SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1669

Primjena postupaka umjetne inteligencije za izradu računalnog igrača u video igri

Leon Šamec

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 15. ožujka 2018.

DIPLOMSKI ZADATAK br. 1669

Pristupnik: **Leon Šamec (0036476501)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Primjena postupaka umjetne inteligencije za izradu računalnog igrača u video igri**

Opis zadatka:

Proučiti područje primjene umjetne inteligencije u računalnim igrama za jednog ili više igrača. Opisati tehnike umjetne inteligencije koje se koriste za oblikovanje računalnih modela igrača. Oblikovati model učenja za diskretnu igru za dva igrača na temelju trenutnog stanja i zadanih pravila. Primijeniti postupke podržanog učenja i optimizacije uz pomoć evolucijskog računarstva za oblikovanje računalnog igrača. Ispitati učinkovitost dobivenih igrača u ovisnosti o pravilima igre, algoritmima učenja i vještini protivnika. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 29. lipnja 2018.

Mentor:

Prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Dječovođa:

Doc. dr. sc. Tomislav Hrkać

Zahvaljujem mentoru prof. dr. sc. Domagoju Jakoboviću na slobodi u izboru teme i podršci tijekom izrade ovog rada.

Sadržaj

Uvod

Pod umjetnu inteligenciju spada skup raznih algoritama koji imitiraju način na koji se uči (npr. podržano učenje) ili imitiraju neke druge prirodne fenomene (npr. evolucijski algoritmi). Primjena postupaka umjetne inteligencije je raznolika: od razvoja raznih proizvoda (npr. njihova optimizacija) do izrade računalnog igrača u video igri (npr. planiranje, podržano učenje). Kako računalna moć sve više raste, algoritmima umjetne inteligencije mogu se riješiti sve složeniji problemi. U sklopu ovog rada primijenit će se postupci umjetne inteligencije za izradu računalnog igrača u video igri.

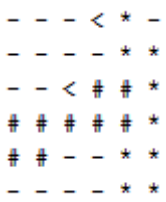
Podržano učenje imitira bihevioralnu psihologiju u kojoj se tvrdi da se ponašanje modificira pomoću davanja nagrade ili kazne. Ako se želi poticati neko ponašanje, onda se nagrađuje, a ako se želi spriječiti neko ponašanje, onda se kažnjava.

U 1. poglavlju opisuje se video igra, igrači izrađeni tehnikama planiranja i načini na koje će igrači razvijeni tehnikama opisanim u idućim poglavljima "vidjeti" igru – stanja igrača. U 2. poglavlju opisuje se podržano učenje i njegova implementacija. U 3. poglavlju opisuje se NEAT i njegova implementacija. U 4. poglavlju uspoređuju se igrači razvijeni na različite načine.

1. Igra

1.1. Opis

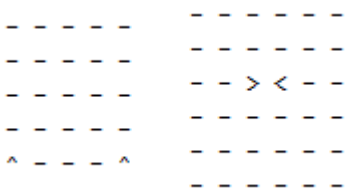
U programskom jeziku Python implementirana je igra po uzoru na postojeću igru "Light Riders″ [1]. Igra "Light Riders″ napravljena je u sklopu natjecanja u izradi računalnih igrača. To je igra na ploči za dva igrača u kojoj je jedan igrač pobjednik, a drugi gubitnik (igra sa sumom nula). U njoj svaki igrač kontrolira jednog agenta. Ploča se sastoji od $size_x \; x \; size_y$ polja. U odnosu na orijentaciju igračevog agenta, mogući su pomaci na polja lijevo, desno i ravno. Orijentacija može biti lijevo, desno, gore i dolje. Kada igrač napravi potez, na polju na kojem je prethodno bio stvori se "zid". Ako se pojedini igrač zabije u "zid" ili izađe s ploče, on je izgubio. No, ako se oba igrača u istom potezu zabiju u "zid", izađu s ploče ili stanu na isto polje, onda je izjednačeno. Primjer jednog odigravanja igre može se vidjeti na slici (Slika 1.1). U implementaciji igre oba igrača istovremeno vuku poteze, igra je diskretna.



Slika 1.1 Primjer odigravanja igre

Implementirani su različiti načini kako se na početku postavljaju igrači na ploču. Jedan je način u kojem se oni postavljaju u suprotne kutove: dolje lijevo i dolje desno (*Slika 1.2*). Oba igrača su orijentirana prema gore. Drugi je način u kojem se oni postavljaju pseudo-nasumično. Opis pseudo-nasumičnog početnog postavljanja igrača slijedi. Kako se ploča sastoji od polja $\left(\{0,1,\dots,\text{size}_x - 1\}, \{0,1,\dots,\text{size}_y - 1\} \right)$, prvi se igrač postavlja na poziciju $\left(\left\{1,\dots,\frac{\text{size}_x}{2} - 1\right\}, \{1,\dots,\text{size}_y - 2\} \right)$ (iz skupova u {} se odabire jedan broj). Početna orijentacija igrača je također nasumična. Drugi se igrač postavlja zrcalno preko

sredine mape u odnosu na prvog igrača. Parametri size_x i size_y ovdje trebaju biti parni brojevi. Pseudo-nasumično početno postavljanje igrača preuzeto je od igre "Light Riders". Primjer pseudo-nasumičnog početnog postavljanja igrača prikazan je na slici u nastavku (Slika 1.2).



Slika 1.2 Primjer početnih pozicija: na suprotnim stranama (lijevo), pseudo-nasumično (desno)

1.2. Implementacija

Igra je implementirana u razredu "Board". Prilikom postavljanja igre moguće je zadati širinu ($size_x$) i dužinu ($size_y$) ploče na kojoj će se igrati. Nakon toga se postavljaju igrači koji mogu biti iz bilo kojeg razreda koji nasljeđuje razred "Player". Svaki razred igrača koji naslijedi razred "Player" mora implementirati metodu "getDirection()" pomoću koje se u svakom potezu bira akcija. Ta metoda može biti proizvoljne složenosti. Može se pri izračunu akcije koristiti stanje ploče (korištenjem razreda "Board"), a igrač može biti i čovjek koji akcije unosi preko tipkovnice. Nakon postavljanja vrste igrača postavljaju se njihove "figurice" na početna mjesta na ploči. To se obavlja po jednom od dva načina opisana u prethodnom poglavlju. Tada igra počinje i odigrava se do kraja izvršavanjem metode "play()" objekta razreda "Board" u kojemu je igra implementirana i postavljena. Moguće je parametrom u pozivu metode "play()" aktivirati ispis na ekranu ($true|false$). Tako se prije svakog koraka igre u tekstualnom obliku na ispisnom prostoru prikazuje trenutno stanje ploče. Nakon što igra završi metoda "play()" vraća brojeve 1, 2 ili 3. Vraća se broj 1 ako je igrač-1 pobijedio, broj 2 ako je igrač-2 pobijedio ili broj 3 ako je izjednačeno.

1.3. Planirajući igrači

Prvi tip računalnog igrača nije razvijen u procesu optimizacije. Nije napravljen tako da se u procesu njegova razvoja treba odigrati veliki broj igara. Njegova strategija je unaprijed smišljena, tj. isplanirana, od strane programera. Napravljena su dva tipa takvih igrača. Jedan tip je nazvan "random player", a drugi "look player". "Random player" je potpuno trivijalan. Svaki potez izvrši nasumično odabranu akciju. "Look player" je nešto složeniji. Parametriziran je jednom varijablom *x*. Svaki potez on pogleda trenutno stanje ploče. Potom odabire onu akciju s kojom može odigrati barem *x* budućih poteza (tako da se ne zabije u "zid" ili izađe s ploče), ali samo prema trenutnom stanju ploče. Ta akcija se traži pretragom u dubinu (engl. *depth-first search*). Stablo koje se pretražuje ima u korijenu trenutno stanje, a čvor-dijete svakog čvora-roditelja je stanje u koje se došlo iz čvora-roditelja nekom akcijom igrača (protivnik ovdje ne vrši akcije). Stanje nije u stablu ako se u njemu igrač zabio u "zid" ili izašao s ploče. Pokušava se doći do dubine *x* tog stabla. Prva akcija u lancu koji kreće od korijena i ide do čvora na dubini *x* je ta tražena. Akcije se izvršavaju nasumično - skup akcija za izvršavanje je uvijek izmiješan (engl. *shuffled*). Slijedi da ako postoji više traženih akcija, odabire se nasumično jedna od njih. Ako ne postoji akcija s kojom se može odigrati barem *x* budućih poteza, onda se odabire ona s kojom se može odigrati najviše. To se ostvaruje tako da se uvijek sprema prva akcija lanca u kojemu se odigrao najveći broj poteza. I da se isprazni stog prije dolaska do dubine x, akcija je spremljena. Treba uočiti da se akcije koje će odigrati protivnik u budućim potezima ne uzimaju u obzir. Može se uočiti da "look player" s $x = 1$ nasumično bira između poteza s kojima se neće zabiti u "zid" ili izaći s ploče, ako takvi potezi postoje. Otprilike se može reći da što je *x* veći, to je "look player" bolji.

1.4. Ostvarenje stanja

Stanje je relativno u odnosu na igrača, dakle igrač-1 i igrač-2 imat će drugačije stanje s istim stanjem ploče. To je analogno tome da je igra drugačija ako se gleda iz perspektive jednog ili drugog igrača. Na primjer, ako jedan igrač pobjeđuje, drugi gubi. Na svim mjestima gdje se upotrebljava riječ stanje u kontekstu učenja zapravo misli na stanje igrača. Stanje je ostvareno kao binarni vektor (vektor jedinica i nula) na tri načina. Prvim

načinom ostvarena je potpuna informacija o igri u svakom potezu. Drugim i trećim načinom ostvarena je reducirana informacija. Razlog za reduciranje informacije o igri je brže učenje. Smatra se da s resursima uloženim u rad nije ostvarivo potpuno iskoristiti potencijal potpune informacije.

Stanje s potpunom informacijom ostvareno je kako slijedi. Prvih $size_x \; x \; size_y$ (širina puta dužina ploče) bitova su 1 ako je na tom mjestu „zid“ koji je stvorio igrač (ne protivnik). Idućih $(size_x + 2) \; x \; (size_y + 2)$ bitova pokazuje poziciju igrača. Od tih bitova samo je jedan 1. Dodatak + 2 je prisutan kako bi se detektiralo na kojem je mjestu igrač izašao s ploče. Može se to gledati kao obrub oko ploče. Iduća 4 bita pokazuju koja je orijentacija igrača (lijevo, desno, gore, dolje). Od njih samo jedan može biti 1, a to je onaj koji odgovara trenutnoj orijentaciji igrača. Svi dosad opisani bitovi se dupliciraju, no sada se gleda iz perspektive protivnika.

Stanje s reduciranom informacijom ostvareno je kako slijedi. Budući da je ostvareno na dva načina, prvo će se opisati prvi način. U prvom načinu se promatraju samo polja oko igrača. Parametriziran je jednom varijablom *howFar* kojom se varira koliko se polja u jednom smjeru od igrača gleda. Ono što se promatra je kvadrat u kojemu se igrač nalazi u sredini. Dužina stranice tog kvadrata je $2 * howFar + 1$, gdje taj + 1 postoji zbog toga što igrač zauzima jedno polje. Bitovi koji predstavljaju polja u tom kvadratu su 0 ako je polje prazno. Oni su 1 ako je na polju "zid", igračev ili protivnikov, što uključuje trenutne igračeve i protivnikove pozicije (one se mogu promatrati kao "zid"). Također su 1 na poljima u kvadratu koja obuhvaćaju područje izvan ploče. Iduća 4 bita ista su kao i ona koja prikazuju orijentaciju igrača u potpunoj informaciji. Drugi način reducirane informacije proširenje je prvog načina. Uz bitove prvog načina dodaju se bitovi kvadrata koji obuhvaća ista polja, a u kojem su svi bitovi 0 osim onog koji odgovara polju na kojem je protivnik. Moguće je da protivnik nije u tom kvadratu, onda su mu svi bitovi 0.

2. Podržano učenje

2.1. Opis problema

Podržano učenje (engl. *reinforcement learning*) jedno je od tri tipa strojnog učenja, uz nadzirano učenje (engl. *supervised learning*) i nenadzirano učenje (engl. *unsupervised learning*). Podržano učenje se može opisati kako slijedi. U nekoj okolini postoji agent koji izvršava akcije i tako potencijalno mijenja stanje okoline, a usput potencijalno dobiva nagradu od okoline. Cilj mu je dobiti što veću ukupnu nagradu do kraja njegove interakcije s okolinom (nagrade koje dobiva nakon svojih akcija se zbrajaju). Agent ima neku informaciju o okolini i ona se naziva stanje. Kako agent izvršava akcije potencijalno se mijenja i stanje, budući da agent može djelovati na okolinu, a time i na stanje. Stanju se može pridodati njegova vrijednost (engl. *value*). Vrijednost stanja - $V(s)$ označava očekivanu ukupnu nagradu od tog stanja pa do kraja agentove interakcije s okolinom. Ovdje se već može uočiti sljedeće. Kako bi se potencijalno dobro izračunala vrijednost stanja (u definiciji vrijednosti stanja je očekivanje), agent treba proći kroz više epizoda, a epizoda je sva interakcija između agenta i okoline od početnog do završnog stanja. Nekada možda nema završnog stanja, ali o tome u ovom radu dalje neće biti riječi. Od vrijednosti stanja, za kontrolu agenta, bitnija je Q-vrijednost (engl. *Q-value*). Q-vrijednost - $Q(s, a)$ je ukupna očekivana nagrada do kraja agentove interakcije s okolinom nakon što on u određenom stanju odabere određenu akciju. Vidjet će se kasnije da je dobra procjena Q-vrijednosti ključna za dobivanje dobrog ponašanja agenta.

Problem koji će se rješavati neće imati poznate vjerojatnosti prijelaza iz stanja X u stanje Y s nagradom Z kada se u stanju X izvrši akcija A. Te vjerojatnosti s pripadnim stanjima, akcijama i nagradama znane su i kao Markovljev proces odlučivanja (engl. *Markov decision process*). Problem nepoznatih vjerojatnosti riješit će se prolascima kroz epizode tj. uzorkovanjem. Taj način zove se podržano učenje bez modela (engl. *model-free*).

Većina informacija o podržanom učenju iz ovog rada, kao i dodatne informacije, mogu se pronaći u knjizi *Sutton, Barto* [2] i predavanjima *Davida Silvera* [3].

2.2. Smanjenje nagrada

Kako se među nagradama koje agent dobiva više cijene one koje su dobivene prije (gleda se iz nekog stanja), uvodi se smanjenje (engl. *discount*) onih nagrada koje su dobivene kasnije (sama funkcija koja računa nagradu ne ovisi o rednom broju poteza pa treba uvesti zasebno smanjenje). Ovo je analogno tvrdnji "Novac danas vrijedi više od novca u budućnosti" (osnovni princip financija). Što su nagrade dobivene kasnije, to je veće smanjenje (1). Ukupna nagrada računa se zbrajanjem svih tih smanjenih nagrada. Parametar kojim se određuje intenzitet favoriziranja nagrada koje su dobivene prije je γ. To je broj između 0 i 1. Ako je $\gamma = 0$, onda se uopće ne gleda buduća nagrada, kao da ne postoji. Ako je $\gamma = 1$, sve nagrade, i trenutna i buduće, imaju jednaku vrijednost. Obično se γ postavlja između te dvije krajnosti, na broj od oko 0.9. Tako buduća nagrada ima velik utjecaj, no u beskonačnosti je 0.

$$R_{sum} = R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots + \gamma^n R_n \quad (1)$$

2.3. Politika

Način na koji agent odabire akciju zove se politika (engl. *policy*). Jedan tip politike može biti da se odabere akcija koja u trenutnom stanju ima najveću Q-vrijednost, to je takozvana pohlepna politika (engl. *greedy policy*). Drugi tip politike bi mogao biti da se s vjerojatnostima koje su proporcionalne Q-vrijednostima vjerojatnosno odabere akcija. Dakle, kako bi se dobilo dobro ponašanje agenta, treba dobro odrediti Q-vrijednosti.

Postoji određena mana pohlepne politike, a ona se može uočiti u igrama poput "kamen-škare-papir". Naime, kada se koristi pohlepna politika, akcija koja se odabire u određenom stanju je uvijek ista i odgovara najvećoj Q-vrijednosti za to stanje (ova igra zapravo i nema stanje, no može se reći da ima samo jedno stanje). Recimo da ta akcija znači igrati "kamen". Kada bi protivnik u igri "kamen-škare-papir" to znao, on bi uvijek odigrao "papir". To je veliki problem za pohlepnu politiku jer igrač koji ju slijedi bi u tom

slučaju uvijek izgubio. Taj se problem može riješiti primjerice tako da igrač vjerojatnosno odabere akciju. Ovaj problem je vezan s prirodom Nashove ravnoteže (engl. *Nash equilibrium*) igre koja se odigrava. Nashova ravnoteža je u tim igrama mješovita strategija, tj. u njoj ne postoji akcija koja bi se s vjerojatnošću 1 odigrala. U nastavku je opis toga što je Nashova ravnoteža.

Nashova ravnoteža u igri s 2 igrača sa samo jednim korakom može se opisati kako slijedi. Prvo, igrač-1 izvrši akciju A. Zatim, igrač-2 izvrši akciju B znajući koju je akciju igrač-1 izvršio. Igrač-1 znajući koju je akciju igrač-2 izvršio, ne bi htio mijenjati svoju početnu akciju. U tom slučaju niti jedan igrač ne bi htio mijenjati svoju akciju i to bi bila Nashova ravnoteža. Pretpostavka je da oba igrača pri izboru akcija žele maksimizirati svoj dobitak. Nashova ravnoteža se može opisati i kao strategija jedinke u nekoj populaciji koju kad primjenjuju sve jedinke u toj populaciji (sve jedinke igraju istu igru), niti jedna jedinka nema koristi od devijacije od te strategije [4]. U igri "kamen-škare-papir" Nashova ravnoteža je mješovita strategija - izabranje akcije vjerojatnosno, pri čemu je vjerojatnost svake akcije 0.33.

U ovoj se igri se u svakom potezu može odigrati svaka akcija (lijevo, ravno, desno). Kad to ne bi bio slučaj, bila bi potrebna posebna modifikacija pohlepne politike koja bira akciju s najvećom Q-vrijednosti samo između onih mogućih. Takav slučaj bio bi u igri "križić-kružić" gdje na početku postoji 9 mogućih akcija, a sa svakim potezom se broj mogućih akcija smanjuje za 1 (kako se ploča popunjava "križićima" i "kružićima").

2.4. Računanje Q-vrijednosti

Q-vrijednosti se određuju iterativno pomoću sljedeće jednakosti (2).

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha(q_t - Q(S_t, A_t)) \\ Q(S_t, A_t) &\leftarrow (1-\alpha)Q(S_t, A_t) + \alpha\, q_t \end{aligned} \tag{2}$$

U prethodnoj jednakosti S_t označava promatrano stanje, A_t akciju koja se izršila u tom stanju, q_t označava cilj (engl. *tagret*) od tog para stanje-akcija (više u poglavlju 2.4.2) i parametar α stopu učenja.

Ako se analizira prethodna jednakost (2) može se uočiti da je to eksponencijalni putujući prosjek - EMA (engl. *exponential moving average*) [6]. On se koristi jer se teži dobiti očekivanje Q-vrijednosti kojemu bi prosjek bio jednak kad bi bilo beskonačno uzoraka (uzorak je ovdje q_t). S EMA zapravo se ne računa prosjek po definiciji. S njime se stavlja naglasak na nove uzorke, a umanjuje utjecaj starih (u prosjeku po definiciji svi uzorci imaju jednaku težinu). Pomoću parametra α određuje se koliku će težinu imati novi uzorci naspram starih. Veći α znači veći utjecaj novih uzoraka tj. veće zanemarivanje starih. Prednost EMA nad običnim prosjekom u podržanom učenju je ta što stariji ishodi, kada se ponašanje agenta unaprijedi, imaju smanjenu relevantnost. Na primjer, gubitak partije kada je ponašanje agenta bilo nasumično trebao bi se zanemariti. U EMA se ne treba spremati broj uzoraka od kojih se računa "prosjek", što je još jedna prednost.

2.4.1. Promjena ponašanja agenta

Bitno je uočiti da se ponašanje agenta stalno mijenja tijekom interakcije s okolinom. Svaki puta kada se promijeni funkcija koja računa Q-vrijednosti (a ona se mijenja tijekom interakcije agenta s okolinom), koristeći nad njom, na primjer, pohlepnu politiku, potencijalno se mijenja i ponašanje. Tvrdi se da korištenje pohlepne politike uvijek nad trenutnom funkcijom Q-vrijednosti vodi u neki optimum [3]. Najvjerojatnije je to neki lokalni optimum. Postoje načini da se ide barem u bolji lokalni optimum, a oni su opisani kasnije. Ti načini se zajedno zovu istraživanje.

2.4.2. Ciljevi

Uzorci od kojih se računa "prosjek" su ciljevi (engl. *targets*, q_t). Suštinski postoje dva načina na koja se može odrediti cilj. Jedan se zove MC (engl. *Monte-Carlo*), a drugi TD (engl. *Temporal-Difference*). Kod MC-ja epizoda se odigrava do kraja i za svaki par stanje-akcija cilj postaju zbrojene nagrade od tog para stanje-akcija do kraja epizode. TD je

cijeli spektar ciljeva kojima je zajedničko da se trenutna Q-vrijednost računa u odnosu na buduću Q-vrijednost. Kod TD-1 cilj postaje nagrada koja se dobije nakon što ste u trenutnom stanju izvrši akcija po zadanoj politici zbrojena s Q-vrijednosti novog para stanje-akcija. Stanje iz tog novog para je ono u kojemu se agent nađe nakon prethodno izvršene akcije. Akcija iz tog novog para je ona koja bi se u novom stanju izabrala po politici koja se slijedi (engl. *On-Policy*), iako se može odabrati i po drugačijoj politici (engl. *Off-Policy*). U nastavku su prikazani MC i TD načini računanja cilja (3),(4),(5). TD-1-On-Policy se još zove i *Sarsa*. U svim budućim nagradama i Q-vrijednostima postoji smanjenje γ. Može se uočiti da kod TD epizoda ne mora završiti kako bi se osvježila Q-vrijednost, dok kod MC-a mora.

TD-On-Policy ciljevi

$$\text{n} = 1, \qquad (Sarsa) \quad q_t^1 = R_{t+1} + \gamma Q(S_{t+1}, a) \tag{3}$$

$$\text{n} = 2, \qquad q_t^2 = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, a) \tag{4}$$

...

MC cilj

$$\text{n} = \infty, \qquad q_t^\infty = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T \tag{5}$$

Kada se koristi jedan vrlo zastupljen TD-1-Off-Policy cilj (6) u kojemu se akcija iz novog para (dakle ne ona koja se bira u trenutnom stanju, nego u idućem) bira pohlepnom politikom, a ne politikom koja se slijedi (vidjet će se kasnije da politika koja se slijedi uglavnom uključuje istraživanje), onda se takvo učenje zove Q-učenje (engl. *Q-learning*).

$$(Q - učenje) \quad q_t^1 = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \tag{6}$$

Kako se u TD metodama trenutna Q-vrijednost računa u odnosu na buduću Q-vrijednost, problem koji ostaje za riješiti je kako računati Q-vrijednost u završnom stanju,

kada više nema buduće Q-vrijednosti. U završnom stanju se radi modifikacija formule i uzima se u obzir samo nagrada koja se dobije u tom stanju.

2.5. Istraživanje

Kada bi agent uvijek slijedio pohlepnu politiku, završio bi u nekom rješenju koje bi moglo biti bolje. Ostvarenje tog poboljšanja bi bilo kad bi agent istraživao tj. ne uvijek slijedio pohlepnu politiku. Kad agent istražuje, lako je moguće da se otkrije neki put do završnog stanja koji daje veću ukupnu nagradu od one koja se dobiva kad se s trenutnim Q-vrijednostima slijedi pohlepna politika. Cijela bit istraživanja je u tome da se poboljšaju procjene Q-vrijednosti. Ako je u istraživanju otkriven put koji je bolji od onog dobivenog slijeđenjem pohlepne politike, Q-vrijednosti nisu dobro procijenjene. S dobrom procjenom Q-vrijednosti na kraju se onda može slijediti pohlepna politika.

Postoji više načina na koje se ostvaruje istraživanje. Jedan od njih je ε-*pohlepno* (*ε-greedy)* [3]. U njemu se dodaje šum pohlepnoj politici. S vjerojatnošću ε bira se nasumična akcija, a s vjerojatnošću 1 – ε slijedi se pohlepna politika. Kako se učenje agenta odvija kroz mnogo epizoda, ε se bira tako da je veći u prvim epizodama, a kasnije se smanjuje. Takav raspored ε odgovara tome da je u početku lošija procjena Q-vrijednosti nego kasnije. Drugi način istraživanja, onaj koji se koristio u ovom radu, naziva se *Boltzmannov pristup* (engl. *Boltzmann approach*) [7]. U Boltzmannovom pristupu vjerojatnost odabiranja pojedine akcije je veća time što je veća Q-vrijednost za tu akciju. U prvim epizodama se razdioba vjerojatnosti modificira da bude što bliža uniformnoj, a kasnije teži onakvoj kakvi su odnosi među Q-vrijednostima. Za izračun vjerojatnosti iz Q-vrijednosti se koristi *softmax* funkcija (7) s parametrom temperatura (τ) koji regulira odstupanje od uniformne razdiobe. Temperatura se smanjuje kako prolaze epizode. Nakon svake epizode temperatura se umanji za $\frac{početna\ temperatura}{ukupan\ broj\ epizoda}$ i ne spušta se ispod 1. Veća početna temperatura (τ_0) znači da će u cijelom procesu učenja, uspoređujući epizodu po epizodu, odabir akcije biti uniformniji, a ne proporcionalniji Q-vrijednostima. Boltzmannov pristup je odabran radi toga što u određenom radu pokazuje bolje rezultate od ε-pohlepnog [7]. Naime, oba načina istraživanja imaju nedostatke. Glavni je taj što niti jedan od ova dva

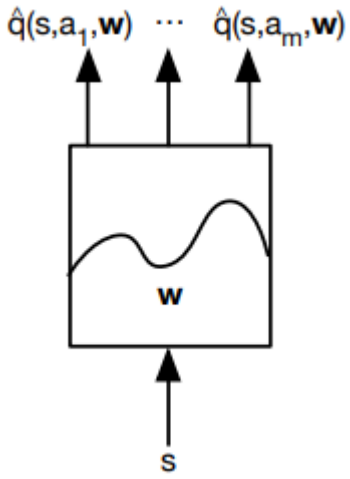
način ne uzima u obzir neizvjesnost procjene Q-vrijednosti, na primjer, broj uzoraka cilja od kojih je procijenjena Q-vrijednost. Neizvjesnost bi trebala biti što manja.

$$P(a' \mid S) = \frac{\exp(\frac{Q(S,a')}{\tau})}{\sum_{i=1}^{n_{action}} \exp(\frac{Q(S,a_i)}{\tau})} \quad (7)$$

Autor smatra da kada bi se podržanim učenjem izrađivalo računalnog igrača za igru u kojoj je Nashova ravnoteža mješovita strategija, postupak istraživanja koji bi bilo dobro koristiti je Boltzmanov pristup. Ne bi bili adekvatni načini istraživanja koji teže korištenju pohlepne politike, poput ε-pohlepnog, budući da je vjerojatnosni odabir akcija nužan ako se u takvoj igri želi doći do Nashove ravnoteže.

2.6. Funkcija aproksimacije

Q-vrijednosti se mogu spremati u "lookup" tablici u kojoj je svaka ćelija određena stanjem i akcijom. Naime, kada broj stanja poraste iznad nekog reda veličine, takav način spremanja Q-vrijednosti postane neizvediv. Potencijalno rješenje tog problema je zamjena "lookup" tablice funkcijom aproksimacije. Funkcija aproksimacije Q-vrijednosti na ulazu prima stanje i akciju, a na izlazu daje Q-vrijednost. Naime, zbog lakše implementacije koristit će se malo drugačiji oblik funkcije aproksimacije. Ona će na ulazu primati stanje, a na izlazu će davati Q-vrijednosti za svaku moguću akciju (Slika 2.1). Funkcija aproksimacije, osim što zauzima manje memorije, može generalizirati za neviđena stanja.



Slika 2.1 Prikaz funkcije aproksimacije Q-vrijednosti parametrizirane s ***w*** *(učeći parametri) [3]*

Funkcija aproksimacije u ovom radu realizirana je neuronskom mrežom. Neuronska mreža je pogodna zato što ima definiran gradijent u svim točkama. Kad funkcija ima definiran gradijent u svim točkama, može se koristiti algoritam gradijentnog spusta za traženje njezinog optimuma, a on je pogodan u ovom problemu. Nije korištena linearna regresija zato što se da smatra je potrebniji veći kapacitet koji stvaraju slojevi neuronske mreže. Postupak kojim se računa gradijent neuronske mreže naziva se propagacija pogreške u nazad (engl. *backpropagation*).

Neuronska mreža se sastoji od ulaznog sloja, izlaznog sloja i određenog broja skrivenih slojeva (slojeva između ulaznog i izlaznog). Veličina ulaznog sloja odgovara veličini stanja. Veličina izlaznog sloja odgovara broju akcija i daje Q-vrijednost za svaku pojedinu akciju. Aktivacijska funkcija izlaznog sloja je linearna jer treba dati Q-vrijednost koja je neki realan broj. Skrivenih slojeva može biti proizvoljno (pa i niti jedan). Za aktivacijsku funkciju skrivenih slojeva postavljena je *ReLu*, iako to može biti i neka druga. ReLu je odabrana jer je to trenutno zastupljena funkcija koja pokazuje dobre rezultate [8].

2.6.1. Gradijentni spust i nadzirano učenje

Algoritam gradijentni spust (engl. *gradient descent*) (8) radi tako da u jednom svom koraku pomiče parametre funkcije u suprotnom smjeru od smjera gradijenta računatog u parametrima (točki) u kojima se trenutno nalazi (radi se o minimizaciji). On izvršava

korake dok god nije zadovoljen kriterij zaustavljanja. U radu će se koristiti poseban tip gradijentnog spusta, stohastički gradijentni spust - SGD (engl. *stochastic gradient descent*). To je modifikacija gradijentnog spusta koja se može koristiti kad god je funkcija koja se optimira suma drugih funkcija. Po nekom rasporedu se onda, u svakom koraku SGD-a, parametri funkcije pomiču po gradijentu jedne (iako je moguće i više) od tih drugih funkcija. Ta druga funkcija bit će kvadratna pogreška (engl. *squared error*) u čijem je izračunu razlika cilja i trenutne procjene Q-vrijednosti. Može se uočiti da pomicanje u smjeru gradijenta (8) kvadratne pogreške analogno osvježavanju Q-vrijednosti (2). Stopi α kojom se parametri funkcije pomiču u smjeru gradijenta (stopa učenja) analogna je α iz EMA. Može se uočiti da ovdje nema fiksne funkcije koja se SGD-om minimizira, nego se radi o imitaciji EMA.

$$a_{n+1} = a_n - \alpha * \nabla \mathrm{F}(a_n) \tag{8}$$

Inačica SGD-a koja se koristiti u ovom radu je *RMSprop*. RMSprop ima individualne stope učenja za različite parametre. To se uvelo kao modifikacija SGD-a jer funkcija koja se optimira može biti osjetljiva na promjenu nekih parametara, a neosjetljiva na promjenu nekih drugih. RMSprop je korišten u poznatom radu gdje su se učili računalni igrači za ATARI igre [9].

Korištenjem kvadratne pogreške ulazi se u područje nadziranog učenja. Podaci koji se koriste u nadziranom učenje podijeljeni su na svojstva (engl. *features*) i oznake (engl. *labels*). Cilj nadziranog učenja je da se neviđenim svojstvima pridruže točne oznake, da se generalizira. U kontekstu ovog problema svojstva su stanje i akcija, a oznaka je cilj. Funkcija koja se uči u procesu nadziranog učenja naziva se funkcija predikcije. Ovdje je ona istovjetna funkciji aproksimacije - $Q_{approx}(S_t)$. Ta funkcija predviđa Q-vrijednosti iz stanja i akcije, a naučena je na ciljevima. Kao što je već prije navedeno, radi lakše implementacije, funkcija aproksimacije primat će samo stanje, a na izlazu će davati Q-vrijednosti za sve moguće akcije. Kako se epizode odigravaju, agent uči od iskustava koje dobiva. Jedno iskustvo se sastoji od trojke: stanja, akcije izvršene u tom stanju i cilja za taj par stanje-akcija (svaki potez se dobiva novo iskustvo).

Funkcija u nastavku je ona koja će se minimizirati.

$$\frac{1}{n_{exp}} * \sum_{i=1}^{n_{exp}} \sum_{a=1}^{n_{action}} (a = action) * (q_{t_i} - Q_{approx}(S_{t_i})_a)^2 \quad (9)$$

Parametar n_{exp} označava ukupan broj iskustava od kojih se uči. Taj parametar ne mora označavati sva iskustva koje je agent ikada dobio. Zapravo, taj parametar nikada neće označavati sva iskustva. Kad bi parametar n_{exp} označavao sva iskustva ikad, agent bi mogao promijeniti svoje ponašanje samo jednom. Tako se nikada ne bi mogla otvoriti mogućnost da agent sakupi iskustva koja može dobiti samo s promijenjenim ponašanjem. Zbog ovoga je prikladan algoritam SGD pomoću kojega se uči samo iz jednog iskustva ($n_{exp} = 1$). Naime, pomoću algoritma SGD može se učiti i iz određenog skupa iskustva (engl. *mini-batch*), ne iz samo jednog. Oba načina će se koristiti u ovom radu. Parametar n_{action} označava ukupan broj akcija koje se mogu izvršiti, a u ovom radu on iznosi 3 (lijevo, gore, desno). Izraz $(a = action)$ služi tome da se u izračun uvrsti samo ona akcija koja je izvršena. Izraz $(q_{t_i} - Q_{approx}(S_{t_i})_a)^2$ je kvadratna pogreška. Pomoću $\frac{1}{n_{exp}} * \sum_{i=1}^{n_{exp}}$ se dobiva prosjek, dakle koristit će se srednja kvadratna pogreška (engl. *mean squared error*).

U uobičajenom nadziranom učenju postoji skup za učenje i skup za testiranje. Ovdje skupa za testiranje neće biti. Sva iskustva koje agent dobije bit će skup za učenje. Provjera kvalitete ponašanja agenta odvijat će se u testnoj okolini gdje će se agent empirički procijeniti pomoću mjera specifičnih za problem, a ne pomoću standardnih mjera poput preciznosti na skupu za testiranje, kako se to čini u uobičajenom nadziranom učenju. Mjera koja će se koristiti uključuje broj pobjeda, poraza i izjednačenih partija između dvije strategije.

2.7. Protivnik

U okruženje agenta može se svrstati i njegov protivnik u igri. On je uglavnom postavljen kao jedan od planirajućih igrača, iako to može biti i sam agent tj. agent može

igrati sam protiv sebe (engl. *self-play*). Kada agent igra sam protiv sebe, to zapravo znači da ista neuronska mreža vuče poteze i za jednog i za drugog i igrača. Kako se poboljšava igra prvog igrača, poboljšava se i igra drugog igrača, a kako je poboljšana igra drugoga igrač, to otvara priliku da se poboljša igra prvog igrača, i tako to točke ravnoteže. Tvrdi se da učenje u kojemu igrač igra sam protiv sebe završava u nekoj Nashovoj ravnoteži [3].

Za potrebe testiranja, prikaza rezultata ili razonode protivnik može biti i ljudski igrač.

2.8. Nestabilnost

Određena konfiguracija podržanog učenja može biti nestabilna i divergirati u učenju. Ta konfiguracija je TD-Off-Policy učenje sa skalabilnom funkcijom aproksimacije (kao što je neuronska mreža). Zove se još i smrtonosna trijada (engl. *the deadly triad*) [2]. Postoje trikovi kojima se može otkloniti ta nestabilnost, a oni su sljedeći. Jedan od njih je ponavljajuće korištenje istog iskustva - ER (engl. *experience replay*), a drugi je fiksiranje funkcije aproksimacije.

U ER-u iskustva se spremaju u spremnik. Pri svakom koraku učenja uniformno nasumično se uzorkuje iz spremnika određen broj iskustava koja se kao skup (engl. *mini-batch*) predaju SGD-u. Nakon što se spremnik napuni, novo iskustvo zamjenjuje ono koje je najduže bilo u spremniku.

Fiksiranje funkcije aproksimacije opisano je u nastavku. Kako se u TD načinu učenju treba odrediti cilj koji u sebi ima izračun aproksimacije buduće Q-vrijednosti, a od tog cilja se pri izračunu kvadratne pogreške oduzima trenutna Q-vrijednost, takvo učenje funkcije aproksimacije na temelju vlastitog izlaza izaziva nestabilnosti. Stoga se funkcija aproksimacije, kojom se računa buduća Q-vrijednost u izračunu cilja, fiksira određen broj koraka učenja. U implementaciji bi to bilo kopiranje strukture podataka koja sadrži neuronsku mrežu. „Druga“ funkcija aproksimacije koja računa trenutnu Q-vrijednost nije fiksirana i uči se.

Ono što se može reći za MC jest da je sporiji (treba mu više epizoda za istu učinkovitost), no stabilniji od TD-a [5].

2.9. Implementacija

Učenje agenta ide po sljedećoj proceduri. Prvo se postavlja model koji će se učiti. On se može inicijalizirati ili se može koristiti postojeći model koji se onda nastavlja učiti. Model se inicijalizira kao neuronska mreža s određenim brojem slojeva. Nakon što se postavi model inicijaliziraju se hiperparametri. U hiperparametre ulazi broj epizoda tj. broj partija iz kojih će agent učiti, smanjenje buduće nagrade gamma, početna temperatura za Boltzmannov pristup istraživanja te veličina spremnika za ER. Ako se želi ugasiti ER, veličina spremnika se postavlja na 1. Nakon što se inicijaliziraju hiperparametri, kreće se s epizodama tj. s odigravanjem partija igre. Prije samog odigravanja inicijalizira se ploča, protivnik te se postavljaju igrači na početne pozicije.

Po Boltzmannovom pristupu odabire se akcija za učećeg igrača, a protivnik po svojoj strategiji bira svoju akciju. Ako igrač igra sam protiv sebe, po Boltzmannovom pristupu bira se i protivnikova akcija. Kako bi se izvršila procedura podržanog učenja, elementi jednog koraka igre (istovremenog odigravanja igračevog i protivničkog poteza) gledanog od strane igrača se spremaju. Ako se radi o igranju sam protiv sebe, spremaju se elementi koraka igre gledani s obje strane. Prije nego što se izvrše akcije, sprema se stanje igre. Zatim se izvršavaju protivnikova i igračeva akcija te se igračeva sprema. Nakon što je izvršen korak igre, sprema se dobivena nagrada. Potencijalno se spremaju i novo stanje igre te nova igračeva akcija nakon izvršenog koraka. Novo stanje igre te nova igračeva akcija se spremaju ako se radi o Sarsa-i. Ako se radi o q-učenju, ne treba se spremati nova igračeva akcija jer se koristi pohlepna politika nad svim potencijalnim novim akcijama, dakle bira se ona s najvećom Q-vrijednosti. No, u q-učenju sprema se novo stanje igre jer se pomoću njega dobivaju Q-vrijednosti za sve potencijalne iduće akcije. Spremljeni elementi koraka igre stavljaju se u spremnik za ER ili se s njima, jedan po jedan spremljeni korak, izvršava učenje. U slučaju MC-a postoji poseban spremnik u koji se spremaju elementi svakog koraka igre od početka do kraja, ali u taj spremnik se ne moraju spremati novo stanje i nova akcija, kao što se to treba u Sarsa-i. Taj spremnik se prazni na početku partije (ER spremnik ne). Kod MC-a se na kraju partije obavlja smanjenje nagrada po (5) što je razlog za korištenje tog posebnog spremnika, budući da je potrebno zbrajati nagrade iz različitih koraka. U slučaju TD-a smanjenje nagrada se obavlja kad se pripremaju podaci za učenje, na primjer, po (3). Iz MC-ovog posebnog spremnika nema razloga prebacivati u

spremnik za ER (čim se koristi MC nije smrtonosna trijada), stoga se iz tog spremnika jedan po jedan korak partije odmah ubacuje u proceduru za učenje.

Pripremanje podataka za učenje uključuje stavljanje elemenata iz jednog (bez ER) ili više (s ER) koraka partije uz obradu u svojstva (X) i oznake (y). Trenutno stanje igre se stavlja u svojstva, a u oznake se stavlja vektor Q-vrijednosti izračunat pomoću ostalih spremljenih elemenata koraka partije. Vektor Q-vrijednosti sadrži Q-vrijednosti kakve bi izračunala trenutna funkcija aproksimacije Q-vrijednosti, osim one koja pripada akciji koja se izvršila. Ta Q-vrijednost računa se po (3), (4) ili (5). Nakon što su svojstva i oznake pripremljeni izvršava se korak SGD-a tj. učenja.

Koraci učenja se izvršavaju ili između dvije partije (MC, cijeli poseban spremnik se prazni) ili nakon svakog koraka partije (TD). Ako se koristi ER, a ER spremnik još nije pun, učenje nakon tih koraka partije se ne odvija nego se samo puni ER spremnik. Nakon svake epizode umanjuje se temperatura Boltzmannovog pristupa.

Osnovni kostur implementacije podržanog učenja preuzet je s bloga *outlace.com* [10]. Za korištenje programskog koda pogledati i Privitak.

2.10. Učenje promatranjem partija fiksnih igrača

U radu je isproban način učenja promatranjem partija koje odigravaju fiksni igrači. Pod fiksni se misli da ti igrači ne mijenjaju svoju strategiju kako prolaze epizode. To mogu biti ljudski igrač ili planirajući igrač. Oni ne koriste funkciju aproksimacije Q-vrijednosti za odabir akcije, no ona se uči. Uči se na temelju stanja u kojima se takvi igrači nađu, akcija koje izvršavaju i nagrada koje dobivaju. U implementaciji je jedina razlika ta što nema odabira akcije pomoću funkcije aproksimacije Q-vrijednosti kao što bi se ona odabrala, na primjer, Boltzmannovim pristupom. Na kraju svih epizoda tj. učenja dobiva se neka funkcija aproksimacije Q-vrijednosti na koju se onda može primijeniti pohlepna politika za odigravanje partija.

2.11. Ostvarenje nagrade

Pri ostvarenju nagrade u podržanom učenju treba paziti da agent pomoću takvog nagrađivanja ne nauči uspješno obavljati neki podzadatak, a stoga manje uspješno obavljati krajnji zadatak [2]. Naime, odstupiti od ovog pravila se smatra valjanim ako se pomoću nagrade koja točno opisuje krajnji zadatak ne uspijevaju dobiti željeni rezultati jer je ponašanje koje bi se trebalo naučiti presloženo. Korištenje nagrade za podzadatak odgovaralo bi heuristici, a korištenje nagrade za krajnji zadatak stvarnoj funkciji cilja. U ovoj igri primjer nagrade za podzadatak bio bi davanje određene pozitivne nagrade (npr. 2) nakon svakog poteza i tako maksimizirati broj poteza provedenih na ploči (treba uočiti da bi se u računanju vrijednosti početnog stanja sve nagrade zbrojile). Opis mane ovakvog nagrađivanja slijedi. Kad bi agent imao šansu pobijediti u nekom potezu, kao na primjer, "zidovima zatvoriti protivnika", on to ne bi učinio jer bi tako smanjio ukupan broj poteza provedenih na ploči, budući da se pobjedom igra završava. Primjer nagrade za krajnji zadatak bi bio nagrađivanje agenta samo na kraju partije, pozitivnom nagradom za pobjedu (npr. 10) i negativnom nagradom za poraz (npr. -10). Između poteza bi se davala neutralna nagrada (npr. 0). Kako je takva nagrada sukladna onome što zapravo želimo, možemo biti sigurni da se neće stvoriti neželjeno ponašanje kakvo je uočeno u davanju nagrade za podzadatak.

3. NEAT

3.1. Općenito

Osim prethodno opisanim tehnikama podržanog učenja strategija igrača razvijala se i evolucijskom tehnikom. Razvijanje strategije evolucijskim tehnikama može se svrstati i u podržano učenje, u metode pretraga politika (engl. *policy search*) [11].

Veliki broj evolucijskih tehnika bazira se na teoriji evolucije. Tehnika koja se u ovom radu koristi nije izuzetak. To je tehnika u kojoj se evoluira struktura i parametri (težine) neuronske mreže, a zove se NEAT – neuroevolucija povećavajućih topologija *(*engl. *neuroevolution of augmenting topologies).* Iz prethodnog opisa jasno je zašto se zove neuroevolucija. Povećavajuće topologije se odnose na postupni razvoj neuronske mreže, tj. kretanje od jednostavnije prema složenijoj neuronskoj mreži, na primjer, od one s manjim ukupnim brojem veza između čvorova prema onoj s većim.

Većina tehnika optimizacije koje se baziraju na teoriji evolucije sastoje se od funkcije dobrote, populacije rješenja, selekcije, križanja i mutacije. Opis takvih tehnika slijedi. Svako rješenje iz populacije rješenja evaluira se funkcijom dobrote. Nakon toga se obavlja selekcija rješenja prema nekoj strategiji u kojoj rješenja s većom dobrotom imaju veću vjerojatnost ne biti uklonjena iz populacije. Rješenja s većom dobrotom također imaju veću vjerojatnost sudjelovati u križanju. Križaju se dva rješenja po nekoj strategiji. Svaka strategija križanja sastoji se od stvaranja jednog ili više novih rješenja kombinirajući strukture rješenja koja se križaju. Mutacija je nasumična promjena u strukturi rješenja i primjenjuje se na svako rješenje s određenom vjerojatnošću [12].

U NEAT-u je rješenje neuronska mreža. Jedno obavljanje svih prethodno opisanih operacija naziva se generacija. NEAT radi tako da iterativno prolazi kroz generacije sve dok nije zadovoljen uvjet zaustavljanja.

Verzija NEAT-a koja će se u koristiti implementirana je u programskom paketu *neat-python*. Više informacija može se saznati u dokumentaciji od *neat-python* [13] i u radu *Stanley, Miikkulainen* [14].

3.2. Razvoj neuronske mreže

Jedino što je fiksno u neuronskoj mreži NEAT-a je broj čvorova u ulaznom i izlaznom sloju, sve ostalo je potencijalno podložno evoluciji. Potencijalno je podložno zato što se neki dijelovi mogu konfigurirati da ne budu.

Vrijednosti ulaznih čvorova odgovaraju ulazu u neuronsku mrežu. Funkcija koju računa čvor iz skrivenog ili izlaznog sloja je sljedeća:

$$activation(bias + (response * aggregation(inputs_i)))$$
$$inputs_i = [\, w_{i1} * out_1, \quad w_{i2} * out_2, \quad \dots\,] \tag{10}$$

Funkcije $activation$ i $aggregation$, kao i brojevi $bias$, $response$ i w_{in} potencijalno su podložni mutaciji (10). Funkcije se mutiraju tako da se nasumično iz skupine funkcija odabire ona koja će zamijeniti početnu (ako je mutacija funkcija isključena, uobičajena $activation$ funkcija koja se koristi je sigmoidalna funkcija, a uobičajena $aggregation$ funkcija je suma). Postoji niz načina na koje se brojevi mogu mutirati. Ove mutacije može se nazvati parametarske jer ne mijenjaju strukturu neuronske mreže. Dio $inputs_i$ sastoji se od liste izlaza čvorova koji su ulazno vezani uz zadani čvor (out_n) pomnoženih s težinom njihove veze (w_{in}).

Operacije mutacije kojima se mijenja struktura neuronske mreže su sljedeće: *stvori vezu između dva nevezana čvora*, *ukloni vezu između dva čvora*, *stvori novi čvor*, *ukloni čvor*. Prethodne mutacije može se nazvati strukturalne.

Veze između dva čvora označene su inovacijskim brojem. Inovacijski brojevi su jedinstveni identifikatori tih veza. Postoji globalni brojač koji dodjeljuje novi inovacijski broj svakoj novoj vezi. Problem koji se treba riješiti je dodavanje različitih inovacijskih brojeva efektivno istim vezama. Inovacijski brojevi se mogu promatrati kao historijske oznake pomoću kojih se zna koja je veza u rješenju, ali i globalno, stvorena prije, a koja poslije. Koriste se u križanju i selekciji.

U križanju se kombiniraju strukture dvaju rješenja koristeći pritom inovacijske brojeve (više u *Stanley, Miikkulainen* [14]). Pomoću inovacijskih brojeva se može

napraviti unija veza (posljedično i čvorova) jer se može izbjeći njihovo dupliciranje, lako se određuju one veze koje se podudaraju - one koje imaju iste inovacijske brojeve.

U selekciji inovacijski brojevi služe ze svrstavanje rješenja u vrste (engl. *species*). Vrste su skupine sličnih rješenja, a rješenja su to sličnija u što se više inovacijskih brojeva podudaraju (više u *Stanley, Miikkulainen* [14]). Primarna uloga vrste je očuvanje kompleksnih rješenja koja nemaju veliku dobrotu, nazvano zaštita inovacije (engl. *protection of innovation*). Tako se omogućuje daljnji razvoj sve kompleksnijih rješenja koji može dovesti do rješenja s dobrotom većom nego prijašnje najbolje (jednostavnije) rješenje. Smatra se da je ponekad za velik iznos dobrote potrebno imati kompleksno rješenje, a to se najčešće ne može znati prije nego što se do takvog rješenja dođe. Druga uloga vrste (koja se djelomično preklapa s prethodnom) je očuvanje raznolikosti rješenja. Ono se ostvaruje penaliziranjem dobrote rješenja to više što je broj rješenja u vrsti kojoj to rješenje pripada veći.

3.3. Dodatni mehanizmi

Opis nekih dodatnih mehanizama tehnika optimizacije temeljenih na teoriji evolucije slijedi. Jedna od njih je elitizam (engl. *elitism*). To je nemogućnost da selekcija iz populacije ukloni trenutno najbolje rješenje, ili čak *n* najboljih. Tako se najbolje rješenje nikada ne može izgubiti. Za NEAT specifičan mehanizam je stagnacija (engl. *stagnation*). Ona je vezana uz vrstu. To je broj generacija u kojima ne dolazi do poboljšanja dobrote vrste, a dobrota vrste je dobrota najboljeg rješenja u toj vrsti. Nakon što se prođe prag stagnacije, sva rješenja te vrste se uklanjaju. Kako ne bi došlo do nestanka svih rješenja, nazvanog izumiranje (engl. *extinction*), uvodi se mehanizam zvan elitizam vrste (engl. *species elitism*). To je nemogućnost uklanjanja rješenja iz vrste s najvećom dobrotom pomoću stagnacije. Elitizam vrste se proširuje i na *n* najboljih vrsta. Umjesto elitizma vrste koristi se mehanizam ponovnog pokretanja nakon izumiranja (engl. *reset on extinction*). Kada se zbog stagnacije ukloni zadnja vrsta, algoritam NEAT-a se pokreće ispočetka, od prve generacije.

U radu je korišten elitizam za 4 najbolje jedinke. Veličina populacije je bila 50. Korišten je elitizam vrste za 2 najbolje vrste. Nije korišten mehanizam ponovnog

pokretanja nakon izumiranja. Svi ostali hiperparametri jednaki su onima koji se koriste u primjernom problemu u dokumentaciji od *neat-python* [13].

3.4. Zaustavljanje

Zaustaviti algoritam NEAT-a se može na više načina. Jedan od njih je zaustavljanje kada neko rješenje postigne željenu dobrotu, ili veću od te željene (engl. *fitness threshold*). Drugi način je zaustavljanje algoritma nakon određenog vremena. Treći način je zaustavljanje algoritma nakon određenog broja generacija. Četvrti način je zaustavljanje kada u određenom broju generacija nema znatne promjene u dobroti najboljeg rješenja. Moguće je i kombinirati prethodno opisane načine. Algoritam se, na primjer, može zaustaviti kada se barem jedan od tih prethodno opisanih kriterija zadovolji.

U radu je upotrebljeno zaustavljanje nakon određenog vremena.

3.5. Funkcija dobrote

Funkcija dobrote realizirana je kao bodovanje za ishode odigranih partija igre. Pomoću rješenja tj. neuronske mreže se prilikom odigravanja partija odabiru akcije. Na ulaz neuronske mreže se daje trenutno stanje igre, a na izlazu se dobivaju težine različitih potencijalnih akcija. Bira se ona akcija s najvećom težinom (analogno pohlepnoj politici). Sa svakim rješenjem u populaciji odigrava se određen broj partija protiv različitih protivnika. Broj partija po protivniku i tipovi protivnika su isti za sva rješenja. Za pobjedu u partiji povećava se dobrota rješenja (npr. +1), a za poraz se smanjuje (npr. -1). Prije odigravanja prve partije dobrota rješenja se postavlja na neku konstantu (npr. 0).

Budući da u protivnicima i početnom postavljanju igrača postoji određena stohastičnost, što se više partija odigra s nekim rješenjem, to je bolja procjena njegove dobrote. No, cijena koja se plaća za to je više potrošenog vremena potrebnog za odigravanje svih tih partija.

Konkretno, za tipove protivnika uzeti su "random player" i "look player-i" s parametrizacijom $x = 1$ i $x = 5$ i protiv svakog je odigrano 50 partija.

4. Rezultati

U sklopu rezultata rada računalni igrači su izrađeni na različite načine i zatim evaluirani u igrama s planirajućim igračima. Računalni igrač se izrađivao pomoću podržanog učenja i NEAT-a.

U tablici u nastavku (*Tablica 1.*) prikazani su hiperparametri koji su korišteni u svim izradama računalnog igrača pomoću podržanog učenja.

neuronska mreža	2 skrivena sloja: 1. 70 čvorova, 2. 35 čvorova
τ_0	20.0
γ	0.9
broj epizoda	50 000
veličina ploče (početna konfiguracija : veličina ploče)	na suprotnim stranama (NSS) : 5 x 5 pseudo-nasumično (PSN) : 6 x 6

Tablica 1. Konstantni hiperparametri

U slijedećoj tablici prikazana je osnovna konfiguracija podržanog učenja (*Tablica 2.*). Svaki hiperparametar te konfiguracije mijenjan je radi usporedbe. Hiperparametri se nisu mijenjali po rešetci (engl. *grid search*) nego se u odnosu na osnovnu konfiguraciju uvijek mijenjao samo jedan (*Tablica 3.*). Negativna strana takvog načina je ta što se u usporedbi nije uzela u obzir međuovisnost različitih hiperparametara, ako ona postoji. U NEAT-u se jedino mijenjalo stanje.

Treba napomenuti da se s promjenom korištenog stanja mijenja veličina ulaznog sloja neuronske mreže. Veličina ulaznog sloja jednaka je broju bitova potrebnih za ostvarenje stanja. Za detaljan opis svakog ostvarenja stanja pogledati poglavlje 1.4.

cilj	*Monte-Carlo*
protivnik	"look player" s param. $x = 5$
stanje	s reduciranom informacijom (način s protivnikom) s param. *howFar = 2*
nagrada (početna konfiguracija : nagrada)	na suprotnim stranama (NSS) : 10 za pobjedu, -10 za poraz pseudo-nasumično (PSN) : 6 za pobjedu, -6 za poraz, 2 za svaki novi "zid"

Tablica 2. Osnovna konfiguracija

U ćelijama sljedećih tablica (*Tablica 3.*, *Tablica 4.*) prikazane su pobjede, porazi i neriješeni ishodi naučenih/evoluiranih igrača protiv određenih planirajućih igrača kada su igrači bili početno postavljeni na suprotnim stranama (gore) ili pseudo-nasumično (dolje). Pobjeda se odnosi na pobjedu naučenog/evoluiranog igrača. Svi naučeni/evoluirani igrači, osim kada se u podržanom učenju učilo metodom igranja sam protiv sebe, mogu igrati samo kao igrač-1, jer su tako naučeni/evoluirani. Ukupan broj partija je 60.

Igrač je bodovan kako bi ga se proglasilo uspješnim ili neuspješnim. Dobiva se 1 bod za pobjedu, 0 bodova za neriješeno i -1 bod za poraz. Raspon bodova je od -60 do 60. Oznaka "-" u tablici (*Tablica 3.*) označava da se nije uspjelo izraditi uspješnog igrača, igrača koji u srazu s "random player-om" ima barem 15 bodova u oba načina početnog postavljanja. Razlozi za dobivanje neuspješnih igrača nisu jasni.

pobjeda / neriješeno / poraz *NSS / PSN*	"random player"	"look player" s param. $x = 1$	"look player" s param. $x = 5$
PODRŽANO UČENJE			
osnovna konfiguracija	NSS: 52 / 8 / 0	20 / 34 / 6	8 / 51 / 1

	PSN: 59 / 1 / 0	38 / 8 / 14	31 / 11 / 18
promjena cilja:			
cilj: Sarsa	51 / 9 / 0 60 / 0 / 0	14 / 31 / 15 34 / 8 / 18	4 / 41 / 15 30 / 7 / 23
cilj: Q-učenje	57 / 3 / 0 60 / 0 / 0	36 / 20 / 4 37 / 11 / 12	35 / 23 / 2 32 / 13 / 15
promjena protivnika:			
protivnik: "random player"	53 / 3 / 4 47 / 4 / 9	17 / 15 / 28 4 / 8 / 48	0 / 18 / 42 4 / 2 / 54
protivnik: "random player", "look player-i" s param. $x = 1$ i $x = 5$	50 / 10 / 0 58 / 1 / 1	6 / 43 / 11 24 / 9 / 27	3 / 42 / 15 18 / 5 / 37
protivnik: *self-play*	51 / 9 / 0 54 / 5 / 1	11 / 48 / 1 11 / 8 / 41	4 / 49 / 7 8 / 7 / 45
protivnik: *promatranje partija fiksnih igrača* ("look player-i" s param. $x = 5$)	- -	- -	- -
promjena stanja:			
stanje: s potpunom informacijom	51 / 9 / 0 35 / 10 / 15	24 / 30 / 6 3 / 2 / 55	8 / 47 / 5 0 / 3 / 57
stanje: s reduciranom informacijom (način bez protivnika) s param. $howFar = 3$	50 / 10 / 0 -	14 / 43 / 3 -	8 / 45 / 7 -
promjena nagrade: *(ako se nagrada podudara s onom iz osnovne konfiguracije, rezultat se kopira)*			

nagrada: 10 za pobjedu, - 10 za poraz	52 / 8 / 0 54 / 5 / 1	20 / 34 / 6 9 / 9 / 42	8 / 51 / 1 8 / 7 / 45
nagrada: 6 za pobjedu, -6 za poraz, 2 za svaki novi zid	51 / 5 / 4 59 / 1 / 0	0 / 23 / 37 38 / 8 / 14	0 / 17 / 43 31 / 11 / 18
nagrada: 2 za svaki novi zid	41 / 2 / 17 59 / 1 / 0	0 / 0 / 60 15 / 5 / 40	0 / 0 / 60 6 / 8 / 46

Tablica 3. Prikaz rezultata za podržano učenje

pobjeda / neriješeno / poraz *NSS / PSN*	"random player"	"look player" s param. $x = 1$	"look player" s param. $x = 5$
NEAT			
stanje: s potpunom informacijom	NSS: 58 / 2 / 0 PSN: 41 / 4 / 15	26 / 34 / 0 8 / 5 / 47	16 / 43 / 1 6 / 6 / 48
stanje: s reduciranom informacijom (način s protivnikom) s param. $howFar = 2$	55 / 5 / 0 53 / 3 / 4	40 / 17 / 3 22 / 4 / 33	25 / 33 / 2 7 / 6 / 47
stanje: s reduciranom informacijom (način bez protivnika) s param. $howFar = 3$	51 / 8 / 1 48 / 9 / 3	39 / 20 / 1 15 / 8 / 37	19 / 38 / 3 8 / 4 / 48
stanje: s reduciranom informacijom (način bez protivnika) s param. $howFar = 2$	53 / 7 / 0 57 / 2 / 1	37 / 23 / 0 20 / 6 / 34	27 / 32 / 1 8 / 5 / 47

Tablica 4. Prikaz rezultata za NEAT

U nastavku slijede tvrdnje koje su podržane prethodnim tablicama (*Tablica 3.*, *Tablica 4.*) (nisu izvedene iz nje). Jedne postavke učenja/evolucije se proglašava boljima od druge ako je razlika bodova tih postavki (boduje se kao za uspješnost igrača) u partijama s "look player-om" parametriziranim s $x = 5$ u PSN početnom postavljanju veća od 30.

- podržanim učenjem se izradi bolji igrač za igru sa sumom nula kada mu je protivnik bolji (usporedba 'osnovna konfiguracija (13 bodova)' – 'protivnik: "random player" (-50 bodova)')
- podržano učenje promatranjem partija fiksnih igrača nije proradilo
- u podržanom učenju složenije stanje usporava učenje (usporedba 'osnovna konfiguracija (13 bodova)' – 'stanje: s potpunom informacijom (-57 bodova)')
- problem može biti dovoljno složen da je opravdano koristiti nagradu za podzadatak (usporedba 'osnovna konfiguracija (13 bodova) ' – 'nagrada: 10 za pobjedu, - 10 za poraz (-37 bodova)')
- podržano učenje pokazuje bolje rezultate od NEAT-a (uspoređuju se rezultati s najviše bodova od obje tehnike) (usporedba 'podržano učenje, cilj: Q-učenje (17 bodova)' – 'NEAT, stanje: s reduciranom informacijom (način bez protivnika) s param. $howFar = 2$ (-39 bodova)')

Treba napomenuti da su za svaku postavku učenja/evolucije prikazani rezultati samo jedne izrade računalnog igrača i da statistička značajnost rezultata nije analizirana standardnim mjerama.

4.1. Dodatna analiza

U varijanti igre gdje se igrače na početku postavlja na suprotne strane i $size_x$ je neparan postoji strategija kojom se uvijek pobjeđuje ili igra neriješeno. U početku igrač skreće desno i ide po $x - osi$ do sredine mape. Zatim skreće lijevo i po $y - osi$ ide do kraja mape. Tamo skreće lijevo i kreće se tako da bi popunio "zidom" sva prazna polja koja može. Ova strategija je Nashova ravnoteža. Autor smatra da je to jedina Nashova ravnoteža za dane postavke igre. Za objašnjenje Nashove ravnoteže pogledati poglavlje 2.3.

Nije korišten vjerojatnosni odabir akcija u podržanom učenju jer se smatra da nije potreban, budući da planirajući igrači - protivnici ne mogu naučiti o strategiji igrača, i onda to znanje primijeniti. Za objašnjenje pogledati poglavlje 2.3.

4.2. Trajanje izrade

Podržano učenje s navedenim hiperparametrima (*Tablica 1.*), kada je cilj bio MC ili Sarsa, trajalo je oko 10 minuta pokrenuto kao jedini Python proces na računalu [Privitak]. Q-učenje se odužilo do reda veličine 10 sati. Autor smatra da je to zbog ER-a. (veličina ER spremnika bila je 500, a skup koji se uzorkovao bio je veličine 50).

Svi igrači dobiveni NEAT-om evoluirani su 34 sata. Sve evolucije bile su pokrenute istovremeno na računalu kao zasebni procesi [Privitak]. Broj generacija u evolucijama bio je $600 \ +/- \ 300$.

Zaključak

U ovom radu implementirana je video igra (sa sumom nula) po uzoru na postojeću naziva "Light Riders". Izrađeni su računalni igrači za tu video igru pomoću različitih tehnika. Upotrijebile su se tehnike planiranja, podržano učenje i neuroevolucija. Sve tehnike su opisane u radu. Igrači su bili izrađivani pod različitim hiperparametrima. Rezultati su podržali određene tvrdnje, a neke od njih su u nastavku. Podržanim učenjem se izradi bolji računalni igrač za igru sa sumom nula ako mu je protivnik bolji. U podržanom učenju složenije stanje usporava učenje. U podržanom učenju problem može biti dovoljno složen da je opravdano koristiti nagradu za podzadatak. Podržano učenje pokazuje bolje rezultate od NEAT-a. Način usporedbe je opisan u radu.

Literatura

[1] "Light Riders", URL:https://playground.riddles.io/competitions/light-riders, 26.05.2018.

[2] Sutton R. S., Barto A. G., "Reinforcement Learning: An Introduction", The MIT Press, 2014, 2015, 2016, URL: http://incompleteideas.net/book/bookdraft2017nov5.pdf, 26.05.2018.

[3] Silver D., "Reinforcement Learning", UCL course, URL:http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html, https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT , 26.05.2018.

[4] Jackson M. O., Leyton-Brown K., Shoham Y. ,"Game Theory", URL:https://www.coursera.org/learn/game-theory-1, 26.05.2018.

[5] Panin A., "Footnote: Monte-Carlo vs. Temporal-Difference", URL:https://www.coursera.org/learn/practical-rl/lecture/v1bel/footnote-monte-carlo-vs-temporal-difference, 26.05.2018.

[6] "Exponential Moving Average - EMA", URL:https://www.investopedia.com/terms/e/ema.asp, 26.05.2018.

[7] Juliani A., "Action-Selection Strategies for Exploration", URL:https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-strategies-for-exploration-d3a97b7cceaf, 26.05.2018.

[8] "Why is ReLu better than the other activation functions", URL:https://datascience.stackexchange.com/questions/23493/why-relu-is-better-than-the-other-activation-functions, 26.06.2018.

[9] Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., Riedmiller M., "Playing Atari with Deep Reinforcement Learning", DeepMind Technologies, URL:https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf, 26.05.2018.

[10] "Learning Gridworld With Q-learning", URL:http://outlace.com/rlpart3.html, 26.05.2018.

[11] Arulkumaran K, Deisenroth M. P., Brundage M., Bharath A. A., "A Brief Survey of Deep Reinforcement Learning", URL: https://arxiv.org/pdf/1708.05866.pdf, 26.05.2018.

[12] Jakobivić D., "Analiza i projektiranje računalom: Genetski algoritmi - predavanje", URL:http://www.fer.unizg.hr/_download/repository/GApredavanje.pdf, 26.05.2018.
[13] "NEAT-Python", URL:http://neat-python.readthedocs.io/en/latest/, 26.05.2018.
[14] Stanley K. O., Miikkulainen R., "Evolving Neural Networks through Augmenting Topologies", The MIT Press, URL:http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf, 26.05.2018.

Sažetak

Primjena postupaka umjetne inteligencije za izradu računalnog igrača u video igri

U sklopu ovog rada implementirana je video igra po uzoru na postojeću naziva "Light Riders". Cilj rada je bio izraditi računalnog igrača za tu video igru. Korištene su razvojne tehnike podržanog učenja i neuroevolucije povećavajućih topologija i tehnike planiranja. Sve korištene tehnike su opisane. Igrači izrađeni tehnikama planiranja korišteni su i kao dio razvojnih tehnika, bilo kao okruženje agenta u podržanom učenju, bilo kao dio funkcije dobrote u neuroevoluciji. Korištena je i tehnika učenja igranjem sam protiv sebe. Implementacija razvojnih tehnika uključivala je dizajniranje stanja igre i nagrađivanja / funkcije dobrote. Igrači su izrađeni pod različitim hiperparametrima i njihovom usporedbom su se podržale određene tvrdnje.

ključne riječi: podržano učenje, NEAT, video igra, računalni igrač

Summary

Artificial intelligence methods for creating a computer player in a game

In this thesis the video game modeled by the one called "Light Riders" was implemented. The goal of thesis was to develop a computer player for this video game. Development techniques that were used were reinforcement learning and neuroevolution of augmenting topologies (NEAT). Also, planning techniques were used. All techniques were described. Planning players were used as a part of development techniques, either as an agent's environment in reinforcement learning, either as a part of fitness function in neuroevolution. Self-play technique was also used. Implementation of development techniques included the design of game state and reward function / fitness function. Players were developed with different hyperparameters and by their comparison certain statements were supported.

keywords: reinforcement learning, NEAT, video game, computer player

Skraćenice

EMA	*Exponential Moving Average*	eksponencijalni putujući prosjek
MC	*Monte-Carlo*	-
TD	*Temporal-Difference*	-
SGD	*Stochastic Gradient Descent*	stohastični gradijentni spust
ER	*Experience Replay*	ponavljajuće korištenje istog iskustva
NEAT	*Neuroevolution of Augmenting Topologies*	neuroevolucija povećavajućih topologija
RL	*Reinforcement Learning*	podržano učenje
RF	*Reduced Features*	svojstva s reduciranom informacijom (misli se na stanje)
ODS / NSS	*On Different Sides*	na suprotnim stranama
PSR / PSN	*Pseudo-Random*	pseudo-nasumično
WO	*With Opponent*	s protivnikom
FULL	*Full*	potpuno (misli se na stanje s potpunom informacijom)
WATCH	*Fixed Players*	fiksni igrači
TECHN	*Technique*	tehnika
F	*Final*	završno

Privitak

Korišteno računalo

procesor: i3-4030U (1.9 Ghz, 3MB L3 cache)

radna memorija: 12 GB

(nije korišteno ubrzanje pomoću grafičke kartice)

Korištenje programa

Igra se može pokrenuti iz Python datoteke "runGame.py". Primjeri korištenja različitih igrača i ostalih postavki navedeni su kao komentari u datoteci. Kontrole za igranje preko tipkovnice su "w" za ravno, "a" za lijevo i "d" za desno. Naučeni modeli mogu se pronaći u mapama s prefiksom "results*". Ako se želi učiti novi model, može se kao inicijalni programski kod koristiti onaj iz Python datoteka s prefiksima "RL*" i "NEAT*". Iz imena datoteke može se vidjeti kakve su postavke učenja u pojedinoj datoteci (objašnjenje potražiti među Skraćenice). Za korištenje NEAT-a pogledati dokumentaciju od *neat-python* [13]. Za korištenje podržanog učenja pogledati poglavlje 2.9 i blog *outlace.com* [10]. Ako se želi testirati model, može se koristiti programski kod iz Python datoteke "runStat.py". Uz rad je priložen PyCharm projekt naziva "dipl" u kojemu su sve dosad spomenute datoteke.