such as games on the Atari 2600 gaming console simulation. The purpose of this thesis is to determine how successful two reinforcement learning agents manage to learn the game of Pong in an adversarial setup. With the help of the Python programming language and libraries PettingZoo and TensorFlow we implemented the deep reinforcement learning algorithm (DQN) without convolutional layers, because we gained insight into the game's memory and read input values from there. For the experimental part, we let two neural networks play the game Pong in an adversarial setup. We found that after a sufficient number of learning steps, the neural networks develop an offensive strategy that helps them score points. We presented the results in two ways: by measuring the length of each episode and by counting the hits of the ball for each agent in every episode.

Kazalo vsebine

Kazalo slik in grafikonov

Seznam kratic

ALE	Arcade Learning Environment
DQN	Deep Q-Network
MSE	Mean Square Error
TanH	Hyperbolic Tangent Function
ReLU	Rectified Linear Unit
ADAM	Adaptive Moment Estimation

1 UVOD

V zadnjih letih se je z razcvetom umetne inteligence področje strojnega učenja začelo pospešeno razvijati. Del tega je tudi spodbujevalno učenje [6], ki se osredotoča na učenje igralcev (agentov). Ti so postavljeni v okolje, v katerem morajo sprejemati odločitve, ki jih pripeljejo bližje cilju, kot odziv na izvedene akcije pa prejmejo nagrado oziroma kazen. Najbolj znana oblika spodbujevalnega učenja je Q-učenje, ki za ocenjevanje koristnosti akcij v nekem stanju uporablja Q-funkcijo. Na ta način se agent po zadostnem številu korakov lahko nauči precej dobro odločati, celo boljše od človeka. Ker je v večini učnih okolij različnih stanj in odločitev preveč, da bi Q-funkcijo hranili v obliki matrike, Q-funkcijo aproksimiramo z nevronsko mrežo. Ta se med treningom uči igralno strategijo, kasneje pa agent na njeni podlagi sprejema svoje odločitve.

Prva uspešna implementacija globokega spodbujevalnega učenja je bila predlagana leta 2013, ko je V. Minh s sodelavci z uporabo globoke nevronske mreže implementiral algoritem Q-učenja, ki se je lahko naučil igranja sedmih različnih iger v simulatorju igralne konzole Atari 2600 [5]. Ena izmed njih je bila igra Pong, ki jo uporabljamo v tem diplomskem delu. Avtorji so uspeli nevronsko mrežo natrenirati do te mere, da je igro Pong lahko igrala bolje od človeka. Podobno smo v diplomskem delu poskusili tudi mi, le da smo uporabili različico igre Pong za dva igralca in trenirali dva agenta enega proti drugemu.

Cilj te diplomske naloge je ugotoviti, kako uspešno bi se dva agenta globokega spodbujevalnega učenja lahko naučila igrati igro Pong v nasprotniški postavitvi, tako da igrata eden proti drugemu. Delo zajema implementacijo Q-učenja v programskem jeziku Python z uporabo knjižnic TensorFlow [1] in Keras ter vzpostavitev arkadnega učnega okolja (ALE) [2] s pomočjo knjižnice PettingZoo [7], ki je razširitev knjižnice openAI Gymnasium [8] s podporo igranja igre za dva igralca, na katerem je igra simulirana. Nevronski mreži smo trenirali več dni, dokler način igranja obeh igralcev ni konvergiral k določeni strategiji. To pomeni, da vsako nadaljnje učenje ne bi prineslo nikakršnega napredka v uspešnosti igranja igre. Preizkusili in primerjali smo rezultate različnih globokih arhitektur nevronskih mrež in različnih vrednosti učnih parametrov, ki smo jih poskusili optimizirati.

V diplomskem delu bomo najprej predstavili teorijo spodbujevalnega učenja in Q-učenja ter opisali nevronske mreže, ki so potrebne za učenje računalniških igralcev. Sledi predstavitev implementacije, kar zajema opis posameznih komponent, ki so bile uporabljene in implementirane. Na koncu bomo predstavili rezultate dela in jih evalvirali. Kot zaključek sledi diskusija o možnih izboljšavah in nadaljnjem delu.

2 GLOBOKO SPODBUJEVALNO UČENJE

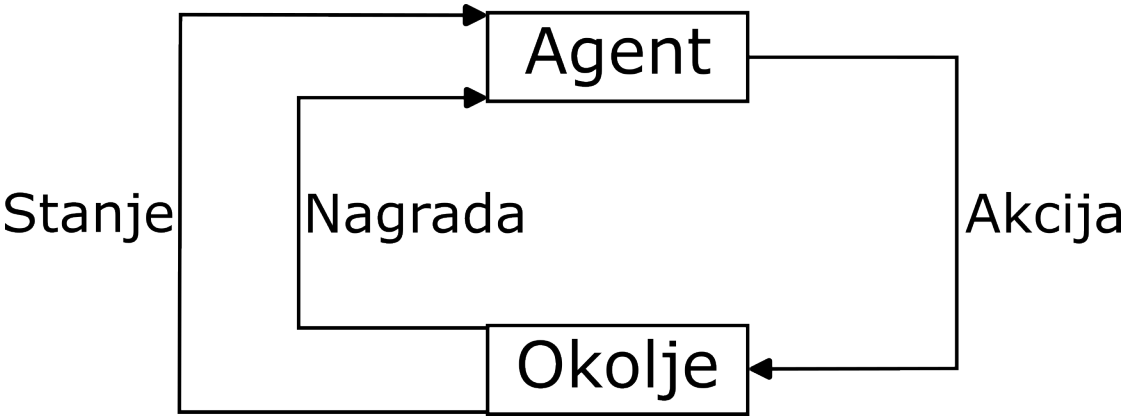
2.1 SPODBUJEVALNO UČENJE

2.1.1 Kaj je spodbujevalno učenje

Spodbujevalno učenje je področje strojnega učenja, ki se ukvarja z metodami agentnega učenja. Agent se v svojem okolju uči sprejemati odločitve, ki bi pripeljale do želenega cilja. Pri tem ga vodi funkcija nagrajevanja, od katere za dosežen cilj ali podcilj prejme nagrado. Agent se uči tako, da poskuša nagrado optimizirati, kar ob ustrezni definiciji funkcije nagrajevanja privede do odkritja uspešne strategije doseganja cilja [6]. Ta princip se v vsakdanjem življenju pojavi vsakič, ko se želimo naučiti nečesa novega. V primeru učenja vožnje s kolesom bi spodbujevalno učenje lahko opisali takole: Na začetku še ne vemo, kakšna je naša najboljša strategija vožnje, zato z različnim zaporedjem akcij poskušamo poganjati pedala, usmerjati krmilo in ostati v ravnovesju. V prvih nekaj poskusih verjetno ne bomo uspešni, zato potrebujemo trening. Z vsakim poskusom nam gre bolje in kolo lahko vozimo vedno dlje. Zaradi uspešnosti dobimo občutek zadovoljstva, ki je pravzaprav nagrada za naše delo. Včasih pa se vseeno zgodi, da pademo. Zato je pomembno, da naše odločitve o izbiri akcij ustrezno prilagodimo tako, da se to v nadaljnje to ne bo več zgodilo. Po zadostni količini treninga znamo kolo že precej dobro voziti, kar pomeni, da se znamo v vsakem trenutku pravilno odločiti, kaj storiti, da ne pademo. Agent spodbujevalnega učenja na začetku v svojem okolju prav tako ne zna sprejemati odločitev, ki bi ga pripeljale do želenega rezultata. Med treningom se nato preko zaporedja poskusov in napak nauči sprejemati vedno boljše odločitve. Bolj kot je okolje kompleksno oziroma več kot je parametrov in akcij, več korakov agent potrebuje, da odkrije uspešno strategijo sprejemanja odločitev.

2.1.2 Osnovni koncepti spodbujevalnega učenja

Spodbujevalno učenje najlažje prikažemo z modelom na sliki 1. Osnovni koncepti spodbujevalnega učenja so *agent*, *okolje*, *stanje*, *nagrada* in *akcija*. Vsakega izmed njih bomo na kratko predstavili.



Slika 1: Model spodbujevalnega učenja

- **Agent** je entiteta, ki je postavljena v okolje in od njega pridobiva stanje, na podlagi katerega se odloči za naslednjo akcijo. Agent po vsaki izvedeni akciji prejme nagrado, ki je lahko pozitivna ali negativna. Koristnejša akcija prinese višjo nagrado, pri čemer negativna nagrada (kazen) lahko pomeni škodljivo akcijo.
- **Okolje** je prostor, v katerega je agent postavljen in na katerega s svojimi akcijami vpliva. Okolje je lahko zvezno ali diskretno.
- **Stanje** prikazuje trenutno situacijo agenta znotraj okolja oziroma njegovo konfiguracijo. Agent lahko stanje opazuje in iz njega zbira informacije, ki mu pomagajo pri izbiri akcij.
- **Nagrada** ja povratna informacija agentu po vsaki izvedeni akciji, ki mu sporoča kakovost njegove odločitve. Agentov cilj je maksimirati nagrado v vsakem koraku in s tem povečati skupno nagrado.
- **Akcija** je odločitev, ki jo agent sprejme v nekem stanju. Ta sproži prehod iz enega stanja v drugega.

V okolju spodbujevalnega učenja je agent seznanjen s trenutnim stanjem in na podlagi tega s pomočjo Q-funkcije, ki jo bomo predstavili v naslednjem poglavju 2.1.3, izbere in izvede naslednjo akcijo. S tem posodobi stanje in od okolja prejme nagrado, na podlagi katere posodobi Q-funkcijo. Tako Q-funkcija (ki jo v našem primeru aproksimiramo z nevronsko mrežo) počasi kovergira k neki sprecifični strategiji.

2.1.3 Q-učenje

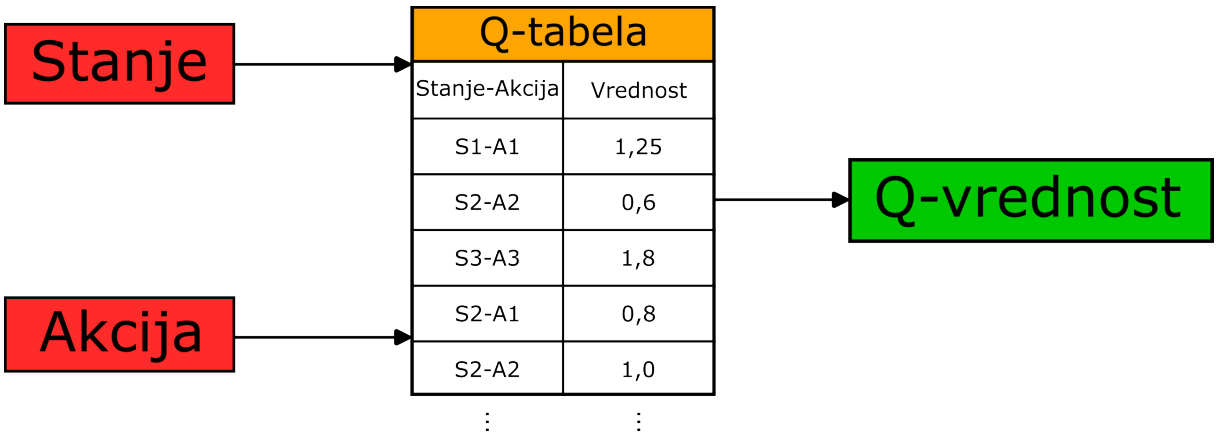
Q-učenje je ena izmed najpogosteje uporabljenih izvedb spodbujevalnega učenja. Deluje tako, da agent vsakemu paru (s, a) priredi vrednost Q, ki pomeni kvaliteto izvedene akcije a v stanju

s. Vrednost Q se izračuna po Bellmanovi enačbi:

$$\underbrace{\text{New}Q(s,a)}_{\text{Nova Q-vrednost}} = \overbrace{Q(s,a)}^{\text{Q-vrednost}} + \underbrace{\alpha}_{\text{Hitrost učenja}} [\underbrace{R(s,a)}_{\text{Nagrada}} + \underbrace{\gamma}_{\text{Stopnja pojemanja}} \overbrace{\max Q'(s',a')}^{\text{Najvišja napovedana nagrada glede na trenutno stanje}} - Q(s,a)] \tag{2.1}$$

Črka Q v besedi Q-učenje izhaja iz angleške besede *Quality*, ki pomeni kvaliteta oziroma koristnost vsake akcije h končnem cilju.

Osnovno Q-učenje poteka tako, kot je prikazano na sliki 2.



Slika 2: Q-učenje s pomočjo Q-tabele

Agent v vsakem koraku opazuje svoje trenutno stanje in vzame v obzir nabor vseh možnih akcij v tem stanju. S pomočjo Bellmanove enačbe (2.1) nato izračuna pričakovano Q-vrednost ob izvedbi vsake od možnih akcij. Nato izvede akcijo z najvišjo Q-vrednostjo in na podlagi prejete nagrade posodobi vrednosti v Q-tabeli.

Algoritem Q-učenja lahko na kratko opišemo z naslednjo psevdokodo:

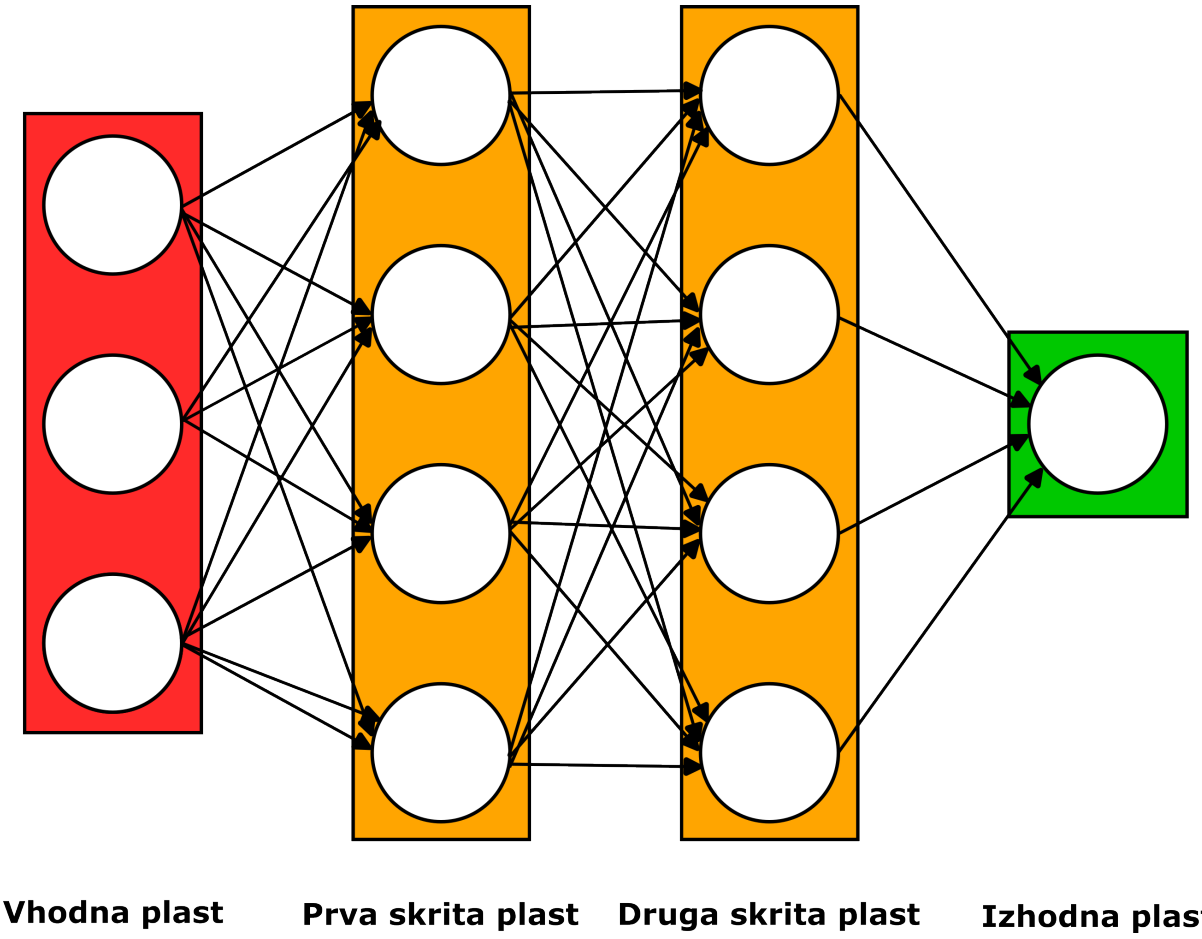
```
1. Inicializacija Q-vrednosti na 0.
2. Za vsak korak znotraj epizode:
    2.1 Agent glede na trenutno stanje izbere akcijo z najvišjo
        Q-vrednostjo.
    2.2 Agent izvede akcijo in opazuje njen učinek (pridobi novo
        stanje in nagrado).
    2.3 Agent z uporabo enačbe (2.1) na podlagi prejete nagrade
        posodobi Q-vrednosti prejšnjega stanja in izvedene
        akcije.
3. Evalvacija uspešnosti epizode.
4. Ponovi koraka 1 in 2 za podano število epizod.
```

2.2 UMETNE NEVRONSKE MREŽE

Z uporabo Q-tabele lahko implementiramo Q-učenje le v diskretnih domenah, zato v zveznih domenah preslikavo med pari stanje-akcija in Q-vrednostmi realiziramo preko realne funkcije. Učinkovit splošno-namenski aproksimator funkcij so nevronske mreže, ki lahko aproksimirajo, sicer z neko točnostjo, a ne glede na njihovo kompleksnost, poljubno matematično funkcijo. Točnost aproksimacije nevronske mreže je odvisna od velikosti njene arhitekture, števila iteracij preko učnih podatkov in narave učne množice podatkov. Bolj kot so podatki kompleksni, več plasti potrebujemo, da je aproksimacija funkcije točnejša.

Umetne nevronske mreže so sestavljene iz nevronov, ki so organizirani v plasti. Vsaka nevronska mreža je sestavljena iz ene ali več skritih plasti. Izhod posameznega nevrona je preko njih povezan na vhod naslednjega. Poznamo več vrst plasti umetnih nevronskih mrež. Prvi plasti rečemo vhodna, zadnji izhodna, vsem vmesnim plastem pa skrite plasti. Velikost vhodne plasti je neposredno povezana s številom vhodnih parametrov, saj potrebujemo toliko vhodnih nevronov, kot imamo vhodnih parametrov. Enako velja za izhodno plast, ki je velika toliko, kot potrebujemo izhodnih parametrov. Vmesne skrite plasti pa so odvisne od kompleksnosti učenja. Če želimo model naučiti nečesa bolj kompleksnega, potrebujemo več in večje skrite plasti. To pomeni, da nevronska mreža aproksimira matematično funkcijo treh neodvisnih spremenljivk, ki poljubno trojico realnih vrednosti enolično preslika v izhodno realno vrednost.

Primer nevronske mreže je prikazan na sliki 3. Pri tem primeru je prva plast sestavljena iz treh nevronov, ki se preko dveh skritih plasti povezuje na en nevron izhodne plasti.



Slika 3: Arhitektura napajalne nevronske mreže

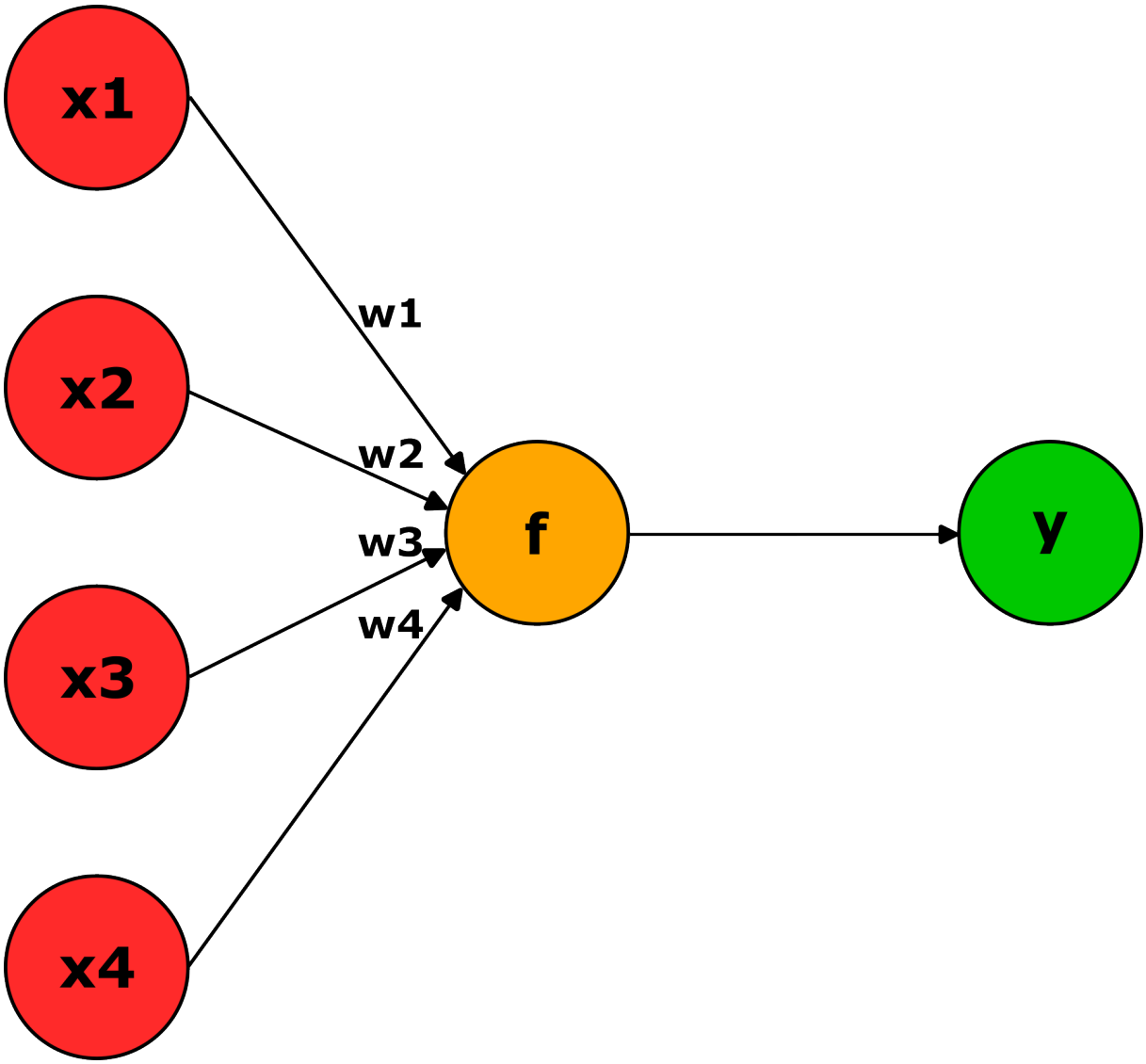
Glede na smer pretoka podatkov poznamo dve vrsti nevronskih mrež: napajalna (angl. *feedforward*) in ponavlajoča se (angl. *recurrent*). Pri prvi podatki tečejo od vhodne plasti enosmetno do izhodne. Za razliko od nje se pri drugi podatki lahko vračajo v predhodne plasti z namenom izboljšanja učenja. V diplomskem delu obravnavamo le napajalne nevronske mreže.

Delovanje nevronskih mrež lahko opišemo na sledeč način. V vsakem koraku dobi vhodne podatke, ki so navadno parametri okolja. Ti so podani vhodni plasti. Preko nevronskih povezav nato potujejo skozi skrite plasti proti izhodni plasti, pri čemer povezave med nevroni delujejo kot uteži. Na ta način nevronska mreža deluje pravzaprav kot preslikava ali funkcija, ki vhodne podatke preslika v izhodne. Na aktivacijo nevrona vpliva aktivacijska funkcija. Njen namen je v nevronsko mrežo vnesti nelinearnost, kar ji omogoča učenje tudi nelinearnih funkcij.

Vrednost posameznega nevrona izračunamo po naslednji enačbi:

$$y = f\left(b + \sum_{i=1}^{n}(w_i \cdot x_i)\right) \tag{2.2}$$

V tej enačbi x_i predstavlja vhodne vrednosti nevrona, w_i uteži vhodnih povezav, b prag proženja, funkcija f pa aktivacijsko funkcijo. Na sliki 4 je predstavljeno delovanje umetnega nevrona tudi vizualno.



Slika 4: Delovanje umetnega nevrona

Najpogosteje uporabljene aktivacijske funkcije so:

- *Sigmoidna funkcija* vhodne vrednosti preslika na interval (0,1), kar je uporabno predvsem pri binarni klasifikaciji.
- *Funkcija TanH* je podobna sigmoidni funkciji le, da vhod preslika na interval (-1,1).
- *ReLU* (Rectified Linear Unit) je aktivacijska funkcija, ki je zaradi svoje enostavnosti najpogosteje uporabljena pri globokih nevronskih mrežah. Zanjo je značilno, da vse negativne vhodne parametre preslika v 0, ostale pa pusti take kot so. To pomeni, da so v vsaki iteraciji aktivni le nenegativni nevroni.

2.2.1 Učenje nevronskih mrež

Učenje nevronskih mrež poteka iterativno. Na začetku se uteži nastavijo naključno. To pomeni, da je tudi matematična funkcija nevronske mreže na začetku naključna in pri napovedovanju izhodnih vrednosti dela relativno veliko napako. Potem se med učenjem ta funkcija in uteži

prilagajajo in v vsakem koraku izboljšujejo. Funkcijo, ki določi napako nevronske mreže, imenujemo funkcija izgube (angl. *loss function*). Pri učenju nevronskih mrež se pogosto uporablja povprečna kvadratna napaka (angl. *Mean Square Error* ali MSE), ki smo jo uporabili tudi v tem diplomskem delu. MSE je definirana kot:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{2.3}$$

Glavni namen učenja nevronskih mrež je, da se to napako minimizira. Za minimizacijo napake se najpogosteje uporablja metoda vzvratnega razširjanja napake (angl. *backpropagation*), ki deluje tako, da izhodno napako po vsakem izračunu izhodnih vrednosti širi po nevronski mreži od izhodne preko skritih to vhodne plasti. Med potjo uteži prilagodi tako, da se za trenutno aktivni primer izhodna napaka nekoliko zmanjša. Da se nevronska mreža na ta način bistveno izboljša, je potrebno postopek na velikem številu primerov več sto ali tisočkrat ponoviti. Ob ustrezno izbrani arhitekturi za dano domeno lahko pričakujemo, da bo nevronska mreža naučeno teorijo posplošila tudi na primere, ki jih s to metodo ni obdelala ter dajala pravilne napovedi tudi za primere, ki se v učni množici niso pojavili.

Postopek učenja nevronske mreže lahko opišemo na sledeči način:

1. Vhodna plast nevronske mreže sprejme enega ali več učnih primerov.
2. Učni nevroni aktivirajo vhodne nevrone glede na trenutno nastavljene vhodne uteži.
3. Vhodni nevroni glede na trenutno vrednost aktivirajo skrite plasti, te pa izhodno plast.
4. Funkcija izgube izračuna napako izhodne plasti.
5. Algoritem vzvratnega razširjanja uravnava uteži od izhodnih proti vhodnim plastem tako, da zmanjšuje napako.
6. Zgornje korake ponavljamo, dokler ne dosežemo želene točnosti izhodnih vrednosti.

2.2.2 Uporaba nevronskih mrež pri Q-učenju

Globoko Q-učenje se od klasičnega spodbujevalnega Q-učenja razlikuje po tem, da Q-tabelo zamenjamo z umetno nevronsko mrežo. To je primerno v situacijah, ko je okolje preveč kompleksno, da bi hranili vsa možna stanja v tabeli, zato odločitve aproksimiramo z umetno nevronsko mrežo. Prvi znani algoritem globokega Q-učenja DQN [5] deluje na principu konvolucije, kar pomeni, da kot vhod prejme kar surovo zaslonsko sliko, iz katere konvolucijske plasti nevronske mreže izluščijo informativne atribute. V našem primeru konvolucijske plasti izpustimo in atribute stanja podamo neposredno, kar bistveno zmanjša prostor stanj in pospeši proces učenja.

To smo storili, ker dodajanje drugega učečega agenta v igro, kjer se je v predhodnih DQN implementacijah učil en sam agent, bistveno poveča prostor stanj, ki ga je potrebno preiskati, naš čas in naši razpoložljivi viri pa so bili omejeni.

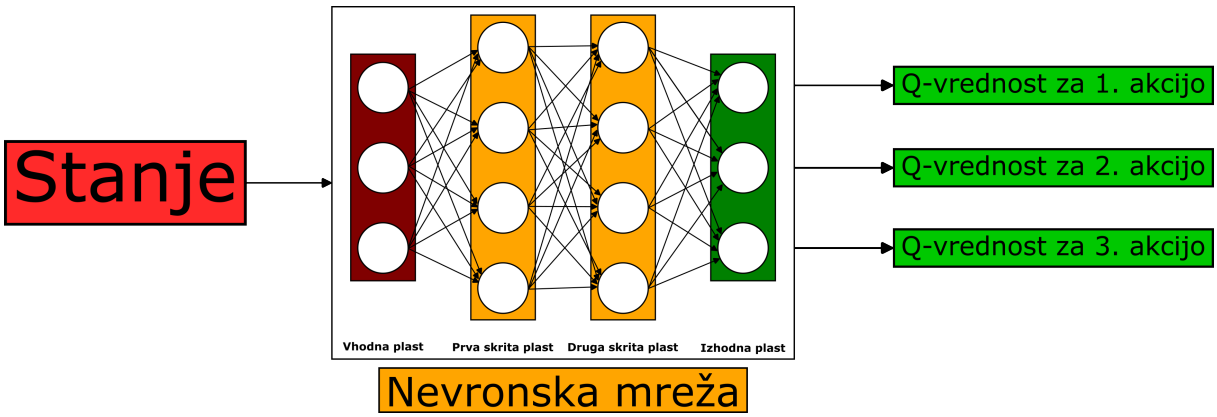
Vsaka agentova akcija proizvede novo izkušnjo, ki jo agent shrani v svoj pomnilnik izkušenj. Ta izkušnja je sestavljena iz štirih komponent: trenutno stanje, izvedena akcija, prejeta nagrada in novo stanje (kot posledica izvedene akcije). To pomeni, da vsak zapis v pomnilniku izkušenj vsebuje dve zaporedni stanji.

Ko se v pomnilniku izkušenj nabere dovolj izkušenj za vsaj en sveženj (angl. *batch*), nevronska mreža prične z učenjem. V našem primeru je velikost svežnja konstantna, in sicer 32 izkušenj. Po vsaki izvedeni akciji agent novo pridobljeno izkušnjo doda v pomnilnik izkušenj, iz njega pa naključno izbere 32 izkušenj, nad katerimi nevronska mreža izvede en učni korak. To pomeni, da vsakemu koraku igranja sledi učni korak nad naključno izbranim svežnjem 32-ih preteklih izkušenj. Ta pristop k učenju smo prevzeli od algoritma DQN.

Algoritem globokega Q-učenja lahko predstavimo z naslednjo psevdokodo:

```
1. Inicializiraj pomnilnik izkušenj.
2. Inicializiraj igralno in tarčno umetno nevronsko mrežo
   z naključnimi utežmi.
3. Ponovi naslednje korake za M epizod:
    3.1 Inicializiraj začetno stanje igre.
    3.2 Glede na vrednost stopnje preiskovanja izberi naključno
        akcijo ali s pomočjo tarčne nevronske mreže.
    3.3 Izvedi akcijo in pridobi nagrado ter opazuj naslednje
        stanje.
    3.4 Pridobljeno izkušnjo shrani v pomnilnik izkušenj.
    3.5 Iz pomnilnika izkušenj izberi sveženj k naključnih
        izkušenj in z uporabo Bellmanove enačbe izračunaj
        dejanske Q-vrednosti.
    3.6 Izvedi učni korak igralne nevronske mreže nad svežnjem
        izračunanih dejanskih Q-vrednosti.
    3.7 Če je že preteklo določeno število korakov, prepiši uteži
        igralne nevronske mreže v tarčno nevronsko mrežo.
```

Slika 5 prikazuje preslikavo stanja v akcijo, ki naj bi jo agent v danem stanju izvedel. Nevronska mreža vsaki akciji priredi Q-vrednost, algoritem pa nato izbere najvišje ocenjeno akcijo. Učenje nevronske mreže poteka tako, da za izbrani sveženj izkušenj z uporabo Bellmanove enačbe (2.1) izračunamo dejanske Q-vrednosti vsake akcije in nad temi dejanskimi vrednostmi izvedemo učni korak s povratnim širjenjem napake.



Slika 5: Q-učenje s pomočjo nevronske mreže

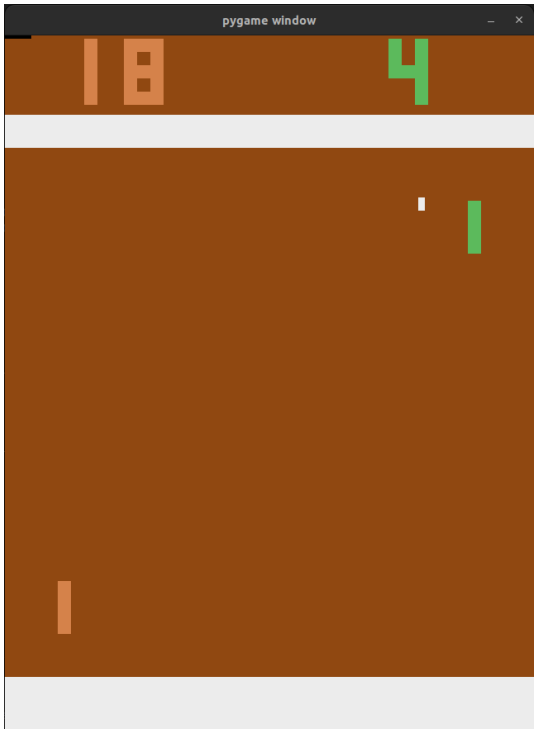
3 IMPLEMENTACIJA

Algoritem spodbujevalnega učenja smo implementirali v programskem jeziku Python, s pomočjo knjižnic Tensorflow [1] in Keras [4]. Ker smo potrebovali podporo za dva igralca, smo za simulacijo igre Pong namesto osnovnega okolja Gymnasium [8] uporabili njegovo ovojnico z imenom PettingZoo [7]. Vse omenjene knjižnice smo namestili z uradnim Pythonovim upraviteljem paketov Pip [3].

3.1 ARKADNO UČNO OKOLJE

Arkadno učno okolje je ogrodje, ki simulira igralno konzolo Atari 2600 in nam omogoča enostaven razvoj agentov za igranje poljubne igre, ki jo ta konzola podpira [2]. Na voljo je več različnih iger. Za potrebe te diplomske naloge se bomo osredotočili na igro Pong. V vsakem koraku je agentu na voljo 6 različnih akcij. Te smo še okrnili in uporabili le 3 najpomembnejše, in sicer akcija 1 (serviraj žogo), akcija 2 (premakni se gor) in akcija 3 (premakni se dol). Simulator v vsakem koraku prejme agentovo akcijo, jo izvede in kot izhod vrne zaslonsko sliko, ki predstavlja trenutno stanje igre, ter nagrado. Ker je podatkov v zaslonski sliki veliko, smo namesto nje za učenje nevronske mreže izvedli vpogled v pomnilnik igre, iz katerega smo prebrali položaj igralne žoge, ki je podana z dvema koordinatama x in y, ter položaj obeh igralcev, ki sta podana le s koordinato y. Imamo torej 4 vhodne parametre.

Slika 6 prikazuje zaslonsko sliko igre Pong.



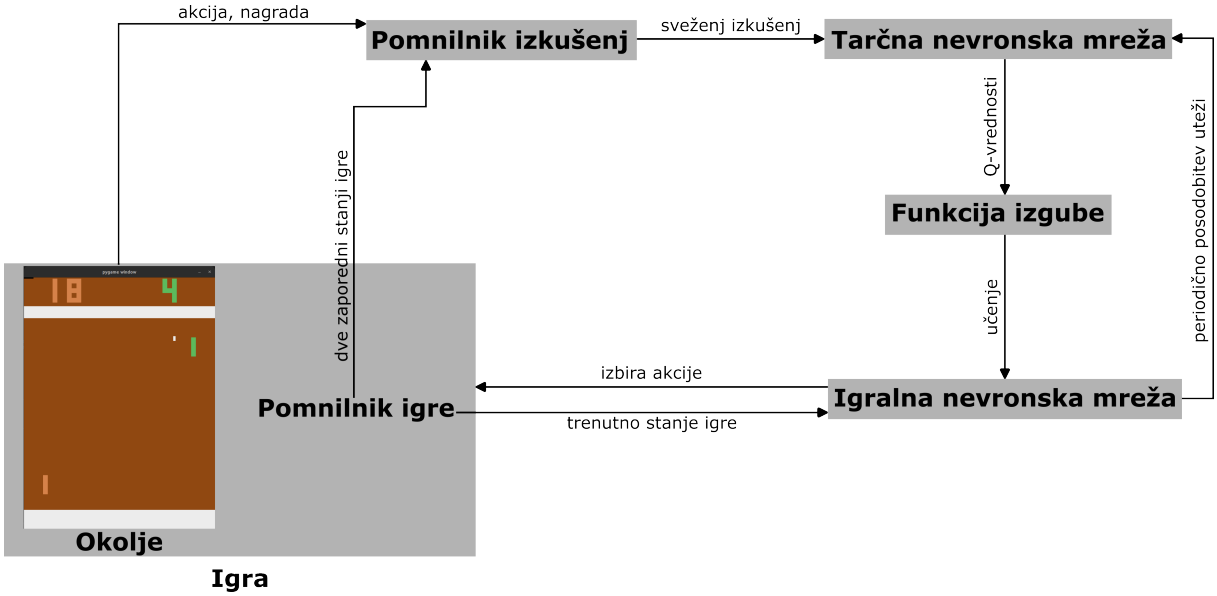
Slika 6: Zaslonska slika igre Pong

Na tej sliki lahko vidimo prvega igralca (oranžne barve) na levi, drugega igralca (zelene barve) na desni in igralno žogo (bele barve). Na vrhu lahko opazujemo tudi rezultat trenutne igre. V tem primeru levi igralec vodi z rezultatom 18 proti 4.

Ker smo v tej nalogi želeli izvesti nasprotniško učenje dveh agentov, smo potrebovali simulator, ki podpira igranje za 2 igralca, kar nam je omogočila knjižnica PettingZoo. Simulator dodeli nagrado le ob danem (vrednost nagrade 1) ali prejetem (vrednost nagrade -1) zadetku, kar pomeni, da je velika večina prejetih nagrad enaka 0. Ob razširitvi igre na dva igralca se je izkazalo, da je ob takšnem sistemu nagrajevanja učenje prepočasno, zato smo implementirali nov sistem dodeljevanja nagrad. Za dosežen zadetek smo agentu podelili 10 točk, za prejet zadetek -10 točk, za uspešen odboj žoge 1 točko in za vse ostalo 0 točk. S tem smo agenta nagradili tudi v primeru, ko je žogo odbil, kar v prejšnjem sistemu ni bilo zaznano.

3.2 GLOBOKO Q-UČENJE

V našem primeru je agent postavljen v okolje, kjer mora na podlagi štirih vhodnih podatkov — pozicija obeh igralcev in žoge — pravilno napovedati najboljšo naslednjo akcijo. V našem primeru agent lahko izbira med tremi akcijami. Te so: serviranje žoge po prejetem zadetku, premik loparja gor in premik loparja dol. Ker poskušamo dva igralca naučiti igrati igro enega proti drugemu, potrebujemo za vsakega izmed njih svojo instanco algoritma spodbujevalnega učenja. Da se učenje ne bi zataknilo v nekem lokalnem maksimumu, smo uporabili koncept algoritma DQN, ki namesto le ene nevronske mreže uporabi dve, in sicer igralno (angl. *policy*) in tarčno (angl. *target*) nevronsko mrežo. Igralna nevronska mreža je tista, na kateri se v vsakem koraku izvaja učenje. Tarčna nevronska mreža pa za igralno mrežo zaostaja določeno število korakov. S tem dosežemo stabilnejše učenje.



Slika 7: Arhitektura globokega spodbujevalnega učenja po vzoru algoritma DQN za učenje igre Pong

Na sliki 7 je prikazana shema poteka igre Pong in učenja tarčne in igralne nevronske mreže. V vsakem koraku se nova izkušnja shrani v pomnilnik izkušenj. Iz njega se potem izbere sveženj 32 naključnih izkušenj, ki se jih servira tarčni nevronski mreži. Ta napove Q-vrednosti, za vsako izmed njih. Potem se s pomočjo funkcije izgube to preda igralni nevronski mreži, ki na prejetih vrednostih izvede korak učenja. V vsaki iteraciji igre pa igralna nevronska mreža od okolja prejme tudi trenutno stanje igre, na podlagi katerega se mora odločiti za najboljšo akcijo.

3.3 ARHITEKTURA NEVRONSKE MREŽE

Izbira arhitekture umetne nevronske mreže za globoko spodbujevalno učenje je odvisna od kompleksnosti okolja in števila akcij, ki se jih mora agent naučiti. Če je okolje enostavno, je bolje izbrati enostavnejšo arhitekturo, saj se te lahko učijo hitreje. Po drugi strani pa se enostavna arhitektura ne more naučiti zahtevnejših strategij v kompleksnejših okoljih.

Pri izbiri ustrezne arhitekture oziroma velikosti vseh skritih plasti nevronske mreže smo imeli kar nekaj težav, saj smo začeli s premajhnimi skritimi plastmi. To je pripeljalo do tega, da se mreži nista naučili dovolj, da bi začeli uspešno odbijati žogo. Preizkusili smo več različnih konfiguracij in na koncu izbrali naslednjo:

1. `Vhodni sloj: 4 nevroni`
2. `Prvi skriti sloj: 256 nevronov`
3. `Drugi skriti sloj: 2048 nevronov`
4. `Izhodni sloj: 3 nevroni`

Velikost vhodne in izhodne plasti je določena z dimenzijo stanj in številom akcij. Velikost prve skrite plasti smo nastavili na 256 nevronov, druge pa na 2048 nevronov. To je omogočilo, da je arhitektura dovolj velika, da se po zadostnem številu učnih korakov mreži naučita odbijati žogo. Za izhodno plast smo uporabili linearno aktivacijsko funkcijo, na vseh ostalih pa funkcijo ReLU. Vse plasti nevronske mreže so polno povezane, kar pomeni, da je vsak nevron povezan z vsemi nevroni predhodnje plasti.

3.4 PREISKOVANJE PROSTORA STANJ

Na začetku so uteži v nevronski mreži inicializirane na naključne vrednosti, kar pomeni, da agent sledi naključni igralni strategiji, kar lahko vodi (in pogosto tudi vodi) k ponavljanju istega zaporedja akcij. Agent tako ne pridobiva novih izkušenj in se zato ne more ničesar naučiti. To rešimo tako, da vpeljemo naslednji princip. Najprej definirajmo pojem stopnje preiskovanja (angl. *exploration rate*) kot $\varepsilon \in [0, 1]$, kjer 0 pomeni popolno izkoriščanje, 1 pa popolno preiskovanje. Pred vsako izbiro akcije sistem izbere naključno realno število med 0 in 1. Če je naključno izbrano število med 0 in 1 večje ali enako vrednosti ε, agent izbere akcijo na podlagi trenutno naučene strategije, sicer pa izbere naključno akcijo. Vrednost ε na začetku nastavimo

na 1, kar pomeni, da agent na začetku izbira le naključne akcije, z vsako naslednjo epizodo pa vrednost ε nekoliko zmanjšamo. Tako se agent vedno bolj opira na naučeno igralno strategijo. Vrednost ε vedno obdržimo nekoliko nad 0, saj s tem v igro vnesemo nekaj variabilnosti, še posebej, če so začetni pogoji vedno enaki. Najnižja stopnja preiskovanja, ki smo jo uporabili tako med treniranjem kot med igranjem, je bila $\varepsilon = 0{,}1$. To pomeni, da je bilo vedno vsaj 10 % akcij izbranih naključno.

4 REZULTATI

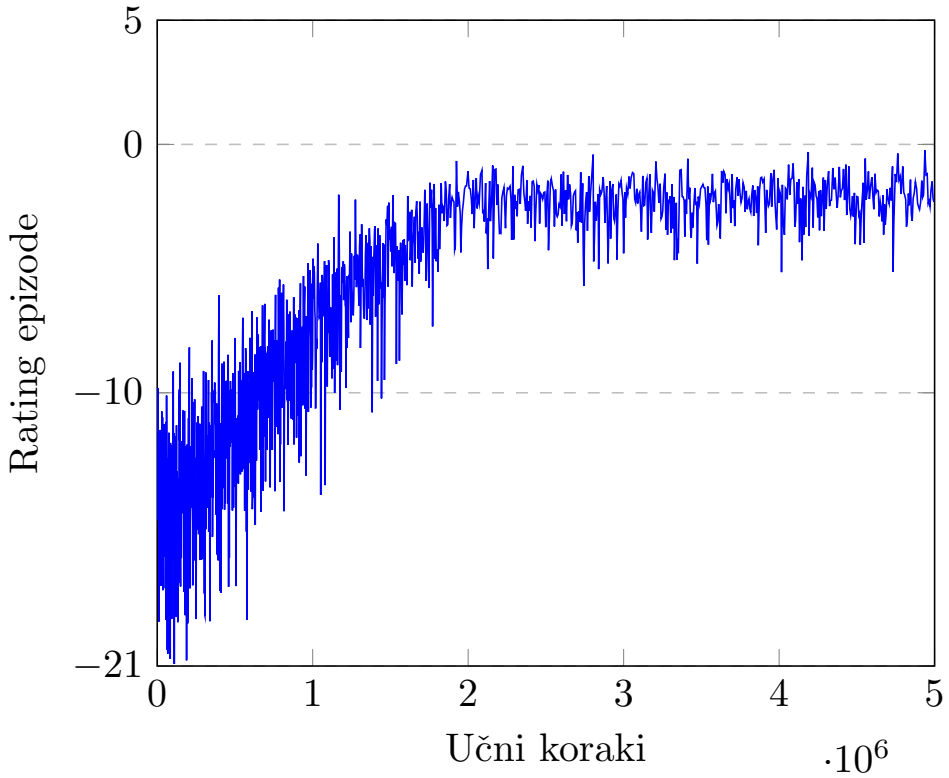
4.1 IGRA ZA ENEGA IGRALCA

Svojo implementacijo globokega spodbujevalnega učenja smo najprej preizkusili na igri Pong za enega igralca, pri čemer lopar drugega igralca upravlja algoritem, ki so ga sprogramirali avtorji igre Pong. Gre za preprost algoritem, ki predvsem sledi gibanju žogice. Tako smo se prepričali, da naš učni algoritem deluje pravilno. Uporabili smo sledečo arhitekturo nevronske mreže, ki se je v takšni igralni postavitvi izkazala najbolje:

1. `Vhodni sloj: 4 nevroni`
2. `Prvi skriti sloj: 64 nevronov`
3. `Drugi skriti sloj: 512 nevronov`
4. `Izhodni sloj: 3 nevroni`

Preiskusili smo tudi nevronske mreže z arhitekturo skritih plasti 16/128, 32/256 in 128/1024, vendar so te dosegale precej slabše rezultate. Eksperimentirali smo tudi z vrednostjo ε. Na koncu smo se odločili, da ε do 1 milijona učnih korakov pada linearno od 1 proti 0,1 ter tam ostane do konca učenja. Določili smo tudi število začetnih korakov, med katerimi agent še ne izvaja učenja, ampak izvaja naključne akcije in s tem polni pomnilnik izkušenj. Za to smo določili 50.000 začetnih korakov.

Učenje smo pustili teči 72 ur na strežniku s procesorjem Intel Xeon, ki ima frekvenco jedra 3,5GHz. V tem času je agent odigral 1325 epizod, kar je približno 5 milijonov učnih korakov.



Slika 8: Graf učenja igre Pong za enega igralca

Na sliki 8 je prikazan *rating* igranja igre Pong za enega igralca proti vgrajenem računalniškem igralcu. Rating predstavlja oceno uspešnosti agenta v posamezni epizodi. Izračunamo ga z uporabo naslednje enačbe:

$$rating = \frac{\text{nagrada epizode}}{\text{dolžina epizode}} \cdot 1000 \tag{4.1}$$

Na sliki 8 lahko opazimo, da se v prvih 2 milijonih učnih korakov nevronska mreža uči skoraj linearno, takoj ko doseže 2 milijona učnih korakov, pa se učenje upočasni in do konca ostaja približno na enakem nivoju. To pomeni, da je ta nevronska mreža konvergirala pri približno 2 milijona učnih korakih. Če bi želeli, da bi bila kovergenca kasnejša in pri višji vrednosti *ratinga*, bi lahko povečali arhitekturo skritih plasti in pustili nevronsko mrežo trenirati ponovno. Ker pa se v diplomskem delu osredotočamo na igranje igre za dva igralca, pa smo s tem želeli le preiskusiti, če postavljeno okolje deluje. Pri tem smo bili uspešni.

4.2 IGRA ZA DVA IGRALCA

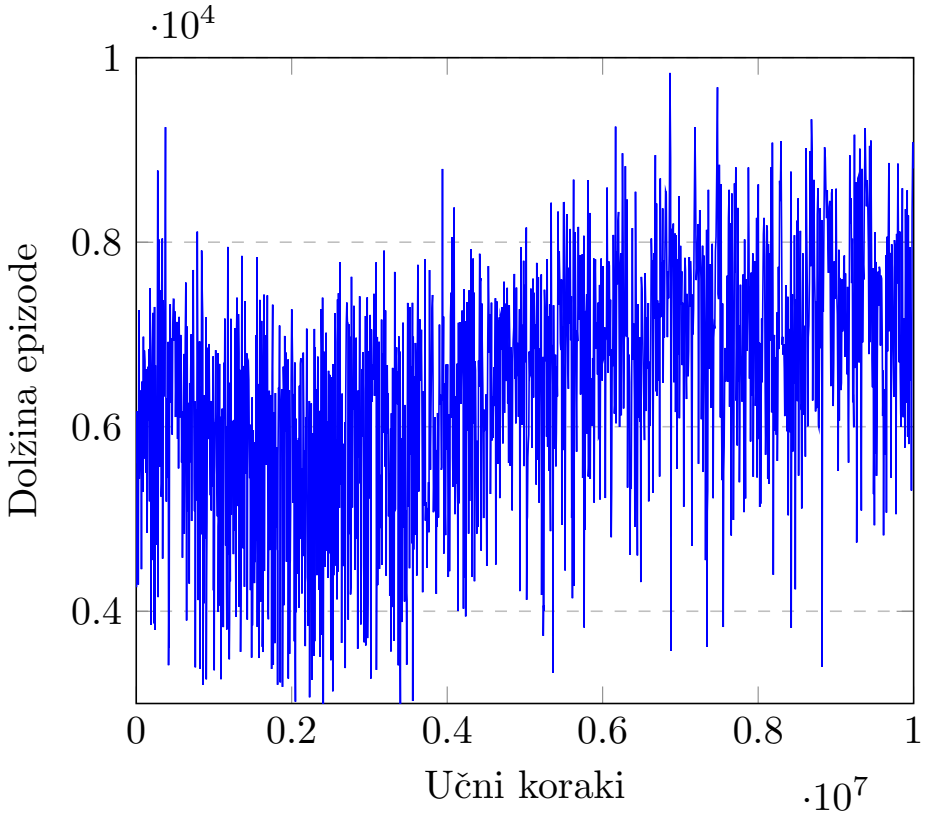
Ker knjižnica openAI Gymnasium podpira igranje igre za le enega igralca, smo za implementacijo okolja z dvema igralcema uporabili knjižnico PettingZoo [7]. Tudi tu smo preizkusili več različnih arhitektur nevronskih mrež in se na koncu odločili za sledečo, ki je bila od preizkušenih najuspešnejša:

1. Vhodni sloj: 4 nevroni

2. `Prvi skriti sloj: 256 nevronov`
3. `Drugi skriti sloj: 2048 nevronov`
4. `Izhodni sloj: 3 nevroni`

Skriti plasti sta v tem primeru precej večji kot pri igri za enega igralca, saj smo z dodanim igralcem prostor stanj bistveno povečali. Več nevronov tako omogoča oblikovanje kompleksnejših strategij igranja. Preizkusili smo tudi arhitekture z velikostjo skritih plasti 16/128, 32/256, 64/512, 128/1024 in 512/4096. Vse izmed njih so bile premajhne, razen zadnje, ki bi za učenje porabila več časa, kot pa smo ga imeli na voljo. Stopnjo preiskovanja ε smo nastavili enako kot pri igri za enega igralca, odločili pa smo se, da omejimo velikost pomnilnika na 1 milijon izkušenj. Ko se je pomnilnik zapolnil, je vsaka nova izkušnja prepisala najstarejšo, ki je bila trenutno v pomnilniku. To je omogočilo, da se je nevronska mreža vedno učila le na zadnjih milijon zapisih.

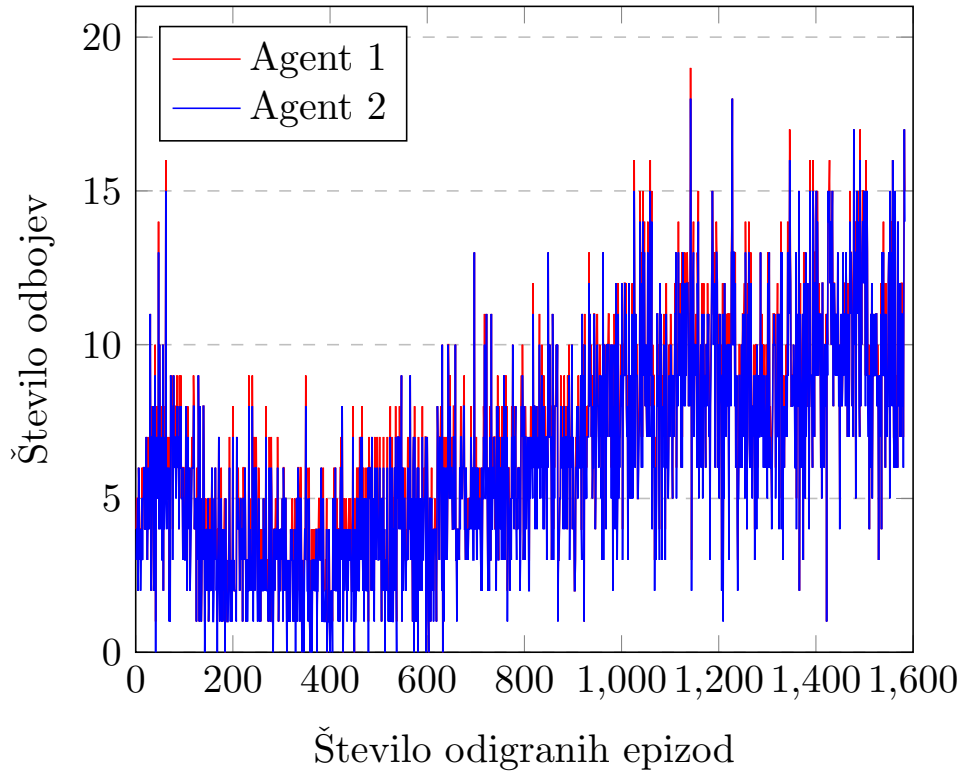
Algoritem smo pustili teči do 10 milijonov učnih korakov, in sicer na istem strežniku kot učenje za enega igralca. Učenje je trajalo približno en teden. V tem času sta agenta odigrala 1583 epizod.



Slika 9: Dolžina epizod glede na število korakov

Ker pri učenju dveh igralcev enega proti drugem ne moremo uporabiti enakega merila uspešnosti kot pri enoigralskem načinu, smo rezultate prikazali na dva drugačna načina. Kot prvo merilo smo izbrali dolžino posamezne epizode v odvistnosti od števila učnih korakov. Rezultat je prikazan na sliki 9. Opazimo lahko, da se dolžina epizode postopoma viša. To je

posledica dejstva, da dlje ko se agenta učita, bolje znata odbijati žogo, kar podaljša dolžino epizode (trajanje igre do prejetega ali danega zadetka). Kot drugo merilo uspešnosti igranja smo določili število odbojev žoge posameznega agenta v vsaki epizodi. Rezultati učenja po tem merilu pa so prikazani na sliki 10.

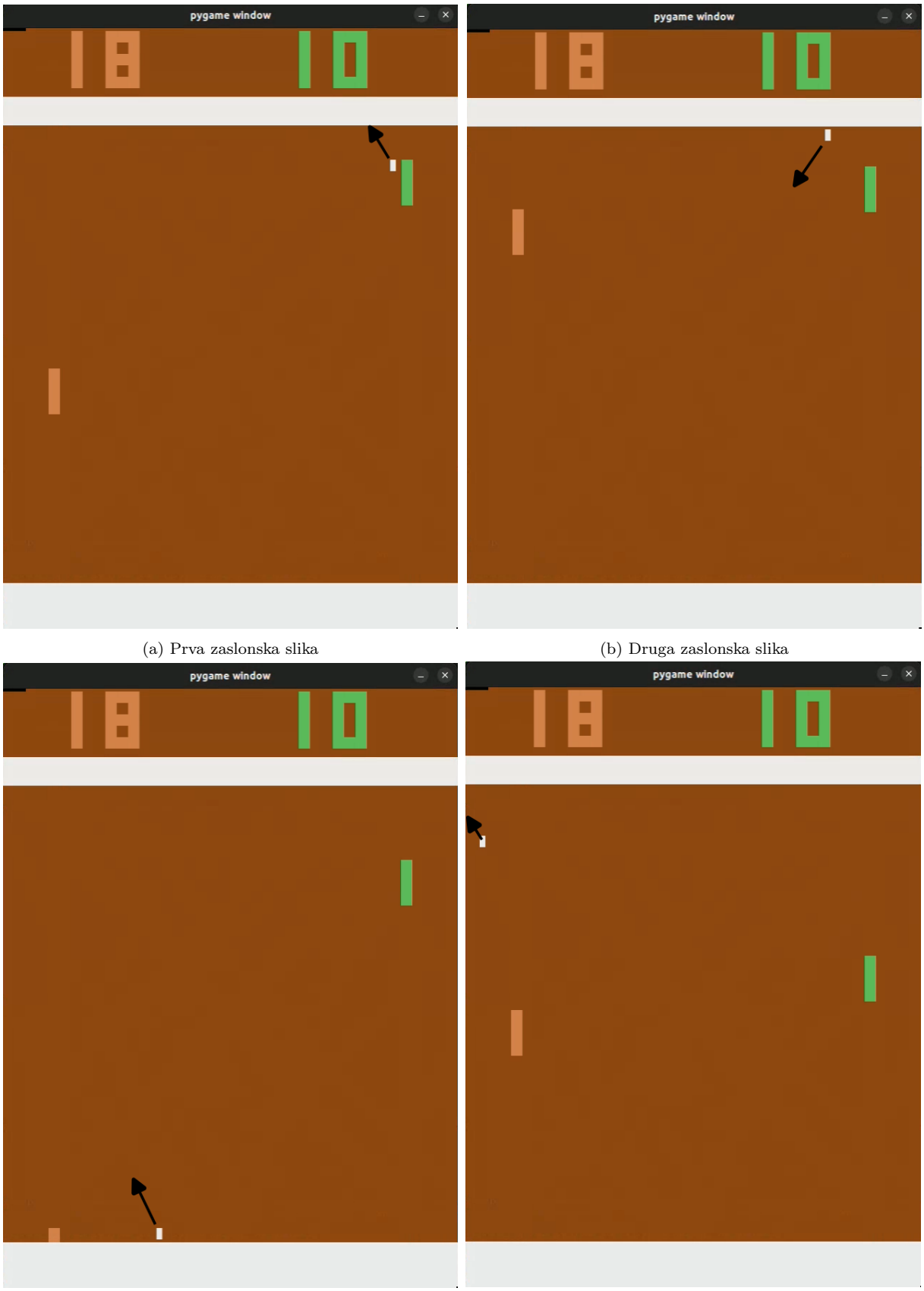


Slika 10: Število odbojev žoge glede na število odigranih epizod

Opazimo, da se agenta učita približno z enako hitrostjo, kar je pričakovano, saj oba začneta ob enakih začetnih pogojih. Lahko pa vidimo tudi, da ima agent 1 skozi celoten potek učenja nekaj prednosti pred agentom 2, kar je v tem primeru slučajno in posledica začetne sreče agenta 1.

Rezultate lahko opišemo tudi kvalitativno. Okolje PettingZoo nam omogoča, da lahko potek igre spremljamo v realnem času. To nam omogoča, da lahko razberemo strategije, ki sta se jih nevronski mreži naučili. Na začetku, ko je večina akcij, ki jih nevronski mreži izbereta, naključnih, igralca nimata razvite še nobene strategije. Ko pa izvedeta več učnih korakov, pa postane očitno, da vsaka izmed njiju želi z napadalno strategijo doseči zadetek. Ta strategija je taka, da ko se žoga približuje enemu izmed njiju, jo ta želi odbiti čim bolj z robom loparja, kar povzroči, da se žoga odbije s precej večjo hitrostjo, kot je priletela proti loparju. Ker je žoga odbita z enim izmed robov, je smer odboja vedno proti zgornjem oziroma spodnjem robu igralne površine. Od nje se potem tudi odbije, kar nasprotnika še dodatno zmede in posledično lažje prejme zadetek, saj žoge ne uspe odbiti.

Naučeno napadalno igralno strategijo najlažje ponazorimo z zaporedjem zaslonskih posnetkov, prikazanih na sliki 11.



(a) Prva zaslonska slika

(b) Druga zaslonska slika

(c) Tretja zaslonska slika

(d) Četrta zaslonska slika

Slika 11: Napadalna igralna strategija. Puščice prikazujejo smer gibanja žogice.

Desni igralec zelene barve na prvem posnetku 11a žogo odbije z zgornjim robom loparja,

kar mu omogoči, da žogo dobije z večjo močjo. Zato ta dobi pospešek in se odbije od roba igralne površine. Na slikah 11b in 11c lahko vidimo, da se je žoga najprej odbila od zgornjega roba igralne površine in potem še od spodnjega, kar je prvega igralca oranžne barve zelo zmedlo in ni vedel, kako naj se postavi, da bo lahko žogo odbil. Posledica tega je slika 11d, ki prikazuje dosežen zadetek. Prvi igralec je bil pri obrambi neuspešen in je prejel zadetek.

Predstavljeno napadalno strategijo sta razvila oba igralca. Treba je poudariti, da to ne uspeta izvesti prav z vsakim odbojem, temveč to storita nekajkrat med vsako odigrano igro. Če jima to ne uspe, prejmeta zadetek ali pa žogo odbijeta počasi in nasprotniku dopustita dovolj časa, da se pravilno postavi.

5 DISKUSIJA

V diplomskem delu smo z uporabo arkadnega učnega okolja (ALE) poskušali ugotoviti, kako dobro se lahko dva agenta globokega spodbujevalnega učenja naučita igrati igro Pong na simulaciji igralne konzole Atari 2600, eden proti drugemu. Prostor stanj dvo-igralske različice je bistveno večji od eno-igralske, na kateri je bil DQN demonstriran v preteklosti, zato smo uvedli nekatere prilagoditve — izločitev konvolucijskih plasti, ki smo jih nadomestili z ročno izluščenimi značilkami iz pomnilnika igralne konzole, ter uvedba pogostejše nagrade. Da bi preiskusili uspešnost svoje prilagojene implementacije globokega spodbujevalnega učenja, smo začeli z implementacijo igre za enega igralca. Algoritem smo pustili igrati 1325 iger proti vgrajenemu igralcu do 5 milijonov učnih korakov, kar je trajalo tri dni na osebnem računalniku. Nadaljevali smo z implementacijo dvoigralskega načina. Algoritmu smo pustili igrati 1583 epizod, kar je trajalo 10 milijonov učnih korakov in približno en teden učenja na istem osebnem računalniku.

Pri enoigralskem načinu so rezultati pokazali, da nevronska mreža konvergira pri 2 milijona učnih korakih. To se zgodi tik preden bi uspela premagati vgrajenega igralca, kar pomeni, da doseže približno enako sposobnost igranja. Če bi povečali arhitekturo skritih plasti te nevronske mreže, bi ta verjetno uspela vgrajenega igralca po zadostnem številu učnih korakov tudi premagati. Uspešnost učenja igre za dva igralca pa smo prikazali na dva nekoliko drugačna načina. Prvi je bil z merjenjem dolžine posamezne epizode. Ta se je višala s številom učnih korakov, kar je posledica uspešnega učenja obrambne strategije. Kot drugo merilo uspešnosti smo šteli število odbojev posamezne nevronske mreže po vsaki odigrani epizodi. Tudi tu so se vrednosti višale s številom učnih korakov, razvidno pa je postalo, da igralca napredujeta približnp enako hitro. V neki epizodi sta igralca dosegla kar 19 in 18 odbojev. Ta najdaljša opažena epizoda je trajala 9835 korakov.

Opazili smo, da sta igralca razvila tudi napadalno strategijo, ki smo jo opisali v poglavju 4. Ugotovili smo, da sta se igralca naučila, da je potrebno žogo odbiti kar se da z robom loparja, kar povzroči, da se žoga odbije pod ostrim kotom in s povečano hitrostjo. To nasprotnika zmede, saj ta v večini primerov ne zna dovolj hitro prestaviti loparja in preprečiti zadetka.

Pri implementaciji smo imeli največ težav z izbiro primerne arhitekture nevronskih mrež za dvoigralski način. Ker na začetku nismo predvidevali, da je učno okolje za dva igralca precej kompleksnejše od tistega za enega, smo izbirali arhitekture, podobne tistim za enega igralca. To je pripeljalo do tega, da pri učenju nismo bili uspešni in smo na ta način izgubili veliko časa, saj preverjanje vsake konfiguracije vzame veliko časa.

Z opisanimi rezultati smo zadovoljni, saj smo pokazali, da se tudi dva začetniška računalniška igralca uspeta naučiti igro Pong eden proti drugem. Dobljene rezultate bi lahko še izboljšali, če bi arhitekture skritih plasti nevronskih mrež še povečali, a bi se s tem čas učenja podaljšal preko procesnih zmogljivosti, ki smo jih imeli na voljo. To zato ostaja kot možnost za nadaljnje

delo. Zanimivo bi bilo tudi preiskusiti, kako dobro bi se naučena igralca iz dvo-igralskega okolja obnesla v eno-igralskem okolju. Zanimalo bi nas, če sta se naučila dovolj splošne strategije igranja, da bi lahko obvladala vgrajeni igralni algoritem. Še zanimivejše pa bi bilo namesto vgrajenega igralca nasproti igralcema iz dvoigralskega načina postaviti človeka. Izpeljali bi turnir, na katerem bi ljudje igrali proti naučenima agentoma. Na podlagi rezultatov bi ugotavljali, če sta naučena agenta dosegla ali celo presegla človeški nivo igranja igre.

6 LITERATURA

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[3] Ian Bicking. Python package index - pypi. `https://pypi.org/`, Datum dostopa: 3. 8. 2023.

[4] Francois Chollet et al. Keras, 2015. `https://github.com/fchollet/keras`, Datum dostopa: 3. 8. 2023.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv, 2013.

[6] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Jordan Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall Williams, and Yashas Lokesh. PettingZoo: Gym for multi-agent reinforcement learning. `https://github.com/Farama-Foundation/PettingZoo`, Datum dostopa: 3. 8. 2023.

[8] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. `https://zenodo.org/record/8127025`, Datum dostopa: 3. 8. 2023.