UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Matei Jon Stanču

UČENJE AUTONOMNOG KRETANJA U IGRAMA SA PERSPEKTIVOM IZ PRVOG LICA

master rad

Beograd, 2022.

Mentor:

dr Mladen Nikolić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Jovana Kovačević, docent
Univerzitet u Beogradu, Matematičku fakultet

dr Aleksandar Kartelj, docent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Zahvaljujem se komisiji koja je učestvovala u izradi ovog master rada, profesoru Mladenu na korisnim savetima i pomoći, a porodici i prijateljima na strpljenju.

Naslov master rada: Učenje autonomnog kretanja u igrama sa perspektivom iz prvog lica

Rezime: Autonomno upravljanje u video igrama predstavlja jednu od najatraktivnijih oblasti mašinskog učenja u poslednjih deset godina. Ostvarila je značajan napredak u primeni algoritama vizuelnog učenja potkrepljivanjem u praksi, međutim, za primenu ovih algoritama na probleme učenja autonomnog upravljanja u realističnim 3D okruženjima potrebno je napraviti korak koji će približiti ove dve oblasti. Problem koji se u ovom radu istražuje je učenje autonomnog kretanja u pucačkim igrama sa perspektivom iz prvog lica. Cilj je ispitati da li su metode koje su korišćene u ranijim istraživanjima koja obuhvataju primenu na probleme sa 2D okruženjima dovoljno moćne da uspešno reše problem autonomnog upravljanja u igrama sa perpsektivom iz prvog lica. Takođe, cilj je pokazati da je okruženje ViZDoom pogodno za izvođenje eksperimenata u oblasti vizuelnog učenja potkrepljivanjem i utvrditi osnovu za buduća istraživanja na ovom ili sličnim problemima. Rezultati ovog istraživanja pokazuju da je napredna implementacija algoritma duboka Q mreža uspešno rešila problem autonomnog kretanja i upravljanja resursima u 3D okruženju. Takođe, potvrđeno je da je ViZDoom platforma korisna zamena realističnim 3D okruženjima u izvođenju eksperimenata u oblasti vizuelnog učenja potkrepljivanjem.

Ključne reči: vizuelno učenje potkrepljivanjem, autonomno upravljanje, TD učenje, duboka Q mreža

Sadržaj

Glava 1

Uvod

Postojanje života je oduvek bilo određeno njegovom sposobnošću prilagođavanja promenama u okruženju. Prilagođavanje života se dešava u kontinuitetu na dva nivoa. Jedno je na kolektivnom nivou pomoću mehanizama evolucije, a drugo je na individualnom pomoću inteligencije. Da bi inteligencija postojala mora da postoji i razlog, jer ako se posmatraju organizmi koji danas postoje na svetu, može da se primeti da su inteligenciju razvili i zadržali samo oni koji su imali razlog i potrebu za tim. Glavni razlog za to je uvek bila neka vrsta opasnosti koja, direktno ili indirektno, ugrožava opstanak organizma. Pored potencijala za opstanak organizma, morao je da postoji i nivo težine problema u okruženju koji odgovara tom potencijalu kako bi se podsticao razvoj inteligencije.

Od trenutka rođenja deteta pa do kraja njegovog života, ono se prilagođava svom okruženju pomoću urođenog mehanizma učenja. Njegov senzomotorni sistem je povezan sa okruženjem tako da mu ponašanje omogućava generisanje iskustva vezanog za posledice njegovih akcija. Na osnovu tog iskustva može da nauči koje su akcije doprinele ostvarivanju kog događaja da bi kasnije moglo da, u odgovarajućim situacijama, ponovi ispravne akcije kako bi postiglo svoje ciljeve. Ovaj proces učenja potkrepljivanjem će mu vremenom omogućavati da, od koraka do koraka, vrši sve složenija predviđanja vezana za okruženje na osnovu kojih će moći da vrši složenije radnje poput igranje fudbala ili vožnje bicikla.

Okruženje i subjekti koji se u njemu posmatraju ne moraju biti stvarni svet i živi organizmi, to može biti neko digitalno okruženje poput video igre i veštačke inteligencije koja tom igrom upravlja. Oblast učenja potkrepljivanjem u video igrama u poslednjih deset godina ostvaruje velike rezultate u razumevanju ovih koncepata i primeni algoritama u praksi. Međutim, i dalje postoji veliki jaz između

simulacije ovih sistema na računarima i primene u stvarnom svetu. Zbog toga je potrebno napraviti korak koji će smanjiti ovu razliku tako što će ideje korišćene u vizuelnom učenju potkrepljivanjem u igrama sa 2D okruženjima biti proširene na probleme autonomnog upravljanja u igrama sa perspektivom iz prvog lica.

Potrebno je najpre pronaći platformu koje omogućava lako ispitivanje metoda učenja potkrepljivanjem u složenijim okruženjima. Jednu takvu platformu predstavlja ViZDoom okruženje koja je zasnovana na pokretaču poznate igre Doom. Cilj rada je ispitati mogućnosti koje nudi okruženje ViZDoom i pokazati da ova platforma predstavlja dobar poligon za izvođenje eksperimenata u oblasti vizuelnog učenja potkrepljivanjem. Takođe, potrebno je ispitati poznate metode iz ranijih istraživanja na probleme autonomnog upravljanja u igrama sa perspektivom iz prvog lica i zaključiti kolika je njihova moć u primenama koja zahtevaju značajnije prostorno rezonovanje. Konačno, utvrditi čvrstu osnovu za izvođenje sličnih eksperimenata i proširivanje na buduća istraživanja koja u isto vreme omogućava i lako poređenje sa rezultatima ranijih istraživanja.

Za ostvarivanje svih ovih ciljeva, metode su ispitane na jednostavnoj igri prikupljanja paketa prve pomoći u okruženju koje postepeno povređuje igrača. Analizirani su rezultati obučavanja različitih implementacija duboke Q mreže. Najpre je utvrđen osnovni model koji je implementiran na osnovu verzija ovog algoritma koje su korišćene u ranijim istraživanjima. Zatim su ispitane performanse različitih modela koji predstavljaju unapređenja osnovne implementacije. Na kraju su svi modeli kombinovani u jednu naprednu implementaciju, pri čemu su rezultati svih implementacija upoređivani. Dodatno, ispitana je i komplementarnost komponenti napredne implementacije tako što su bile ispitane verzije napredne implementacije čija su pojedinačna unapređenja bila odstranjena.

Rezultati ovog istraživanja pokazuju da napredne metode korišćene u ranijim istraživanjima mogu da reše problem autonomnog kretanja i upravljanja resursima u pucačkim igrama sa perspektivom iz prvog lica. Takođe, rezultati pokazuju da svi zaključci vezani za komplementarnost komponenti naprednih implementacija duboke Q mreže iz ranijih istraživanja važe i u slučaju autonomnog upravljanja u igrama sa perspektivom iz prvog lica.

U nastavku ovog rada biće najpre definisani osnovni pojmovi i metode koje su korišćene tokom istraživanja. Zatim, biće detaljno opisan problem koji se istražuje, kao i algoritmi koji su korišćeni za rešavanje ovog problema. Konačno, biće predstavljeni i rezultati ispitivanja svih korišćenih algoritama.

Glava 2

Osnovni pojmovi i korišćene metode

Svako naučno istraživanje počinje posmatranjem prirodnih fenomena radi sticanja nekog ograničenog saznanja u vezi sa problemom koji je potrebno rešiti. To neprecizno poznavanje posmatranog problema se zatim na precizan način predstavlja matematičkom formulacijom koja predstavlja model posmatranog problema. U zavisnosti od izabranog modela biraju se i metode i koncepti koji će se koristiti za opisivanje, analizu ili predviđanja u terminima matematičkog jezika. Rezultati tih ispitivanja se zatim prevode u praktična rešenja originalnog problema čijom se primenom dolazi do novih saznanja vezanih za inicijalna posmatranja.

Izbor metoda je važan iz više razloga. Poređenje različitih metoda omogućava razumevanje koje su metode pogodnije za rešavanje posmatranog problema. Neke metode su teoretski zanimljivije jer uspevaju da na jasan način objasne suštinu problema, dok su druge bolje rešenje kada je potrebno rešavati probleme velikih dimenzija u realnom vremenu kao što je to obično u praksi. Rezultati primena metoda su od suštinskog značaja jer pokazuju da li izabrani model odgovara inicijalnim opservacijama. Ako rezultati ispitivanja nemaju smisla, moguće je da model ne može da uhvati sve važne karakteristike problema, pa je potrebno revidirati proces modeliranja.

Matematički modeli mogu imati mnogo oblika, uključujući dinamičke sisteme, statističke modele, diferencijalne jednačine ili teorijske modele igara. Za razliku od statičkih modela koji izračunavaju sistem koji se nalazi u ravnoteži, dinamički modeli uzimaju u obzir vremensko zavisne promene stanja sistema. Jedan od takvih modela je i model koji se koristi u rešavaju problema učenja potkrepljivanjem.

2.1 Markovljevi procesi odlučivanja

Markovljevi procesi odlučivanja (eng. *Markov decision process*, MDP) predstavljaju diskretni stohastički kontrolni proces i proširenje Markovljevih lanaca. Čine teorijski okvir za modeliranje odlučivanja u situacijama u kojima su ishodi delom nasumični i delom pod kontrolom donosioca odluka. MDP-ovi su korisni za proučavanje optimizacionih problema koji se rešavaju dinamičkim programiranjem. Predstavljaju klasičnu formalizaciju sekvencijalnog donošenja odluka, u kom akcije utiču ne samo na neposredne nagrade, već i na naredna stanja okruženja i buduće nagrade nakon toga. Stoga, MDP podrazumeva odlaganje nagrađivanja kroz potrebu za nagodbom između primanja neposrednih i budućih nagrada.

U nastavku biće definisani elementi koji čine MDP kao i osnovni pojmovi kao što su dobit, funkcije vrednosti i Belmanove jednačine koji se koriste u velikom broju algoritama učenja potkrepljivanjem.

Sistem agent-okruženje

Modeli koji opisuju problem učenja potkrepljivanjem definišu se pomoću sistema koji menja svoje stanje u diskretnim vremenskim trenucima. Taj sistem čine *agent*, koji je u mogućnosti da preduzima *akcije*, i time utiče na svoju okolinu, i *okruženje*, koje pruža agentu neku reprezentaciju svog *stanja* na osnovu kojeg on može da donosi odluke. Ove interakcije agenta i okruženja se dešavaju u kontinuitetu. Agent preduzima neke akcije a okruženje odgovara na to tako što agentu predstavi nove reprezentacije svog stanja. Uz stanje, okruženje takođe agentu prosleđuje i *nagradu* koja predstavlja numeričku vrednost koju agent, izborom svojih akcija, vremenom pokušava da maksimizuje.

Formalnije, agent i okruženje interaguju u diskretnim vremenskim trenucima, $t = 0, 1, 2, 3, \ldots$. U svakom koraku t, agent opazi neku reprezentaciju stanja okruženja, $S_t \in \mathcal{S}$, i na osnovu toga odabere jednu akciju, $A_t \in \mathcal{A}(s)$. Nakon jednog koraka, kao posledicu svoje odluke, agent primi od okruženja nagradu, $R_t \in \mathcal{R} \subset \mathbb{R}$, pri čemu se agent nalazi u novom stanju S_{t+1}. MDP i agent generišu putanju i nakon nekoliko koraka ona ima sledeći oblik:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Skupovi stanja, akcija i nagrada $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ imaju konačan broj elemenata u *konačnim* MDP-ovima. U slučaju konačnih MDP-ova, slučajne veličine R_t i S_t imaju

jasno definisane diskretne respodele verovatnoća koje zavise samo od prethodnog stanja i akcije. Odnosno, za bilo koje konkretne vrednosti prethodnog stanja i akcije $s \in \mathcal{S}$ i $a \in \mathcal{A}(s)$, verovatnoća pojavljivanja konkretnih vrednosti sledećeg stanja i nagrade $s' \in \mathcal{S}$ i $r \in \mathcal{R}$ definisana je na sledeći način:

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s).$$

Funkcija p definiše *dinamiku* Markovljevih procesa odlučivanja. Odnosno, verovatnoće svih mogućih vrednosti slučajnih veličina S_t i R_t zavise isključivo od neposrednih prethodnih vrednosti stanja i akcija S_{t-1} i A_{t-1} a ne od kompletne istorije procesa. Ovo svojstvo se zove *Markovljevo svojstvo* i ne predstavlja ograničenje procesa odlučivanja, već samog stanja. Markovljevo svojstvo podrazumeva da stanje mora da obuhvati informaciju sa svih aspekata prethodne interakcije agenta sa okruženjem koja zaista pravi razliku u budućnosti. Za uvođenje preostalih pojmova, u nastavku biće pretpostavljeno da Markovljevo svojstvo važi, čak i ako to u praksi često nije slučaj, pre svega kada su u pitanju metode koje se oslanjaju na funkcionalnu aproksimaciju.

Modeliranje pomoću MDP-a omogućava opisivanje velikog broja različitih problema na različitim nivoima apstrakcije. Njihova fleksibilnost omogućava definisanje problema u kojima vremenski koraci preduzimanja akcija ne moraju biti jednaki vremenski intervali. Oni mogu da se odnose na različite faze delanja. Na primer, trenutak preduzimanja akcije može da se odnosi na pojedinačne nervne impulse u mišićima, ili na neke odluke visokog nivoa kao što je šetnja kroz park ili čitati knjigu. Takođe, stanja mogu imati različite nivoe reprezentacije. Na primer, stanje sa reprezentacijom niskog nivoa bi moglo biti elektromagnetno zračenje ili promena pritiska u vazduhu, dok bi stanje sa reprezentacijom visokog nivoa mogao biti simbolični opis objekta.

Važno je napomenuti da granica između agenta i okruženja sa aspekta MDP-a nije uvek toliko jasna kao što se to čini u fizičkom svetu. Prirodno je pomisliti da su zglobovi delovi agenta jer su oni ujedno i delovi tela životinje, međutim, granica između agenta i okruženja u kontekstu MDP-a je zapravo bliža agentu. Slično bi moglo da se kaže i za nagradu - da je ona deo agenta jer se računa unutar tela životinje, ali ona zapravo predstavlja deo okruženja jer nju agent ne može da promeni. Sve što agent ne može svojevoljno da promeni direktno preduzimanjem akcija, smatra se delom spoljašnje sredine. Čest je slučaj da agent zna da se

nagrade računaju kao funkcija stanja okruženja i preduzetih akcija, ali bi ih ipak trebalo shvatiti kao deo okruženja jer one definišu zadatak agenta i ne bi trebalo da budu nešto što agent može da promeni. Takođe, agent može da zna kako okruženje funkcioniše ali ne i da reši zadatak. Na primer to se može posmatrati na primeru rešavanja Rubikove kocke. Svi ljudi znaju kako kocka funkcioniše, ali ne znaju svi da je reše. Stoga, granicu između agenta i okruženja u kontekstu MDP-a je najbolje definisati njegovom potpunom kontrolom, a ne njegovim znanjem.

Cilj i nagrada

Jednu od najprepoznatljivijih karakteristika učenja potkrepljivanjem predstavlja formalizacija pojma cilja pomoću signala nagrade. U problemu učenja potkrepljivanjem cilj agenta je da maksimizuje ukupnu akumuliranu dugoročnu nagradu. Na osnovu ove neformalne definicije se može izneti i osnovna hipoteza učenja potkrepljivanjem koja se zove *hipoteza nagrade* [44]:

> *Sve ono što se podrazumeva pod ciljem i svrhom, može se shvatiti kao maksimizacija očekivane vrednosti sume primljenog skalarnog signala koji se zove nagrada.*

Iako se formulisanje ciljeva pomoću signala nagrade na prvi pogled može činiti veoma ograničavajuće, u praksi se pokazalao kao fleksibilno i široko upotrebljivo. Međutim, iako su rezultati koji su postignuti upotrebom ove hipoteze impresivni, treba imati na umu da je njena upotrebljivost ipak ograničena, jer nije najjasnije kako modelom maksimizacije nagrade uhvatiti određena ponašanja kao što je minimizacija rizika, odnosno ponašanja koja uključuju izbor akcija koje nisu najbolje u proseku ali minimizuju verovatnoću ostvarivanja najgoreg mogućeg ishoda.

Definisanje nagrade predstavlja jedan od najtežih zadataka u učenju potkrepljivanjem. Naime, potrebno ju je definisati tako da ne određuje potprobleme cilja jer, u suprotnom, postoji opasnost da će agent da pronađe način da reši veliki broj potproblema bez da ikad dostigne cilj koji se od njega traži. Na primer, u pucačkim igrama se može desiti da pored glavne mete, do koje nije lako doći, postoje i obični neprijatelji čijom se eliminacijom agent takođe nagrađuje. Ovo može dovesti do situacije da je ukupna narada koja se dobija za eliminaciju regularnih neprijatelja približna nagradi koja se dobija za eliminaciju glavne mete, što može da demotiviše agenta da pronađe glavnu metu. Nagrada treba da predstavlja način kojim se agentu nagoveštava šta treba da uradi, a ne kako da to uradi.

Dobitak i epizoda

Postavlja se pitanje kako formalno definisati ukupnu dugoročnu nagradu? Ako je niz dobijenih nagrada $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$, tada *očekivani dobitak*, u oznaci G_t, predstavlja funkciju niza nagrada. Ta funkcija je u osnovnom obliku definisana sledećim izrazom:

$$G_t \doteq R_{t+1} + R_{t+2} + \cdots + R_T$$

gde je T poslednji korak. U zavisnosti od prirode problema, različite vrste MDP-ova se mogu primenjivati, i u odnosu na putanju koju formiraju, oni se dele na:

- MDP sa konačnim horizontom (epizodični), kod kojih je T slučajna veličina.
- MDP sa beskonačnim horizontom, kod kojih $T \to \infty$.

U nastavku će biti pretpostavljeni i detaljnije opisani epizodični MDP-ovi jer su oni odgovarajući za problem koji se u ovom radu istražuje.

Epizodični MDP-ovi predstavljaju pristup koji ima smisla primeniti kada se u problemu koji MDP opisuje, prirodno može javljati poslednji korak, odnosno kada se interakcije između agenta i okruženja mogu na prirodan način razdvojiti u nezavisne podsekvence, koje nazivamo *epizodama*. Svaka epizoda se završava u specijalnom stanju koje se zove *završno stanje*, nakon čega proces započinje novu epizodu od nekog od početnih stanja okruženja nezavisno od završnog stanja prethodne epizode. Vremenski korak T u kom se epizoda završava, predstavlja slučajnu veličinu čija vrednost varira od epizode do epizode.

Još jedan koncept koji je potrebno uvesti kako bi se očekivani dobitak mogao u potpunosti definisati je *umanjenje*. U ovom slučaju, uzimajući u obzir umanjenje, agent pokušava da maksimizuje zbir umanjenih nagrada koje vremenom dobija. Naime, izborom svojih akcija pokušava da maksimizuje očekivani *umanjeni dobitak* koji se definiše na sledeći način:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{i=1}^{T} \gamma^i R_{t+i+1}$$

gde je γ parametar koji uzima vrednosti $0 \leq \gamma \leq 1$ i zove se *hiperparametar umanjenja*. Uloga ovog parametra je da upravlja time koliko agent uzima u obzir buduće nagrade kada računa dobitak. Što je njegova vrednost bliža jedinici to je agent osetljiviji na budućnost, pri čemu ako je njegova vrednost jednaka jedinici onda agent podjednako uzima u obzir sve buduće nagrade. Sa druge strane, što je

vrednost ovog parametra bliža nuli to je agent usredsređeniji na nagrade u bližoj budućnosti, pri čemu ako je vrednost parametra jednaka nuli agent odbacuje sve nagrade osim neposrednih. Pored toga, ovaj parametar ima važnu ulogu u slučaju beskonačnih epizoda. Naime, dovoljno je da niz nagrada bude ograničen pa će vrednost $\gamma < 1$ obezbediti konvergenciju dobitka. U slučaju epizodičnih MDP-ova prirodno je izabrati $\gamma = 1$.

Postoji sledeća veza između dobitka uzastopnih koraka koja je ključna za teoriju i algoritme učenja potkrepljivanjem:

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Politika i funkcije vrednosti

U zavisnosti od problema koji rešavamo, razlikujemo više vrsta algoritama učenja potkrepljivanjem koje mogu biti primenjena na dati problem. Dva osnovna tipa su algoritmi zasnovani na *funkcijama vrednosti* i algoritmi zasnovani na *politikama*. Većina algoritama učenja potkrepljivanjem obuhvata evaluaciju stanja i akcija, odnosno procenu koliko je poželjno za agenta da se nalazi u nekom stanju ili koliko je poželjno za agenta da preduzme neku akciju u datom stanju. U ovom kontekstu pojam kvaliteta nekog stanja ili akcije definisan je u terminima očekivanog dobitka, pri čemu očekivane nagrade koje agent dobija u budućnosti zavise od akcija koje on preduzima. Stoga, funkcije vrednosti se definišu u odnosu na neke specifične načine delanja koje se zovu politike.

Politika predstavlja preslikavanje stanja u raspodele verovatnoća za svaku od mogućih akcija $a \in \mathcal{A}(s)$ pri svakom mogućem stanju $s \in \mathcal{S}$. Ako agent dela u skladu sa politikom π u trenutku t, tada je $\pi(a|s)$ verovatnoća da je $A_t = a$ ako je $S_t = s$. Centralni problem koji metode učenja potkrepljivanjem rešavaju je definisanje načina kako se politika agenta menja i nalaženje takvih politika na osnovu njegovog iskustva.

Vrednost stanja $s \in S$ pri politici π, u oznaci $v^\pi(s)$, predstavlja očekivani dobitak kada agent, počevši od stanja s, dela u skladu sa politikom π. Funkcija v^π se zove *funkcija vrednosti stanja* i za MDP se može formalno definisati na sledeći način:

$$v^\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \middle| S_t = s\right], \forall s \in \mathcal{S}$$

gde je $\mathbb{E}_\pi$ očekivanje slučajne veličine kada agent prati politiku π, a t je bilo koji vremenski trenutak. Slično, vrednost akcije $a \in \mathcal{A}(s)$ u stanju $s \in \mathcal{S}$ pri politici π, u oznaci q^π, predstavlja očekivani dobitak kada agent, počevši od stanja s prvo preduzme akciju a i zatim dela u skladu sa politikom π. Funkcija q^π se zove *funkcija vrednosti akcije u stanju* i za MDP se formalno može definisati na sledeći način:

$$\begin{aligned} q^\pi(s,a) &\doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right], \forall s \in \mathcal{S} \end{aligned}$$

Razlika izmedu poznavanja funkcije v i funckije q je u tome da funkcija v često ima kompaktniju reprezentaciju, pošto stanja ima manje nego parova stanja i akcija, ali je odlučivanje o akciji koju treba preduzeti komplikovanije na osnovu funkcije v jer funkcija q uključuje tu akciju, dok je u slučaju korišćenja funkcije v potrebno vršiti pretragu po akcijama koje je moguće preduzeti u potrazi za najboljim budućim stanjem.

Osnovno svojstvo funkcija vrednosti koje se javljaju u učenju potkrepljivanjem je da imaju rekurzivne odnose slične onom koji je već pokazan u izrazu za računanje dobitka. Naime, za bilo koju politiku π i bilo koje stanje s, sledeće tvrđenje važi za vrednost stanja s i vrednost njegovih mogućih sledbenika:

$$\begin{aligned} v^\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big[r + \gamma v^\pi(s')\Big], \forall s \in \mathcal{S}, \end{aligned}$$

gde je akcija $a \in \mathcal{A}(s)$ i sledeće stanje $s' \in \mathcal{S}$. Prethodni izraz se zove *Belmanova jednačina* za funkciju vrednosti stanja i ima ključnu ulogu u algoritmima učenja potkrepljivanjem. Ova jednačina opisuje odnos između tekućih stanja i stanja koja slede. Belmanova jednačina računa srednju vrednost svih mogućih ishoda tako što pomnoži vrednost svakog ishoda sa verovatnoćom njegovog pojavljivanja. Ona tvrdi da vrednost početnog stanja mora biti jednak zbiru očekivane nagrade i umanjene vrednosti očekivanog sledećeg stanja. Analogno prethodnom se može izvesti i Belamnova jednačina za funkciju vrednosti akcije u stanju:

$$
\begin{aligned}
q^{\pi}(s,a) &\doteq \mathbb{E}_{\pi}[G_t|S_t=s,A_t=a] = \mathbb{E}_{\pi}[R_{t+1}+\gamma G_{t+1}|S_t=s,A_t=a] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r+\gamma \mathbb{E}_{\pi}[G_{t+1}|S_t=s']\Big] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r+\gamma \sum_{a'} \pi(a'|s')q^{\pi}(s'|a')\Big], \forall s\in\mathcal{S}, a\in\mathcal{A}(s)
\end{aligned}
$$

Optimalne politike i funkcije vrednosti

Kao što je već napomenuto, centralni problem učenja potkrepljivanjem predstavlja nalaženje politika koje, dugoročno posmatrano, maksimizuju ukupnu akumuliranu nagradu. Kako bi optimalna politika mogla precizno da se definiše, potrebno je najpre definisati relaciju parcijalnog uređenja nad politikama. Politika π je bolja ili jednaka politici π' ako je njen očekivani dobitak veći ili jednak očekivanom dobitku politike π' u svakom stanju $s \in \mathcal{S}$. Odnosno, $\pi \geq \pi'$ ako i samo ako je $v^{\pi}(s) \geq v^{\pi'}(s)$, za svako $s \in \mathcal{S}$. Uvek postoji bar jedna politika koja je bolja ili jednaka svim ostalim politikama. Te politike se nazivaju *optimalnim politikama*, i označavaju se sa π^*. Optimalne politike dele istu funkciju vrednosti stanja koja se zove *optimalna funkcija vrednosti stanja*. Ova funkcija se označava sa v^* i definiše se sledećim izrazom:

$$v^*(s) \doteq \max_{\pi} v^{\pi}(s)$$

za svako $s \in \mathcal{S}$. Optimalne politike takođe dele istu funkciju vrednosti akcije u stanju koja se zove *optimalna funkcija vrednosti akcije u stanju*. Označava se sa q^* i ima sledeći oblik:

$$q^*(s,a) \doteq \max_{\pi} q^{\pi}(s,a)$$

za svako $s \in \mathcal{S}$ i $a \in \mathcal{A}(s)$. Za par stanja i akcije (s,a), ova funkcija vraća očekivani dobitak pri preduzimanju akcije a u stanju s i nakon toga pri delanju u skladu sa optimalnom politikom. Stoga, izraz koji definiše q^* se može napisati u terminima funkcije v^*:

$$q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1})|S_t=s, A_t=a].$$

Pošto je v^* funkcija vrednosti neke politike, ona mora da zadovolji uslov konzistentnosti definisan Belmanovom jednačinom za vrednosti stanja. Uzimajući u obzir da je ona optimalna, njen izraz uslova konzistentnosti se može napisati u specijalnom obliku bez referisanja na bilo koju politiku. Počevši od činjenice da

je vrednost stanja pri optimalnoj politici jednaka očekivanom dobitku najbolje akcije u tom stanju, može se izvesti Belmanova jednačina za optimalnu funkciju vrednosti stanja:

$$\begin{aligned} v^*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi^*}(s,a) = \max_a \mathbb{E}_{\pi^*}[G_t | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v^*(s')]. \end{aligned}$$

Slično se može uraditi i za q^*. Belmanova jednačina za optimalnu funkciju vrednosti akcije u stanju je:

$$\begin{aligned} q^*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q^*(S_{t+1}, a') \Big| S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma \max_{a'} q^*(s', a')\Big] \end{aligned}$$

Belmanove jednačine su suštinski važne za problem učenja potkrepljivanjem jer pružaju vezu između trenutnih stanja i akcija i njihovih sledbenika u MDP-ovima. Ovo omogućava da se, pomoću veze u strukturi formulacije problema, suma koja opisuje dati MDP redukuje na jednostavan sistem jednačina. Kod konačnih MDP-ova, optimalne Belmanove jednačine čine sistem nelinearnih jednačina, po jedna za svako stanje ili par stanja i akcije. Ovaj sistem ima jedinstveno rešenje, i ako je dinamika okruženja poznata, onda može biti rešen bilo kojom metodom za rešavanje sistema nelinearnih jednačina.

Uvek postoji bar jedna akcija pri kojoj optimalna Belmanova jednačina ima maksimalnu vrednost. Tada optimalna politika dodeljuje vrednost različitu od nule samo tim akcijama. Ako su optimalne funkcije vrednosti poznate, da bi se odredila optimalna politika, dovoljno je definisati politiku koja je pohlepna u odnosu na ove funkcije:

$$\pi^*(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma v^*(s')]$$

$$\pi^*(s) = \text{argmax}_a q^*(s, a)$$

Kako Belmanove jednačine uzimaju u obzir sva buduća ponašanja, pohlepan pristup je dovoljan za određivanje optimalne politike, jer izbor akcija na osnovu njihovih kratkoročnih posledica vodi do optimalnog ponašanja na duge staze.

Tabelarne metode dinamičkog programiranja predstavljaju još jedan način računanja funkcija vrednosti, koji je možda i najsrodniji Belmanovim jednačinama. Međutim, i u ovom slučaju postoje ograničenja, kako računska tako i druga, koja u praksi otežavaju rešavanje problema učenja potkrepljivanjem. Zbog toga se učenje potkrepljivanjem u praksi najčešće rešava pomoću funkcionalne aproksimacije, koja će biti objašnjena kasnije. Veliki broj metoda učenja potkrepljivanjem se može rešiti aproksimacijom rešenja Belmanove jednačine upotrebom velikog broja primera stanja i prelaska između njih koja su se zaista dogodila umesto da se koristi funkcija prelaska između stanja koja najčešće i nije poznata.

2.2 Učenje u nepoznatom okruženju

Kako bi konvergencija ka pravim funkcijama vrednosti bila zagarantovana u nepoznatom okruženju, neophodno je da agent na neki način skuplja informaciju iz celog prostora stanja i akcija. Potrebno je podsticati istraživanje tokom celog procesa učenja, što se može postići na dva načina. Jedan je pomoću metoda učenja *u skladu sa politikom*, koje pokušavaju da evaluiraju ili ažuriraju politiku koja se istovremeno korisiti za donošenje odluka. Drugi je pomoću metoda učenja *mimo politike*, koje donose odluke na osnovu politika različitih od onih koje pokušavaju da poprave. Oba ova pristupa imaju svoje prednosti i ograničenja koja će se detaljnije razmatrati u nastavku, pri čemu će se, zbog potrebe istraživanja, više pažnje posvetiti metodama učenja mimo politike.

Ocena vrednosti akcija

Ako model nije dostupan, onda je naročito korisno oceniti vrednosti parova stanja i akcija umesto vrednosti stanja. Kada je model dostupan, vrednosti stanja su dovoljne da bi se izračunala politika, jer je dovoljno samo proveriti jedan korak ispred i izabrati akciju koja vodi do najpovoljnije kombinacije nagrade i narednog stanja. Sa druge strane, ako model nije dostupan, određivanje politike samo na osnovu vrednosti stanja nije dovoljno jer svaka akcija mora eksplicitno da se evaluira kako bi vrednosti mogle da se koriste pri izvođenju politike. Prema tome, glavni cilj metoda koje uče u nepoznatom okruženju jeste ocena funkcije q^*.

Par stanja i akcije (s, a) se smatra posećenim ako se u nekom koraku učenja okruženje nalazi u stanju s i agent preduzima akciju a. Pošto se u slučaju učenja

u nepoznatom okruženju vrednosti akcija u stanju ne računaju pomoću funkcije p, već se to radi pomoću uzoraka iskustva dobijenih u hodu tokom učenja, neophodno je obezbediti da svaki par stanja i akcije bude posećen beskonačan broj puta kako bi konvergencija ka pravim vrednostima tih akcija bila zagarantovana. Jedan problem kod determinističkih politika je to što veliki broj parova akcija i stanja nikad ne bude posećen. Kako pri determinističkim politikama u istom stanju agent bira istu akciju, ostale neposećene akcije nikad neće biti ažurirane. Ovo predstavlja ozbiljan problem jer je svrha učenja vrednosti akcija da se akcije uporede i izabere najbolja od ponuđenih pri trenutnom stanju. Stoga je ključno definisati mehanizam koji obezbeđuje istraživanje celog prostora stanja i akcija tokom celog procesa učenja.

Eksploracija i eksploatacija

Ako se računaju vrednosti akcija, onda u svakom koraku postoji akcija čija je trenutna vrednost veća od vrednosti drugih akcija. Ove akcije se nazivaju *pohlepnim* akcijama. Kada se izaberu pohlepne akcije tada se vrši *eksploatacija* trenutnog znanja o vrednostima akcija. Inače, ako se izaberu akcije koje nisu pohlepne tada se vrši *eksploracija* (*istraživanje*), jer se time omogućava da se potencijalno popravi trenutna vrednost izabrane akcije.

Eksploatacija je pravi izbor ako je cilj maksimizovati očekivanu nagradu nakon jednog koraka, ali eksploracijom se, dugoročno gledano, može doći do većeg ukupnog dobitka. Na primer, ako je sa sigurnošću poznata vrednost pohlepne akcije, pri čemu još nekoliko akcija imaju vrednosti približne pohlepnoj, ali sa određenim stepenom nepouzdanosti. Tada postoji mogućnost da je neka od nepohlepnih akcija zapravo bolja od trenutne pohlepne akcije. Ako postoji još puno koraka do završetka epizode, onda je možda bolje istražiti nepohlepne akcije i saznati koja od njih je bolja od trenutne pohlepne. U početku bi zbog eksploracije nagrada bila niska, ali bi vremenom rasla zbog toga što bi vrednije akcije koje su otkrivene mogle biti eksploatisane više puta u budućnosti. Pošto nije moguće istovremeno primeniti eksploraciju i eksploataciju na nivou pojedinačnih akcija, uvek će postojati konflikt između ove dve strategije.

Nagodba između eksploracije i eksploatacije je složen koncept i način na koji se primenjuje u većini slučajeva zavisi od više parametara. Pre svega zavisi od tačnosti ocena stanja i akcija, njihove pouzdanosti i broja preostalih koraka do završetka učenja. U nastavku će biti predstavljena jedna od tehnika istraživanja.

ε-pohlepne politike

Jedan od mehanizama koji obezbeđuju eksploraciju tokom učenja je da se na početku epizode nasumično izaberu stanje i akcija. Ovaj pristup garantuje da će svaki par stanja i akcije biti posećen beskonačan broj puta kako se broj epizoda približava beskonačnosti. Međutim, u praksi se u opštem slučaju ne može primeniti ovakvo rešenje, naročito kada je u pitanju interakcija sa stvarnim okruženjem kao što je autonomno upravljanje automobila. Najprimenjenija alternativa nasumičnim početnim stanjima koja obezbeđuje posećivanje svakog mogućeg para stanja i akcije je upotreba stohastičke politike.

Najjednostavnije pravilo izbora akcija je izabrati najvredniju (pohlepnu) akciju. Ako postoji više takvih akcija, onda se može nasumično izabrati jedna od njih. Pristup pohlepnog izbora akcija uvek eksploatiše trenutno znanje o funkciji vrednosti akcija, odnosno nijedan korak učenja se ne potroši na uzorkovanje prividno suboptimalnih akcija kako bi se proverilo da li će vremenom dovesti do poboljšanja. Jednostavna alternativa ovom pristupu bi bilo da se politika ponaša pohlepno samo u većini slučajeva, a sa vremena na vreme, recimo sa verovatnoćom ε, izabere nasumično bilo koju akciju, bez obzira na njihovu vrednost, u odnosu na uniformnu raspodelu. Politike koje biraju akcije približno pohlepno se nazivaju *ε-pohlepnim politikama*. Glavna prednost ovih politika je da obezbeđuju da svaka akcija bude izabrana beskonačan broj puta kako se broj koraka približava beskonačnosti, što je neophodan uslov da bi q^π konvergirala ka q^*. Ovo naravno povlači da verovatnoća izbora optimalne akcije konvergira ka $1-\varepsilon$, odnosno blizu sigurnom događaju. Ovo su samo asimptotske ocene koje ne govore ništa o tome kakva je efektivnost ovih politika u praksi.

Postoji klasa politika koje obezbeđuju da je $\pi(a|s) > 0$ za svako $s \in \mathcal{S}$ i $a \in \mathcal{A}(s)$. U pitanju su *ε-meke* politike koje istovremeno garantuju eksploraciju i koje se tokom učenja približavaju deterministčkim politikama. Podskup ovih politika čine ϵ-pohlepne politike koje omogućavaju da se veći deo vremena biraju pohlepne akcije ali i da ostale akcije takođe budu izabrane sa verovatnoćom koju određuje parametar ε. Odnosno, svakoj akciji koja nije pohlepna je pridružena verovatnoća $\frac{\varepsilon}{|\mathcal{A}(s)|}$, dok je preostala verovatnoća $1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|}$ pridružena samo pohlepnoj akciji. Od svih ε-mekih politika, ε-pohlepne su najbliže pohlepnim politikama kod kojih je $\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}$ za svako stanje i akciju, za neko $\varepsilon > 0$.

2.3 TD učenje

Čest je slučaj u praksi da model koji opisuje dinamiku okruženja nije poznat. Srećom, postoji cela klasa algoritama koja se u praksi dobro nosi sa ovim ograničenjem, i koja je ujedno i centralna oblast učenja potkrepljivanjem. U pitanju je oblast *učenja zasnovanog na vremenskim razlikama* (eng. *temporal-difference learning*, u nastavku TD učenje).

TD učenje predstavlja kombinaciju ideja prisutnih u metodama koje uče na osnovu neposredne interakcije sa okruženjem i algoritmima dinamičkog programiranja. Naime, TD metode mogu da uče direktno iz skupa iskustava bez ikakve potrebe za modelom okruženja. Sa druge strane, kao u metodama dinamičkog programiranja, TD metode se oslanjaju na tehnike *samoaproksimacije* vrednosti (eng. *bootstrap*) koje računaju vrednosti stanja na osnovu ocena vrednosti drugih stanja.

U nastavku biće predstavljena uopštena procedura na kojoj se zasniva konstrukcija algoritama učenja potkrepljivanjem zasnovanih na vrednostima, među kojima su i algoritmi TD učenja. Takođe biće predstavljena dva potproblema ove procedure: problem *predviđanja*, odnosno evaluacija stanja i akcija u odnosu na neku fiksiranu politiku i zatim problem *upravljanja* odnosno nalaženje optimalne politike kroz naizmeničnu evaluaciju i popravljanje.

Uopštena iteracija politike

Problem evaluacije politike, poznatiji kao problem predviđanja, u metodama zasnovanim na funkcijama vrednosti se svodi na računanje funkcija vrednosti za proizvoljnu politiku π. Kako bi se funkcije vrednosti izračunale, potrebno je najpre pretvoriti Belmanove jednačine posmatranih funkcija vrednosti u odgovarajuća pravila ažuriranja. U zavisnosti od funkcije vrednosti koja se računa i od načina kako se računa vrednost sledećeg stanja, postoje različite vrste izraza za ažuriranje. Na primeru računanja funkcije q^π dinamičkim programiranjem, to se postiže iterativnom aproksimacijom niza vrednosti $q_{k+1}(s,a)$, pri čemu je inicijalna vrednost q_0 proizvoljna, a svaka sledeća aproksimacija se računa pomoću odgovarajuće Belmanove jednačine za q^π na sledeći način:

$$\begin{aligned} q_{k+1}(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma q_k(S_{t+1}, A_{t+1}) \Big| S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma \sum_{a'} \pi(a'|s') q_k(s', a')\Big] \end{aligned}$$

za svako $s \in \mathcal{S}$ i $a \in \mathcal{A}(s)$, pri čemu su u ovom slučaju politika π i njena funkcija q^π fiksirane u ovom izrazu, a niz $\{q_k\}$ u opštem slučaju konvergira ka q^π kako $k \to \infty$ pod istim uslovima koji garantuju postojanje vrednosti q^π. Ovaj postupak se zove *iterativna evaluacija politike*.

Glavni razlog zašto se računaju funkcije vrednosti je da bi se unapredile postojeće politike. Neka je utvrđena vrednost funkcije v^π za proizvoljnu deterministička politiku π. Postavlja se pitanje da li je za neko stanje s bolje izabrati neku akciju $a \neq \pi(s)$? Jedan način kako se to može utvrditi je da se u stanju s izabere akcija a i nakon toga nastaviti u skladu sa politikom π. Izraz koji opisuje ovo ponašanje je sledeći:

$$\begin{aligned} q^\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v^\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v^\pi(s')\Big] \end{aligned}$$

pri čemu ako je on veći od v^π, odnosno ako je bolje izabrati akciju a u stanju s i zatim nastaviti u skladu sa π nego sve vreme pratiti politiku π, onda je bolje izabrati a svaki put kada se naiđe na s. Drugim rečima, nova politika je bolja od inicijalne pri svakom stanju.

Prethodno tvrđenje predstavlja specijalan slučaj *teoreme o popravljanju politike* [44]. Neka su π i π' deterministčke politike takve da za svako $s \in \mathcal{S}$ važi:

$$q^\pi(s, \pi'(s)) \geq v^\pi(s).$$

Tada je politika π' bolja ili jednaka politici π, odnosno mora da proizvodi dobitak veći ili jednak dobitku koji proizvodi politika π za svako stanje $s \in \mathcal{S}$:

$$v^{\pi'}(s) \geq v^\pi.$$

pri čemu, ako postoji stroga nejednakost u nekom stanju u prvom izrazu, tada ona mora da postoji i u drugom.

Na sličan način deterministička pohlepna politika uzima u obzir samo najvrednije akcije u odnosu na funkciju v^π. Po definiciji, pohlepna politika zadovoljava uslove teoreme o popravljanju politike. Stoga je ona bolja ili jednaka originalnoj politici. Proces kreiranja nove politike koja predstavlja unapređenje neke originalne politike i koja je pohlepna u odnosu na funkciju vrednosti originalne politike se zove *popravljanje politike*.

Nakon što je politika π prepravljena u bolju politiku π' na osnovu v^π, postupak smenjivanja procesa evaluacije i popravljanja se može ponoviti sve dok se ne dođe do optimalne politike. Svaka dobijena politika predstavlja strogo poboljšanje u

odnosu na prethodnu osim ako je prethodna politika već optimalna. Pošto konačni MDP-ovi imaju konačno mnogo različitih politika, ovaj proces sa sigurnošću konvergira ka optimalnoj politici. Ovaj proces nalaženja optimalne politike se zove *iterativno popravljanje politike*. Operacije evaluacije i popravljanja deluju naizmenično, pri čemu one mogu i da se prepliću, to jest da jedna počne pre nego što se druga završi. Preplitanje se može vršiti na više načina i u zavisnosti od granularnosti koja određuje učestalost smenjivanja ovih procesa, mogu se definisati više vrsta iteracije politike. Koncept koji objedinjuje sve ove varijante nezavisno od granularnosti i drugih detalja procesa iteracije politike se zove *uopštena iteracija politike* (eng. *generalized policy iteration*, GPI).

Tokom izvršavanja iterativnog popravljanja politike, procesi evaluacije i popravljanja međusobno interaguju. Evaluacija čini da je funkcija vrednosti u skladu sa trenutnom politikom, dok popravljanje politiku čini pohlepnom u odnosu na trenutnu funkciju vrednosti. Ove operacije se mogu posmatrati istovremeno kao suprotstavljene i sarađujuće. Prepravljanje politike u pohlepnu u odnosu na funkcije vrednosti obično čini da funkcije vrednosti više ne odgovaraju politici, dok prepravka funkcija vrednosti obično čini da politika više nije pohlepna. Kada se oba procesa stabilizuju i ne proizvode više izmene tada politika i funkcije vrednosti postaju optimalne. Funkcije vrednosti postaju stabilne kada odgovaraju trenutnoj politici, dok politika postaje stabilna kada je pohlepna u odnosu na trenutne funkcije vrednosti. Dakle, oba procesa postaju stabilna samo kada je pronađena politika koja je pohlepna u odnosu na svoje funkcije vredosti. Ovo povlači da su Belmanove jednačine za optimalne funkcije vrednosti zadovoljene i stoga su pronađene funkcije vrednosti i politka optimalne.

TD predviđanje

Kao što je već napomenuto, problem predviđanja se kod TD metoda rešava na osnovu neposredne interakcije agenta sa okruženjem. Dakle, prateći politiku π, popravljaju se ocene funkcije vrednosti q^π za nezavršna stanja S_t i akcije A_t koje se javljaju u izabranim primerima iskustva. Problem predviđanja može da se odnosi na bilo koju od funkcija vrednosti. Osnovni oblik pravila ažuriranja za funkciju vrednosti akcije u stanju TD metoda je:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

koje se primenjuje odmah nakon prelaska u stanje S_{t+1} i prihvatanja nagrade R_{t+1}. Razlika između ocene vrednosti akcije A_t u stanju S_t i ciljne vrednosti $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ ima važnu ulogu u algoritmima TD učenja. Ova vrednost se zove *TD greška* i javlja se u različitim oblicima, ali njen osnovni oblik je sledeći:

$$\delta_t \doteq R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

pri čemu je ona dostupna tek nakon sledećeg koraka jer se računa na osnovu sledećeg stanja i nagrade. To jest, δ_t je dostupna tek nakon koraka $t + 1$.

Familija ovih TD metoda se zove *TD(0)* ili *jednokoračne* TD metode, jer predstavljaju specijalan slučaj uopštenja $TD(\lambda)$ i *n-koračnih* TD metoda. U gornjem izrazu metode TD(0) se takođe može videti da se tekuće vrednosti akcija i stanja računaju na osnovu ocena vrednosti drugih akcija i stanja na isti način kao u slučaju samoaproksimacije vrednosti dinamičkog programiranja. Upravo je ova dostupnost ciljnih vrednosti za ažuriranje funkcija vrednosti nakon jednog koraka to što omogućava TD metodama da uče u toku epizode. Prema tome, jasno se može videti moć ovih metoda koje u isto vreme nude mogućnost učenja u nepoznatom okruženju, efikasnost učenja u toku epizode i fleksibilnost u izboru broja koraka nakon kog će se vršiti ažuriranje funkcija vrednosti.

Q-učenje: TD upravljanje mimo politike

Za razliku od metoda učenja u skladu sa politikom koje uče politiku približnu optimalnoj koja ujedno i garantuje eksploraciju, metode učenja mimo politike koriste dve politike. *Ciljnu politiku* koja se uči i postaje optimalna i *politiku ponašanja* koja se koristi za izbor akcija. Prednost ovakve podele je u tome što ciljna politika može da bude deterministička, kao što je na primer pohlepna politika, dok se na osnovu politike ponašanja može obezbediti izbor svih mogućih akcija, kao što je u slučaju ε-pohlepnih politika. Ovaj pristup takođe mora zadovoljiti svojstvo *pokrivenosti*, odnosno da politika ponašanja ima verovatnoću različitu od nule za izbor akcija koje takođe mogu biti izabrane pri delanju na osnovu ciljne politike. Kako bi ovo svojstvo bilo ispunjeno, dovoljno je da politika ponašanja bude ε-meka.

Metode učenja mimo politike su opštije. Uključuju metode učenja u skladu sa politikom kao specijalan slučaj kod kojih su ciljna politika i politika ponašanja iste i za razliku od njih mogu da nauče politike koje su zaista optimalne. Metode učenja mimo politike imaju i značajnijih primena u praksi, kao što je u slučaju kada je potrebno da izbor akcija vrše ljudski eksperti. Sve ove prednosti koje nude

metode učenja mimo politike imaju svoju cenu. Složenije su i zahtevaju definisanje dodatnih tehnika kao što je *uzorkovanje prema važnosti*. Takođe, pošto metode učenja mimo politike koriste podatke generisane pomoću politike različite od one koju uče, često su nestabilnije i sporije konvergiraju od metoda učenja u skladu sa politikom.

Jedan od prvih prodora u oblasti učenja potkrepljivanjem je bio razvoj algoritma TD učenja mimo politike koji se zove Q-učenje [50]. Kada je u pitanju problem upravljanja, za konstrukciju algoritma koristi se uobičajeni postupak koji je definisan procedurom GPI, s tim što se u ovom slučaju koriste TD metode za evaluaciju vrednosti stanja i akcija umesto metoda dinamičkog programiranja.

Pošto se u metodama TD učenja mimo politike za definisanje odgovarajućih MDP-ova koriste prelazi između parova stanja i akcija, prvi korak je definisanje pravila ažuriranja za funkciju vrednosti akcija u stanju. Potrebno je proceniti $q^{\pi}(s, a)$ za trenutnu ciljnu politiku π za svako stanje s i akciju a, pri čemu teoreme koje garantuju konvergenciju vrednosti stanja u metodi predviđanja TD(0) takođe važe i za vrednosti akcija u stanju. Pravilo ažuriranja koje definiše problem predviđanja vrednosti akcjia u stanju je sledećeg oblika:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a(S_{t+1}, a) - Q(S_t, A_t)].$$

Ovaj izraz se primenjuje nakon svakog prelaska iz nezavršnog stanja S_t. Ako je S_t završno, onda je $Q(S_t, a)$ po definiciji nula.

Funkcija vrednosti akcije u stanju Q, koju algoritam uči, direktno aproksimira optimalnu funkciju vrednosti akcije u stanju q^*, pri čemu se tokom učenja prati neka nezavisna politika. Politika koju algoritam uči i dalje ima uticaj na posećenost stanja i akcija i koje vrednosti će biti ažurirane. Međutim, sve što je potrebno da bi konvergencija bila zagarantovana jeste da svaki par stanja i akcije bude posećen bez prestanka. Pod pretpostavkom da korak učenja α zadovoljava uslove konvergencije pod kojima je definisana metoda optimizacije, funkcija Q konvergira ka optimalnoj funkciji vrednosti sa verovatnoćom 1. Na slici 2.1 je prikazan pseudokod algoritma Q-učenje.

Naročito je zgodno smanjivati vrednosti parametra α tokom učenja kako bi se obezbedila konvergencija. Neka $\alpha_n(a)$ označava korak učenja nakon n-tog izbora akcije a i neka je $\alpha_n(a) = \frac{1}{n}$. Tada niz $\{\alpha_n(a)\}$ zadovoljava uslove zadate teorijom stohastičke aproksimacije koji garantuju konvergenciju sa verovatnoćom 1:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad \text{i} \quad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty$$

Slika 2.1: Q-učenje (TD upravljanje mimo politike)

Nasumično inicijalizovati $Q(s, a)$, za svako $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$
Inicijalizovati $Q(terminal, \cdot) = 0$
for *i=1,M* **do**
  Inicijalizovati S
  for *t=1,T* **do**
    Izabrati A na osnovu S koristeći politiku izvedenu iz Q (na primer ε-pohlepnu)
    Preduzeti akciju A i opaziti R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  end
end

Prvi uslov je neophodan kako bi se obezbedila dovoljna veličina koraka koja omogućava dostizanje optimuma, dok drugi uslov obezbeđuje da će vremenom koraci postati dovoljno mali da bi konvergencija bila zagarantovana.

2.4 Funkcionalna aproksimacija

Jedan način nalaženja optimalne politike jeste eksplicitno rešavanje sistema jednačina definisaih optimalnom Belmanovom jednačinom. Međutim, ovo rešenje je retko kad korisno jer podrazumeva pretragu po svim mogućim ishodima i računanje verovatnoće njihovog pojavljivanja kao i njihove vrednosti u terminima dobitka. Dodatno, ovo rešenje se oslanja na tri pretpostavke koje su u praksi retko istovremeno zadovoljene:

- Funkcija verovatnoće koja opisuje dinamiku okruženja je poznata.
- Na raspolaganju postoji dovoljno računarskih resursa.
- Markovljevo svojstvo je zadovoljeno.

Na primer, iako su u društvenim igrama poput šaha i bekgemona zadovoljene prva i treća pretpostavka, nedostatak računarskih resursa obično čini da računanje Belmanovih jednačina za funkcije v^* i q^* zahteva i do hiljade godina na najsavremenijim računarima. Od velikog značaja je količina računarske snage sa kojom agent raspolaže, pre svega količina koja je neophodna za izvođenje jednog koraka učenja.

Količina memorije koja je dostupna agentu takođe predstavlja važno ograničenje. Velika količina memorije je često potrebna za računanje aproksimacija funkcija vrednosti, politika i modela. U problemima sa malim, konačnim skupovima stanja, ove aproksimacije se mogu predstaviti pomoću nizova ili tabela čiji su elementi vrednosti stanja ili parova stanja i akcija. Ovaj način predstavljanja skupova stanja i akcija se naziva tabelarnom reprezentacijom, a metode koje se koriste za rešavanje problema koji uključuju tabelarne reprezentacije kao što su dinamičko programiranje ili osnovni oblici TD učenja se nazivaju tabelarnim metodama. U mnogim slučajevima koji su od praktičnog značaja, broj mogućih stanja je daleko veći od bilo kakve memorije koja može biti napravljena. Poređenja radi, jedna igra bekgemona, koja je pritom daleko jednostavnija od bilo kog problema u praksi, ima čak 10^{20} mogućih stanja. Zbog toga se u ovim slučajevima funkcije moraju aproksimirati pomoću neke kompaktne parametrizovane reprezentacije.

Problem sa velikim skupovima stanja nisu samo vreme i memorija neophodna za računanje i čuvanje vrednosti stanja. U velikom broju slučajeva skoro svako stanje na koje se nailazi biće neposećeno. Kako bi se donosile smislene odluke u tim stanjima potrebno je izabrati akcije na osnovu već posećenih stanja koja su različita, ali u isto vreme dovoljno slična trenutnom stanju. Dodatno, skoro sigurno će postojati veliki broj stanja koja imaju toliko malu verovatnoću pojavljivanja da izbor suboptimalnih akcija u njima ne bi imao značajan uticaj na ukupan dobitak. Dakle, potrebno je aproksimirati optimalnu politiku na takav način da se u stanjima sa velikom verovatnoćom pojavljivanja izaberu optimalne akcije po cenu toga da se više greši u stanjima koja se retko javljaju. Drugim rečima potrebno je rešiti ključni problem koji obuhvata sva prethodna ograničenja. Taj problem se zove *generalizacija*.

Postoji više vrsta metoda za postizanje generealizacije, ali metoda koja je značajna za učenje potkrepljivanjem je *funkcionalna aproksimacija*. Funkcionalna aproksimacija je tehnika koja se koristi u mnogim oblastima nadgledanog učenja. U principu, bilo koja metoda nadgledanog učenja se uz odgovarajuće modifikacije može koristiti kao funkcionalni aproksimator za problem učenja potkrepljivanjem, ali u praksi neke metode odgovaraju ovom problemu više od drugih. Učenje potkrepljivanjem u kombinaciji sa funkcionalnom aproksimacijom proizvodi veći broj otežavajućih okolnosti koje obično nisu prisutne u nadgledanom učenju kao što je nestacionarnost, računanje vrednosti stanja na osnovu ocena vrednosti drugih stanja i odložene ciljne vrednosti.

Neuronske mreže i duboko učenje

Iako je učenje potkrepljivanjem imalo neke uspehe u prošlosti [45, 40, 25, 35], prethodnim pristupima je nedostajala skalabilnost što je znatno otežavalo primenu na visokodimenzione probleme. Ova ograničenja postoje zato što algoritmi učenja potkrepljivanjem dele iste probleme složenosti kao i drugi algoritmi mašinskog učenja kao što su memorijska i vremenska složenost i složenost uzorka [43]. Uspon takozvanog *dubokog učenja* je, oslanjajući se na moćne tehnike nelinearne aproksimacije funkcija poznatijih kao *neuronske mreže*, omogućavao prevazilaženje ovih problema.

Neuronske mreže su trenutno jedan od najpopularnijih i najbrže rastućih pristupa rešavanju problema mašinskog učenja, podstičući napredak u oblastima u kojima je praktična primena teška, od detekcije objekata na slikama i prepoznavanja govora do automatskog prevođenja [29]. Neuronske mreže nude prirodna proširenja dobro poznatih tehnika linearne i logističke regresije u cilju učenja nelinearnih funkcija predviđanja koje mogu da opišu veliki broj stvarnih procesa i problema, pod uslovom da je na raspolaganju dovoljna količina podataka i odgovarajući skup atributa.

Postoji više vrsta neuronskih mreža i one se dele u odnosu na strukturu njihovih slojeva. Tri najprimenjenija tipa neuronskih mreža su *potpuno povezane*, *konvolutivne* i *rekurentne*. Osnovni i najjednostavniji oblik predstavljaju potpuno povezane neuronske mreže. Za obradu slika najpopularnije su konvolutivne neuronske mreže, dok se rekurentne neuronske mreže najviše koriste za obradu podataka u obliku nizova promenljivih dužina. Broj slojeva u okviru mreže kao i njihov tip može da varira u zavisnosti od primene, ali je upotreba neuronskih mreža koje sadrže skrivene slojeve to što razlikuje duboko učenje od ostatka oblasti mašinskog učenja.

Pored toga što su neuronske mreže univerzalni aproksimatori [11, 22, 51, 39], to jest što se svaka neprekidna funkcija može proizvoljno dobro aproksimirati neuronskom mrežom sa jednim skrivenim slojem i konačnim brojem neurona, njihova moć izražavanja takođe raste sa povećavanjem broja skrivenih slojeva [8]. Drugo važno svojstvo dubokog učenja je da duboke neuronske mreže mogu automatski da pronalaze kompaktne niskodimenzionalne reprezentacije (karakteristike) visokodimenzionalnih podataka kao što su slike, tekst ili zvučni zapisi, što je od velikog značaja za učenje potkrepljivanjem gde često nije moguće primeniti osnovne metode zbog prokletstva dimenzionalnosti.

Duboko učenje je na sličan način ubrzalo napredak i u oblasti učenja potkrepljivanjem. Nudi generalizaciju, skalabilnost i kompaktnu reprezentaciju kao rešenja za sva pomenuta ograničenja osnovnih metoda učenja potkrepljivanjem. Upotreba algoritama dubokog učenja u okviru učenja potkrepljivanjem definiše oblast koja se zove *duboko učenje potkrepljivanjem* (eng. *deep reinforcement learning*).

Potpuno povezane neuronske mreže

Potpuno povezanu neuronsku mrežu čine slojevi sastavljeni od jedninica koje se zovu *neuroni*. U potpuno povezanoj arhitekturi svaki neuron nekog sloja je povezan sa svim neuronima susednih slojeva, pri čemu su vezama između neurona pridružene vrednosti koje se zovu *težine*. Izlaz neurona predstavlja nelinearnu transformaciju primenjenu nad linearnom kombinacijom izlaza neurona prethodnog sloja i težina veza pomoću kojih je povezan sa prethodnicima. Ta nelinearna transformacija se zove *aktivacija*.

Kod potpuno povezanih arhitektura razlikujemo dve vrste slojeva: *skriveni* i *izlazni* slojevi. Skriveni slojevi su oni kod kojih neuroni računaju svoje izlaze na osnovu izlaza neurona prethodnog sloja i prosleđuju ih neuronima narednog sloja. Vrednosti koje računaju skriveni slojevi na neki način predstavljaju nove atribute objekta koji se prosleđuje mreži na ulazu. Drugim rečima, svaki sloj nadograđuje se nad vrednostima prethodnih slojeva i time se konstruišu sve složeniji atributi koji se koriste za računanje aproksimacije ciljne funkcije. Izlazni sloj predstavlja poslednji sloj mreže gde se dobijaju izlazne vrednosti aproksimacije ciljne funkcije koju mreža aproksimira. Broj jedinica ovog sloja i tip njihove aktivacione funkcije određuju vrsta predviđanja koju mreža vrši, kao i broj i tip izlaznih vrednosti ciljne funkcije.

U nastavku je prikazan matematički model potpuno povezane neuronske mreže:

$$
\begin{aligned}
&h_0 \doteq x \\
&h_i \doteq g(W_i h_{i-1} + w_{i0}) \qquad i = 1, 2, \ldots, L
\end{aligned}
$$

gde je x vektor ulaznih promenljivih, L je broj skrivenih slojeva, W_i je matrica čija j-ta vrsta predstavlja vektor vrednosti parametara jedinice j u sloju i, w_{i0} predstavlja vektor slobodnih članova linearnih kombinacija koje jedinice $i-tog$ sloja izračunavaju, a g je nelinearna aktivaciona funkcija. Za vektor $\mathbf{v}$, $g(\mathbf{v})$ predstavlja vektor $(g(v_1), g(v_2), \ldots, g(v_m))^T$, gde je m dimenzionalnost vektora. Zbog jednostavnosti skup svih parametara modela biće označen sa $\mathbf{w}$. Važi $f_{\mathbf{w}}(x) = h_L$.

Konvolutivne neuronske mreže

Konvolutivne neuronske mreže predstavljaju arhitekture mreža kod kojih je svojstvo konstrukcije atributa iz sirovog signala najizraženije. Parametri konvolutivne mreže predstavljaju koeficijente filtera koji se koriste za detekciju različitih obrazaca u signalu. Najčešće se koriste u obradi sirovih zapisa poput slika, zvuka i teksta jer u tim sličajevima njihove najbitnije karakteristike dolaze do izražaja.

Konvolutivnu neuronsku mrežu čine konvolutivni, agregacioni i potpuno povezani slojevi. Uobičajena struktura konvolutivne mreže podrazumeva nekoliko konvolutivnih i agregacionih slojeva koji se naizmenično smenjuju nakon čega sledi potpuno povezana komponenta koja vrši predviđanja na osnovu reprezentacija koje su generisane od strane prethodnih slojeva. Iako je ključna operacija koja se vrši u konvolutivnim slojevima zapravo *unakrsna korelacija*, a ne konvolucija, u kontekstu dubokog učenja ne pravi razliku pa se zbog toga ove mreže nazivaju konvolutivnim. Druga operacija koja se obično koristi za izgradnju karakteristika u konvolutivnim neuronskim mrežama jeste *agregacija*. Međutim, poznato je da njena primena u problemu koji se istražuje u okviru ovog rada negativno utiče na učenje pa će zbog toga biti izostavljena.

Formalno se operacije konvolutivnih slojeva na primeru trodimenzionalne konvolucije mogu definisati na sledeći način:

$$
\begin{aligned}
x^l_{ijk} &\doteq \sum_{p=0}^{m^l-1} \sum_{q=0}^{m^l-1} \sum_{r=1}^{C^{l-1}} w^{l-1}_{pqr} y^{l-1}_{(i+p)(j+q)r} \\
y^l_{ijk} &\doteq g(x^l_{ijk} + w^l_{0k}) \\
l &= 1, 2, \ldots, L \\
i &= 1, 2, \ldots, H^l \\
j &= 1, 2, \ldots, W^l \\
k &= 1, 2, \ldots, C^l
\end{aligned}
$$

gde je L broj konvolutivnih slojeva, H^l i W^l su visina i širina sloja l, C^l je broj kanala (dubina) l-tog sloja, m^l je širina filtera l-tog sloja, w^l_0 predstavlja vektor slobodnih članova linearnih kombinacija koje jedinice l-tog sloja izračunavaju, a g je nelinearna aktivaciona funkcija.

Dve važne operacije koje se mogu primeniti u konvolutivnim slojevima su *proširivanje* (eng. *padding*) i *korak* (eng. *stride*). Ove operacije imaju suprotne efekte na rezultat propagacije kroz konvolutivne slojeve. Proširivanje se koristi

za uvećavanje ulaznog signala iz prethodnog sloja kako bi nakon propagacije kroz tekući sloj dimenzije signala bile sačuvane, dok se korak odnosi na pomeraj filtera prilikom računanja konvolucije kako bi se dimenzije signala smanjile u tekućem sloju.

Stohastički gradijentni spust i polugradijentne metode

Jedne od najprimenjenijih metoda funkcionalne aproksimacije su metode zasnovane na *stohastičkom gradijentnom spustu* (eng. *stochastic gradient descent*, SGD). Zbog mogućnosti ažuriranja parametara na osnovu pojedinačnih ili manjim podskupovima primera, ove metode su posebno pogodne za metode učenja potkrepljivanjem koje uče u toku epizode.

U metodama gradijentnog spusta, vektor težina predstavlja vektor sa fiksnim brojem realnih komponenti $\mathbf{w} \doteq (w_1, w_2, \ldots, w_d)^\top$, dok je aproksimacija funkcije vrednosti $q_{\mathbf{w}}(s, a)$ diferencijabilna funkcija po $\mathbf{w}$ za svako $s \in \mathcal{S}$ i $a \in \mathcal{A}(s)$. Kako bi se greška aproksimacije minimizovala na opaženim primerima, potrebno je podešavati vektor težina nakon svakog primera za neku malu vrednost u pravcu koji bi najviše doprineo smanjenju greške na tom primeru. Vrednosti parametara $\mathbf{w}$ se ažuriraju u svakom koraku $t = 0, 1, 2, \ldots$ na sledeći način:

$$\begin{aligned}\mathbf{w}_{t+1} &\doteq \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}\Big[q^\pi(S_t, A_t) - q_{\mathbf{w}}(S_t, A_t)\Big]^2 \\ &= \mathbf{w}_t + \alpha\Big[q^\pi(S_t, A_t) - q_{\mathbf{w}}(S_t, A_t)\Big]\nabla_{\mathbf{w}} q_{\mathbf{w}}(S_t, A_t)\end{aligned}$$

gde je α pozitivan parametar koji se zove *korak učenja*. Jedan korak u računanju $\mathbf{w}_t$ je proporcionalan negativnoj vrednosti *gradijenta* $\nabla_{\mathbf{w}}$ kvadrata greške po $\mathbf{w}$ na jednoj instanci koji pokazuje pravac u kom funkcija greške najbrže opada.

Korak učenja može da se izabere tako da odjednom, u jednom stanju, svede grešku na nulu, ali se to ne radi jer su u pitanju metode aproksimacije koje imaju mnogo manje parametara nego što ima stanja pa će zbog toga uklanjanje greške u jednom stanju sigurno povećati grešku u drugom. Stoga je nemoguće eliminisati grešku na svakom stanju. Umesto toga, potrebno je postepeno tražiti konfiguraciju koja će, uzimajući u obzir i učestalost pojavljivanja stanja, napraviti balans koji minimizuje ukupnu grešku. Ako se korak α smanjuje tako da zadovoljava standaradne uslove stohastičke aproksimacije, tada ova metoda stohastičkog gradijentnog spusta konvergira ka lokalnom optimumu sa verovatnoćom 1.

Pošto je ciljna vrednost $q^\pi(S_t, A_t)$ nepoznata, neophodno je koristiti neku njenu ocenu. Tada je konvergencija ka lokalnom optimumu za opadajuće α zaga-

rantovana pod uslovima stohastičke aproksimacije samo ako je posmatrana vrednost *nepristrasna ocena* vrednosti $q^\pi(S_t, A_t)$, odnosno da za neku ocenu U_t važi $\mathbb{E}[U_t|S_t = s, A_t = a] = q^\pi(S_t, A_t)$ za svako t. Uopšteni izraz za ažuriranje težina metodama gradijentnog spusta za aproksimaciju funkcije vrednosti akcije u stanju je sledećeg oblika:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[U_t - q_{\mathbf{w}}(S_t, A_t)\Big]\nabla q_{\mathbf{w}}(S_t, A_t).$$

Ciljna vrednost U_t može biti bilo koja aproksimacija funkcije $q^\pi(S_t, A_t)$. Na primer, izraz za ažuriranje težina metode jednokoračnog Q-učenja je:

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\Big[R_{t+1} + \gamma \max_a q_{\mathbf{w}}(S_{t+1}, a) - q_{\mathbf{w}}(S_t, A_t)\Big]\nabla q_{\mathbf{w}}(S_t, A_t).$$

Međutim, računanje ocene vrednosti stanja i akcija na osnovu ocena vrednosti drugih stanja zavisi od vektora težina $\mathbf{w}_t$, što znači da će ocene koje ove metode računaju ipak biti pristrasne.

Zamenom ciljne vrednosti $q^\pi(S_t, A_t)$ aproksimacijom pomoću ocena koje zavise od $\mathbf{w}_t$ u prethodnom izrazu se dobija ažuriranje parametara koje uzima u obzir zavisnosti od parametara $\mathbf{w}_t$ prilikom predviđanja ali ne i prilikom računanja ocene ciljnih vrednosti. Drugim rečima uzima se u obzir samo deo gradijenta, te se zbog toga ove metode zovu *polugradijentne metode* i ne smatraju se pravim metodama gradijentnog spusta [5]. Iako konvergencija polugradijentnih metoda nije toliko robusna kao što je to u slučaju gradijentnih, pokazalo se da mogu pouzdano da konvergiraju pri linearnoj funkcionalnoj aproksimaciji, a pod određenim uslovima i u slučaju nelinearne funkcionalne aproksimacije.

Za proširivanje polugradijentnih metoda predviđanja na probleme upravljanja sa aproksimacijom, potrebno je još kombinovati metode predviđanja vrednosti akcija sa tehnikama za izbor akcija i ažuriranje politike. U slučaju ne prevelikog diskretnog skupa akcija, za svaku moguću akciju a u stanju S_t se može računati $q_{\mathbf{w}_t}(S_t, a)$ i potom pronaći pohlepna akcija $A_t^* = argmax_a q_{\mathbf{w}_t}(S_t, a)$. Ažuriranje politike se zatim može postići na isti način kao i u slučaju tabelarnih metoda, tako da bude pohlepna u odnosu na novu funkciju $q_{\mathbf{w}}$ za ažurirane parametre $\mathbf{w}$, dok se akcije mogu izabrati u odnosu na neku politiku ponašanja, na primer ε-pohlepnu. Pun naziv ove metode je *epizodično polugradijentno jednokoračno Q-učenje* i predstavlja osnovu za algoritam koji će biti predstavljen u narednom delu.

DQN: TD upravljanje mimo politike sa aproksimacijom

Postoji više problema koji se javljaju kod polugradijentnih metoda učenja mimo politike. Svaki od njih doprinosi nestabilnosti koja u ovim metodama može dovesti i do divergencije. Uzrok većine tih problema je upravo funkcionalna aproksimacija. Naime, jedno od ograničenja je vezano za neefikasnost stohastičkog gradijentnog spusta sa pojedinačnim instancama, dok se ostatak odnosi na nestabilnosti koje se javljaju zbog različitih raspodela ciljne politike i politike ponašanja i korelacija između stanja koja nastaju kao rezultat generalizacije [44].

Jedan od algoritama koji se u praksi pokazao kao otporan na sve ove probleme je *duboka Q mreža* (eng. *deep Q network*, DQN) [33, 34]. DQN predstavlja varijantu polugradijentnog algoritma Q-učenje koja koristi konvolutivnu neuronsku mrežu za aproksimaciju funkcije Q čiji su parametri $\mathbf{w}$. Za računanje funkcije Q se može koristiti bilo koji funkcionalni aproksimator ali će zbog potreba ovog istraživanja, u nastavku ovog rada biti pretpostavljena varijanta sa konvolutivnom mrežom. Ažuriranje parametara $\mathbf{w}$ se vrši u odnosu na funkciju greške L koja u i-toj iteraciji ima sledeći oblik:

$$L_i(\mathbf{w}_i) \doteq \mathbb{E}_{s,a,r,s'}\left[\Big(y_i - Q_{\mathbf{w}_i}(s,a)\Big)^2\right]$$
$$y_i \doteq r + \gamma \max_{a'} Q_{\mathbf{w}'}(s',a')$$

gde $\mathbf{w}'$ predstavlja vektor težina fiksirane *ciljne mreže*. Ključna inovacija ovog algoritma je korišćenje tehnike *zamrzavanja težina* [34]. Ovo podrazumeva povremeno ažuriranje parametara ciljne mreže $Q_{\mathbf{w}'}$ nakon određenog broja iteracija, dok se parametri *mreže predviđanja* ažuriraju gradijentnim spustom. Gradijent pomoću kojeg se ažuriraju parametri mreže predviđanja je sledeći:

$$\nabla_{\mathbf{w}_i} L_i(\mathbf{w}_i) = \mathbb{E}_{s,a,r,s'}\left[\Big(y_i - Q_{\mathbf{w}_i}(s,a)\Big)\nabla_{\mathbf{w}_i} Q_{\mathbf{w}_i}(s,a)\right]$$

Cilj ove tehnike je da smanji korelaciju između ciljnih vrednosti i vrednosti predviđanja tako što će dve mreže računati vrednosti u odnosu na različite vektore težina, čime se znatno smanjuje i nestabilnost algoritma.

Druga ključna komponenta algoritma DQN jeste *reprodukcija iskustva* (eng. *experience replay*) [30, 34]. Tokom učenja agent akumulira skup $\mathcal{D}_t = \{e1, e2, \ldots, e_t\}$ iskustava čiji su elementi $e_t = (s_t, a_t, r_t, s_{t+1})$ trenuci donošenja odluka iz različitih epizoda. Umesto da se mreža obučava samo na osnovu instance iskustva koju agent u tom trenutku doživljava, kao što predlaže uobičajeni pristup TD-učenja,

koriste se slučajni uzorci iskustva iz skupa $\mathcal{D}$ uzorkovani uniformnom raspodelom. Nakon ove izmene funkcija greške ima sledeći oblik:

$$L_i(\mathbf{w}_i) \doteq \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}\left[\left(y_i - Q_{\mathbf{w}_i}(s,a)\right)^2\right]$$

Reprodukcija iskustva smanjuje zahteve algoritma za količinom podataka kroz ponovnu upotrebu instanci iskustva prilikom ažuriranja parametara, i što je još važnije, ovim pristupom se smanjuje i varijansa zbog toga što se uniformnom raspodelom biraju instance koje nisu međusobno korelisane. Dodatno, upotrebom skupova instaci prilikom ažuriranja parametara se povećava efikasnost gradijentnog spusta. Na slici 2.2 je prikazan pseudokod algoritma DQN.

Slika 2.2: Duboka Q mreža

Alocirati memoriju iskustva $\mathcal{D}$ veličine N
Nasumično inicijalizovati parametre w
Inicijalizovati parametre w' na w
for *i=1,M* **do**
  Inicijalizovati početno stanje s_1
  for *t=1,T* **do**
    $a_t = \begin{cases} \text{nasumična akcija} & \text{sa verovatnoćom } \varepsilon \\ \text{argmax}_a Q_w(s_t, a) & \text{inače} \end{cases}$
    Izvršiti a_t i opaziti r_{t+1} i s_{t+1}
    Sačuvati $(s_t, a_t, r_{t+1}, s_{t+1})$ u $\mathcal{D}$
    Nasumično izabrati podskup $\mathcal{D}$' iz $\mathcal{D}$
    for *j=1,*$|\mathcal{D}'|$ **do**
      $y^j = \begin{cases} r^j_{t+1} & s^j_{t+1}\text{je završno} \\ r^j_{t+1} + \max_a Q_{w'}(s^j_{t+1}, a) & \text{inače} \end{cases}$
    end
    Izvršiti po w korak gradijentnog spusta za funkciju:

$$\sum_{j=1}^{|\mathcal{D}'|} (y^j - q_w(s^j_t, a^j_t))^2$$

    Ako je t deljivo sa C postaviti $w' = w$
  end
end

Glava 3

Rešavanje autonomnog kretanja u FPS igrama

Učenje upravljanja agentima na osnovu visokodimenzionih sirovih signala kao što su svetlost i zvuk je jedan od dugovečnih izazova učenja potkrepljivanjem. Za jedno od najranijih istraživanja u oblasti vizuelnog učenja potkrepljivanjem zaslužan je Minoru Asada [3, 2], koji je obučavao robote raznim veštinama igranja fudbala upotrebom kombinacije Q-učenja i jednostavnog kodiranja stanja na osnovu slika iz video kamere. Drugi značajan rad u kome se istražuje obučavanje robota za kretanje kroz 3D prostor uključuje duboko Q-učenje upotrebom potpuno povezane neuronske mreže za obradu slika iz video kamere [15]. Takođe, isprobavani su i duboki autoenkoderi za učenje politika pomoću algoritama učenja potkrepljivanjem na osnovu predefinisanih uzoraka (eng. *batch-mode*) [28], kao i neuroevolutivni algoritmi za vizuelnu verziju problema planinskog automobila (eng. *mountain car*) [10] i kompresovana neuroevolucija sa rekurentnim neuronskim mrežama za vizuelni simulator vožnje automobila [26].

Zbog nepraktičnosti algoritama učenja potkrepljivanjem u stvarnom svetu, javlja se potreba za okruženjima koja omogućavaju jednostavno i fleksibilno testiranje ovih algoritama. Postoje različite platforme koje nude ove mogućnosti. Jedne od njih predstavlju video igre, među kojima je najpoznatija okruženje učenja arkadnih igara (eng. *arcade learning environment*, ALE). Ovo okruženje omogućava primenu vizuelnog učenja potkrepljivanjem na preko 500 igara na platformi Atari 2600. Ranije primene na ovoj platformi su se oslanjale na ručno izrađene karakteristike u kombinaciji sa linearnim reprezentacijama funkcija vrednosti i politika [6], dok su nedavno ti uspesi nadmašeni pomoću različitih pristupa koji uključuju

primenu dubokog Q-učenja [34, 48, 38, 49, 32, 14, 7, 21].

Prethodne metode su uglavnom primenjivane u 2D okruženjima, tako da je za napredak ka primeni u stvarnom svetu potrebno najpre unaprediti ove metode kako bi im se primena mogla lakše preneti na realna 3D okruženja. Jedan način kako bi se ovo postiglo jeste razmatranje igara sa perspektivom iz prvog lica kao što je *pucačka igra iz prvog lica* (eng. *first person shooter*, FPS) u 3D okruženju. Ova igra je mnogo izazovnija od većine Atari igara jer uključuje upotrebu širokog spektra veština, kao što je navigacija kroz mapu, prikupljanje predmeta i prepoznavanje i borba protiv neprijatelja. Štaviše, stanja su delimično opaziva, a agent se kreće kroz 3D okruženje u perspektivi iz prvog lica, što zadatak čini pogodnijim za primene robotike u stvarnom svetu.

Različite FPS igre su već korišćene kao platforme za istraživanje u oblasti veštačke inteligencije. Prvi akademski rad u kom se pominje primena veštačke inteligencije u FPS igrama se odnosi na modeliranje ponašanja igrača u igri Soldier of Fortune 2 pomoću nadgledanog učenja [16]. Zatim, različiti genetski algoritmi su se koristili za podešavanje parametara statičkih ekspertskih sistema zasnovanih na pravilima u igri Counter Strike [9]. Igra Unreal Tournament 2004 se takođe pokazala kao potencijalan poligon za istraživanje u oblasti veštačke inteligencije [12], dok se igra Quake 3 Arena koristila za proučavanje tehnika izbora oružja pomoću nadgledanog učenja [13]. Naprednije implementacije uključuju algoritam učenja potkrepljivanjem RETALIATE za optimizaciju timskih taktika u igri Unreal Tournament [41] i SARSA(λ) u različitim FPS igrama [31, 17, 18]. Nedavno su primenjene tehnike učenja potkrepljivanjem na kontinualne probleme učenja nestatičkog ponašanje tenkova u igri BZFlag [42].

Pored prethodno navedenih FPS igara, korišćene su i druge igre sa perspektivom iz prvog lica za istraživanje u oblasti veštačke inteligencije. Međutim, svi ovi pristupi, izuzev jednog značajnog istraživanja u igri Minecraft [1], su se oslanjali na informacije visokog nivoa kao što je položaj zidova, predmeta i neprijatelja, koje pri normalnim uslovima igranja ljudima nisu dostupne. U nastavku će biti predstavljeno rešavanje jednog aspekta problema upravljanja u FPS igrama. Cilj je napraviti agenta koji je će moći uspešno da nauči da se kreće kroz 3D okruženja dok njegov opstanak zavisi od prepoznavanja, prikupljanja i izbegavanja različitih predmeta i prepreka. Agent bi morao da reši zadatak samo na osnovu slika sa ekrana, nagrade, prethodno izabranih akcija i stanja svog zdravlja, bez ikakvih informacija koje inače nisu dostupne ljudima pri normalnim uslovima igranja.

3.1 Igra Doom i ViZDoom okruženje

Igra Doom je veoma važan korak u razvoju 3D akcionih igara. Smatra se jednom od najvažnijih i najuticajnijih igara u istoriji industrije video igara. Predstavlja igru koja je definisala žanr pucačkih igara iz prvog lica i koja je zauvek promenila tok razvoja video igara. Uvela je mnoge novine kako u načinu igranja igara tako i u razvoju tehnika grafičkog iscrtavanja u realnom vremenu. Najpopularnije FPS igre kao što su Quke 3 Arena, Unreal Tournament i Counuter-Strike, već su korišćene kao poligoni za ispitivanje algoritama veštačke inteligencije. Međutim, mali je broj istraživanja rađeno na igri Doom. Jedino značajno istraživanje uključuje upotrebu algoritma učenja potkrepljivanjem pomoću rekurentne neuronske mreže za igru protiv drugih igrača kojima takođe upravlja računar [27].

Najistaknutija komponenta igre Doom je njen pokretač (eng. *engine*) idTech 1 koji omogućava pseudo 3D grafiku, igru za više igrača preko mreže i najvažnije, podršku za lako kreiranje dodataka i modifikacija pomoću paketa u obliku WAD formata. Jedinstvena komponenta koja ga izdvaja od ostalih okruženja jeste njegov softverski iscrtavač koji nudi mogućnosti koje nisu dostupne u složenijim 3D igrama sa perspektivom iz prvog lica. Omogućava direktan pristup memoriji na GPU koja čuva sadržaj koji se iscrtava na ekranu, bez da se prvo učitava u radnu memoriju. Iako novije verzije ovog pokretača podrazumevaju iscrtavanje u visokim rezolucijama one takođe podržavaju i rezolucije manje od 320x240. Pored toga, moguće je i iscrtavanje van ekrana što omogućava pokretanje igre bez grafičkog okruženja, pogodno za udaljene konekcije preko komandne linije.

idTech 1 je dizajniran da radi na većini platformi. Štaviše, njegov kod je lako razumljiv i slobodan već decenijama što je omogućavalo razvoj zajednice korisnika kroz kreiranje dodatnog sadržaja i modifikacija koja je i do dan danas aktivna. Jedna od mnogih modifikacija igre Doom je ViZDoom okruženje [24]. Zasnovano je na modernoj verziji ZDoom orignalnog pokretača koja je još uvek aktivno održavano.

ViZDoom okruženje omogućava razvoj autonomnih agenata koji igraju Doom na osnovu informacije sa ekrana. Okruženje podrazumeva 3D virtualni svet sa perspektivom iz prvog lica koji je značajno realističniji od okruženja koje nude igre sa platforme Atari 2600 i takođe nudi relativno realističan fizički model. Agent u okruženju ViZDoom mora uspešno da percipira, interpretira i nauči 3D okruženje kako bi donosio strateške odluke koje bi mu pomogle da reši zadatak koji se od

njega traži. Prednosti ovog okruženja kao platforme za istraživanje učenja potkrepljivanjem, je u mogućnostima prilagođavanja potrebama zadatka. Omogućava lako definisanje scenarija, mapa, različitih elemenata okruženja, sporednih likova, nagrada, ciljeva i mogućih akcija koje agent može da preduzme. Omogućava vršenja akcija koje odgovaraju komandama (ili njihovim kombinacijama) sa miša i tastature. Promenljive koje sadrže informaciju o stanju kao što je zdravlje agenta ili količina municije su takođe dostupne.

Karakteristike okruženja ViZDoom

Okruženje ViZDoom pruža funkcije koje se mogu iskoristiti u različitim vrstama eksperimenata veštačke inteligencije. Njegov API je fleksibilan, jednostavan za korišćenje i omogućava rad na više platformi. Veoma je lagan i efikasan, na savremenim računarima može da izvršava do 7000 frejmova u sekundi. Napisan je na jeziku C++ ali omogućava vezivanja za jezike Python, Lua i Java i nudi veliki broj opcija za podešavanje načina upravljanja i iscrtavanja. Glavne karakteristike uključuju različite režime upravljanja, korisnički definisane scenarije, pristup memoriji dubine prostora (eng. *depth buffer*), preskakanje frejmova i iscrtavanje van ekrana čime se eliminiše potreba za korišćenjem grafičkog interfejsa.

ViZDoom podržava četiri načina upravljanja: *sinhroni igrač*, *sinhroni posmatrač*, *asinhroni igrač* i *asinhroni posmatrač*. U asinhronim režimima, brzina iscrtavanja je fiksirana na 35 frejmova u sekundi, pri čemu agent može da propusti neke frejmove ako reaguje presporo. Sa druge strane, ako donese odluku prebrzo, agent je blokiran se sve do završetka iscrtavanja sledećeg frejma. Dakle, za istraživanje učenja potkrepljivanjem korisniji su sinhroni režimi, u kojima pokretač igre čeka donosioca odluke, što omogućava agentu da uči svojim tempom jer nije uslovljen nikakvim vremenskim ograničenjima. Drugo važno svojstvo sinhronih režima je deterministički način rada, što omogućava eksperimentalnu ponovljivost i pomaže u otkrivanju i otklanjanju grešaka. U režimima igrača, agent je taj koji preduzima akcije, dok u režimima posmatrača, čovek upravlja igrom, a agent samo posmatra akcije igrača. ViZDoom takođe podržava i asinhroni režim za više igrača, koji omogućava igru koja uključuje do osam igrača preko mreže kojima mogu da upravljaju ljudi ili računar.

Jedna od najvažnijih karakteristika ovog okruženja je mogućnost pokretanja korisnički definisanih scenarija, što podrazumeva kreiranje odgovarajućih mapa, programiranje mehanika okruženja koje određuju kada i kako se stvari dešavaju,

definisanje uslova završetka igre i nagrade pri ostvarivanju odgovarajućih događaja kao na primer, eliminisanje neprijatelja, dolazak na određeno mesto, povređivanje ili prikupljanje određenih predmeta. Upotreba korisnički definisanih scenarija podiže nivo fleksibilnosti okruženja, pre svega zbog mogućnosti prilagođavanja težine scenarija tako da odgovara mogućnostima ispitanih algoritama učenja. Kreiranje scenarija je moguće zahvaljujući softverskim alatima kreiranih od strane Doom zajednice koji su jednostavni za korišćenje.

ViZDoom obezbeđuje pristup memoriji koja čuva informaciju o dubini prostora, što može značajno da pomogne agentu u razumevanju 3D okruženja. Naime, ova komponenta omogućava ispitivanje da li algoritmi učenja mogu samostalno da saznaju gde se nalaze objekti u okruženju. Pored toga, primena informacije o dubini bi mogla biti i u simulaciji senzora za detekciju udaljenosti u robotici.

Kako bi se olakšalo izvršavanje računski zahtevnih eksperimenata mašinskog učenja, ViZDoom obezbeđuje funkcije kao što su iscrtavanje van ekrana i preskakanje frejmova. Iscrtavanje van ekrana smanjuje ulaganje resursa u prikazivanje igre na ekranu i omogućava pokretanje eksperimenata na sistemima gde nije dostupno grafičko okruženje, dok preskakanje frejmova omogućava da se izostavi iscrtavanje izabranih frejmova što omogućava obučavanje efikasnih agenata koji ne zahtevaju da im se prosledi svaki frejm kako bi donosili ispravne odluke.

3.2 Definisanje okvira problema

Proces igranja se u klasičnim FPS igrama ranih 90-ih odlikuje velikom brzinom kretanja, izazovnim ciljevima i raznovrsnošću mehanika. Na prvi pogled igra Doom deluje veoma jednostavna - igraču koji se kreće kroz 3D okruženje je dostupan određen broj oružja koji mu omogućava da eliminiše neprijatelje na putu do izlaza u sledeći segment (nivo) igre. Međutim, igra Doom je, kao i ostale FPS igre njene generacije, veoma kompleksna. Njen proces igranja uključuje nekoliko nezavisnih mehanika koje se primenjuju u različitim situacijama. Jedna je *istraživanje*, koja podrazumeva veliki broj neagresivnih aktivnosti kao što je navigacija kroz mapu, prikupljanje resursa koji poboljšavaju stanje igrača kao što su oružja, paketi prve pomoći, municija, štit ili pojačanja u obliku čarobnih sfera i ključeva koji omogućavaju pristup različitim delovima mape. Takođe obuhvata prepoznavanje i aktiviranje okidača za otključavanje vrata i pomeranje platformi i zidova, pristupanje skrivenim delovima okruženja i izbegavanje opasnih površina i zamki.

Druga mehanika je *borba protiv neprijatelja* koja podrazumeva procenu opasnosti kroz prepoznavanje broja i tipa neprijatelja, način kretanja koje uključuje izbegavanje projektila i održavanje rastojanja, i ciljanje i pucanje na neprijatelje. Treća mehanika procesa igranja je *upravljanje resursima* koja se odnosi na napredne strategije koje uključuju kombinaciju prethodne dve mehanike. Neuzimanje određenih predmeta i vraćanje unazad kroz nivo kada će izostavljeni predmeti biti potrebniji, taktičko biranje mete na osnovu prioriteta eliminacije neprijatelja, taktičko biranje mete na osnovu dostupne municije, stanja zdravlja i položaja, udaljenosti i vrste neprijatelja i izazivanje međusobnog sukoba među neprijateljima navođenjem neprijateljskih projektila na druge neprijatelje.

Sve prethodno navedene mehanike se mogu koristiti nepromenjene u korisnički definisanim scenarijima. Takođe, mehanike se mogu dodati, oduzeti ili modifikovati u cilju rešavanja drugih ciljeva igre. Dizajniranje nivoa zajedno sa definisanjem mehanika procesa igranja čine kompletan scenario koji igrači rešavaju tokom igranja. Jedan od alata za kreiranje korisnički definisanih scenarija koji se intenzivno koristi među članovima Doom zajednice je *DoomBuilder 2*. Ovaj alat omogućava lako modifikovanje i kreiranje korisnički definisanih mapa, programiranje mehanika igre pomoću ugrađenog Action Code Script jezika i smeštanje svih izmena u jednu *WAD* (eng. *where's all the data*) datoteku. Pored toga podržava i intuitivno grafičko okruženje koje omogućava neometan razvoj i testiranje izmena kroz pokretanje scenarija bez izlaska iz okruženja.

Cilj ovog rada je ispitati koliko su metode učenja potkrepljivanjem koje su već primenjivane na problemima u 2D okruženjima zaista moćne. Zbog toga će biti testirano da li bi slične metode funkcionisale u složenijem scenariju koji zahteva značajnije prostorno rezonovanje. Problem upravljanja u FPS igrama je previše složen kako bi se u potpunosti koristio za testiranje metoda koje su primenjivane na problemima u 2D okruženjima. Pošto cilj nije evaluirati sposobnosti učenja više poslova odjednom već upoređivanti performanse metoda na jednom zadatku, problem upravljanja u FPS igrama biće redukovan na samo jedan njegov potproblem - istraživanje.

U scenariju koji će se koristiti kao poligon učenja, okruženje predstavlja zatvoreni lavirint iz kog je nemoguće izaći i u kom je ceo pod prekriven kiselinom. Na podu su takođe raspoređeni paketi prve pomoći i bočice sa otrovom. Oba tipa predmeta se neprestano pojavljuju na nasumičnim mestima tokom cele epizode pri čemu je učestalost pojavljivanja regulisana tako da ni u jednom trenutku nije

moguće skupiti sve pakete prve pomoći ili prepuniti okruženje predmetima. Agentu je dozvoljeno da se kreće napred i da se okreće levo i desno oko svoje ose. U početnom stanju epizode, agent se stvara na nasumičnom mestu u lavirintu pri čemu ga kiselina polako, ali neprestano, povređuje. Da bi preživeo, agent treba da nauči da se kreće kroz okruženje, da prepozna i da „razume" šta znače prva pomoć i otrov i da upravlja resursima tako što će da skuplja pakete prve pomoći i da izbegava bočice sa otrovom. Cilj igre je da agent što duže preživi u ovakvom okruženju. Motivacija za dostizanje ovog cilja je definisana kroz nagradu od jednog poena koju agent dobija nakon svakog preživelog koraka i kaznu od -100 poena za smrt. Svaka epizoda se završava nakon 2100 koraka (1 minut u realnom vremenu) ili kada agent umre, tako da je 2100 maksimalni mogući rezultat. Ako se agent nalazi u stanju mirovanja, dobija se minimalna nagrada od 284 poena.

Kako je ovim zadatkom zapravo definisan problem učenja potkrepljivanjem, način upravljanja okruženja ViZDoom biće podešen na sinhroni režim. Pritom, biće korišćena rezolucija iscrtavanja širine 320 piksela i visine 240 piksela u RGB formatu boja, dok će neki elementi koji nisu važni za učenje, a mogu čak i da smetaju, biti isključeni kao što je iscrtavanje statusne površine igrača, iscrtavanje oružja koje igrač trenutno koristi, iscrtavanje krstića za ciljanje i različitih detalja kao što su bojenje tekstura i iscrtavanje čestica. Takođe biće isključeno ograničenje na broju prikazanih frejmova u sekundi i onemogućen zvučni bafer, dok će interna promenljiva iz pokretača igre koja svakom stanju pridružuje vrednost trenutnog stanja zdravlja igrača biti učinjena dostupnom.

3.3 Formulacija problema u vidu MDP-a

Kako bi se izabrale odgovarajuće metode za rešavanje problema autonomnog kretanja u 3D okruženju, potrebno je najpre formalizovati problem kao MDP. Pošto se radi o problemu u kom se rešavanje može podeliti na nezavisne podsekvence, prirodno je izabrati model epizodičnog MDP-a. Neki aspekti definisanja modela su određeni karakteristikama pokretača igre, kao što je vidno polje agenta koje u ovom slučaju iznosi 90 stepeni. Takođe, pošto se radi o igri u kojoj igrač upravlja u 3D okruženju sa perspektivom iz prvog lica, MDP koji će biti definisan mora biti delimično opaziv. To znači da agent nema pristup kompletnom stanju okruženja već mu se prosleđuje samo neka ograničena reprezentacija stvarnog stanja.

Kako bi se MDP precizno definisao, potrebno je odrediti sve elemente koji

čine okruženje MDP-a. Funkciju koja opisuje dinamiku okruženja takođe definiše pokretač igre i nije direktno dostupna, dok će funkcija nagrade biti definisana u skladu sa ciljevim scenarija. Naime, pošto je cilj agenta da što duže preživi tako što će da nauči da se kreće i upravlja resursima, dovoljno je definisati nagradu od +1 koja će mu biti prosleđena na svakom koraku i -100 za smrt. Dakle, upravljanje resursima neće biti direktno nagrađivano jer je ovaj scenario dovoljno jednostavan, a nagrađivanje agenta je dovoljno često tako da implementacija neće zahtevati *oblikovanje nagrade* [36]. Skup stanja okruženja predstavlja skup svih parova (slika, niz brojeva) čiji će tačan format biti definisan konkretnim algoritmima koji će biti implementirani. Slika stanja može biti bilo koja validna slika okruženja, dok niz brojeva obuhvata zdravlje agenta i poslednju preduzetu akciju. Početno stanje može biti bilo koja slika okruženja sa potpuno zdravim agentom i praznom akcijom, pri čemu se slika okruženja bira nasumično na početku svake epizode. Sa druge strane, završno stanje može biti bilo koje stanje sa zdravljem agenta koje ima vrednost 0, ili bilo koje stanje nakon 2100 koraka.

Skup akcija je konačan i diskretan i predstavlja skup komandi koje se, pri podrazumevanim uslovima upravljanja, zadaju preko tastature. Skup svih mogućih akcija je definisan minimalnim skupom akcija koji je neophodan za rešavanje scenarija, pri čemu u svakom stanju (osim završnog) agent može da preduzme bilo koju akciju. Skup akcija uključuje sve moguće kombinacije triju osnovnih akcija: kretanje napred, okretanje oko svoje ose ulevo i okretanje oko svoje ose udesno. U ovom slučaju to je ukupno osam mogućih akcija. Međutim, ako se uzme u obzir da neke kombinacije akcija proizvode identično ponašanje, onda se skup mogućih akcija može redukovati na skup jedinstvenih ponašanja. Skup ponašanja sa odgovarajućim kombinacijama osnovnih akcija je prikazan u tabeli 3.1.

Tabela 3.1: Skup mogućih ponašanja i odgovarajuće kombinacije akcija

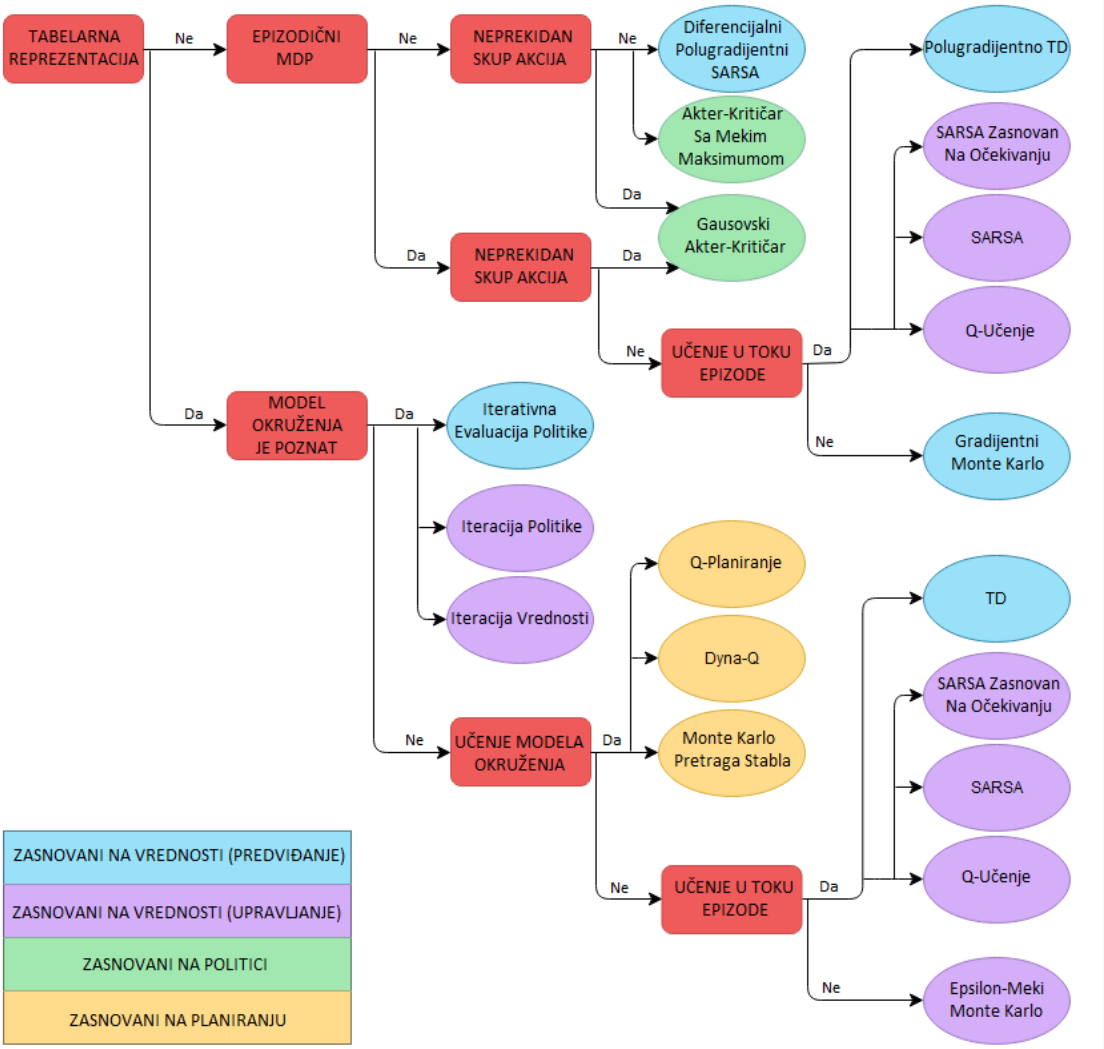
Ponašanje	Kombinacija
Stajanje u mestu	PRAZNA AKCIJA LEVO + DESNO
Kretanje napred	NAPRED LEVO + NAPRED + DESNO
Okretanje oko svoje ose ulevo	LEVO
Okretanje oko svoje ose udesno	DESNO
Kretanje u krug ulevo	LEVO + NAPRED
Kretanje u krug udesno	DESNO + NAPRED

3.4 Konstrukcija osnovnog algoritma

Postoji više ključnih pitanja koja treba postaviti prilikom prepoznavanja odgovarajućeg algoritma za učenje potkrepljivanjem. Prvo se odnosi na veličinu prostora stanja i da li funkcije vrednosti mogu biti predstavljene tabelom. Pošto se radi o skupu stanja koji uključuje slike, nepraktično je konstruisati tabelu koja će čuvati sve moguće vrednosti takvog skupa. Zbog toga je u ovom slučaju najbolje koristiti funkcionalnu aproksimaciju za predstavljanje funkcija vrednosti. Štaviše, algoritam uči nad sirovim signalima iz slika što čini konvolutivne neuronske mreže posebno pogodnim za ovu primenu. Drugo pitanje je vezano za dostupnost funkcije koja opisuje dinamiku okruženja. Kako funkcija prelaska između stanja nije poznata i kako je problem učenja autonomnog kretanja zapravo problem upravljanja, potrebno je da algoritam vrši evaluaciju i računa politiku u odnosu na funkciju vrednosti akcije u stanju. Treća stvar na koju treba obratiti pažnju je tip MDP-a na kom se zasniva posmatrani problem. Ako je u pitanju epizodični MDP, kao što je u slučaju ovog problema, tada je potrebno postaviti pitanje da li je bolje učiti na kraju epizode ili u toku epizode. Postoje različiti potciljevi problema i samim tim i potencijalnih grešaka u predviđanju koje agent može da napravi pre nego što dostigne neko od završnih stanja, pri čemu se agent ne nagrađuje direktno za ostvarivanje posmatranih potciljeva, ali se ipak nagrađuje dosta često tokom epizode. Stoga je bolje omogućiti ažuriranje u toku epizode kako bi agent mogao brže da nauči na „greškama" koje je napravio. Četvrto pitanje se odnosi na tip vrednosti akcija. U slučaju neprekidnog skupa vrednosti akcija, bolje je primeniti metode zasnovane na politici. Međutim, kako je u scenariju koji agent rešava upravljanje ograničeno ulazom koji odgovara samo komandama sa tastature, rezultujući skup akcija je dovoljno mali da bi se mogle računati vrednosti za svaku moguću akciju. Poslednje pitanje predstavlja nagodbu između stabilne konvergencije i učenja deterministčke politike. U kontekstu problema učenja autonomnog kretanja još uvek ne postoji jasan pobednik, pa je izbor između ova dva pristupa proizvoljan. Na slici 3.1 je prikazano stablo procesa identifikacije odgovarajućeg algoritma za dati problem učenja potkrepljivanjem [44].

Odgovaranjem na prethodna pitanja određuju se generalne karakteristike algoritma koji će se koristiti za rešavanje problema učenja potkrepljivanjem. Izborom osnovnih karakteristika algoritma na osnovu specifikacije problema koji se u ovom radu istražuje dolazi se do polugradijentog jednokoračnog epizodičnog Q-učenja

Slika 3.1: Dijagram izbora algoritama učenja potkrepljivanjem



sa dubokom neuronskom mrežom, odnosno do osnovnog oblika algoritma DQN koji je predstavljen u glavi 2. Međutim, postoji veći broj specifičnosti algoritma koje je potrebno definisati kako bi se došlo do potpunog dizajna koji će biti implementiran. Potrebno je definisati komponentu preprocesiranja, različite detalje arhitekture mreže, način inicijalizacije težina mreže, funkciju greške, algoritam optimizacije i eksploraciju.

Najmanja podržana rezolucija u okruženju ViZDoom je 160 piksela širine sa 120 piksela visine dok je podrazumevani format boja RGB. Direktan rad sa ovakvom veličinom frejmova može biti računski zahtevan, naročito ako se radi sa

složenim arhitekturama mreže. Stoga je poželjno primeniti korak preprocesiranja u cilju smanjenja dimenzionalnosti ulaza. Korak preprocesiranja najpre podrazumeva transformaciju RGB reprezentacije u reprezentaciju sa paletom sivih boja pomoću ugrađene metode okruženja ViZDoom. Zatim se veličina ulaza smanjuje na veličinu od 112 piksela širine sa 84 piksela visine. Konačna reprezentacija ulaza se dobija izrezivanjem kvadratnog regiona širine 84 piksela čiji se centar poklapa sa centrom originalnog frejma i koji grubo obuhvata površinu originalnog ulaza. Korak preprocesiranja biće označen sa ϕ i primenjuje se nad svakim novim stanjem pre nego što se unosi u memoriju iskustva, tako da se slike, kada se dohvataju iz memorije, ne moraju svaki put transformisati pre nego što se prosleđuju mreži.

Ključna komponenta algoritma Q-učenje koju je potrebno definisati je njen funkcionalni aproksimator koji aproksimira funkciju Q. Kao što je već napomenuto za aproksimaciju ove funkcije koristi se duboka neuronska mreža čime se dobija osnovni oblik algoritma DQN. Postoji nekoliko načina parametrizovanja funkcije Q pomoću neuronske mreže. Raniji pristupi su koristili istoriju stanja i akciju kao ulazne podatke za mrežu na osnovu kojih se računa vrednost akcije u stanju na ulazu [37]. Glavni nedostatak ovih pristupa je u tome što je potrebna posebna propagacija unapred za izračunavanje vrednosti svake akcije. Naprednije arhitekture uključuju po jednu izlaznu jedinicu za svaku moguću akciju, a ulaz u mrežu predstavlja samo reprezentacija stanja. Na ovaj način se za dato stanje na ulazu, računaju vrednosti svake akcije u samo jednom prolazu. Osnovna arhitektura konvolutivne neuronske mreže se sastoji od tri konvolutivna i dva potpuno povezana sloja. Prvi konvolutivni sloj sadrži 32 filtera širine 8 piksela za 2D konvoluciju sa korakom od 4 piksela. Drugi konvolutivni sloj sadrži 64 filtera širine 4 piksela za 2D konvoluciju sa korakom od 2 piksela. Treći konvolutivni sloj sadrži 64 filtera širine 3 piksela za 2D konvoluciju sa korakom od jednog piksela. Prvi potpuno povezani sloj sadrži 1024 jedinica, dok je izlazni sloj veličine skupa svih mogućih kombinacija akcija, odnosno 8 jedinica. Svi slojevi imaju linearnu ispravljačku jedinicu (eng. *rectified linear unit*, ReLU) kao aktivacionu funkciju, tako da je za inicijalizaciju težina mreže korišćena Kaiming He tehnika koja značajno povećava performanse kod dubokih arhitektura [20].

Ulazne podatke za mrežu predstavlja stanje okruženja koje čini kratkoročna istorija od četiri uzastopne slike okruženja obrađene korakom preprocesiranja ϕ i vektor sa dodatnom informacijom vezanom za odgovarajuće trenutke igre. Dodatna informacija sadrži četiri uzastopne akcije preduzete pri datim slikama okruženja

i vrednosti zdravlja agenta u odgovarajućim trenucima, pri čemu su akcije predstavljene reprezentacijom baznih vektora (eng. *one hot*) sa jedinicama na indeksima koji odgovaraju identifikatorima akcija. Ulazni podaci se prosleđuju mreži u dva zasebna dela. Prvi deo čine slike okruženja koje se prosleđuju prvom konvolutivnom sloju, dok se drugi deo koji čine akcije i vrednosti zdravlja agenta nadovezuje na prvi potpuno povezani sloj mreže koji se dobija tako što se izlaz iz poslednjeg konvolutivnog sloja transformiše u jednodimenzioni sloj. Vektor dimenzije ulaza je definisan veličinom uzorka iskustva, veličinom kratkoročne istorije, dimenzijama slika i veličinom reprezentacije akcija i vrednosti zdravlja agenta. Na prvom konvolutivnom sloju dimenzija ulaza je (64, 4, 84, 84), dok je ulaza na prvom potpuno povezanom sloju veličine (64, 4, 36).

Ažuriranje parametara mreže se vrši pomoću RMSProp optimizacije u odnosu na funkciju greške koja je implementirana pomoću korisnički definisane funkcije na osnovu kvadrata TD greške δ. TD greška predstavlja razliku između ciljnih vrednosti koje se računaju pomoću ciljne mreže sa zamrznutim težinama i vrednosti predviđanja koje se računaju pomoću mreže sa aktuelnim težinama. Optimizacija korisiti uzorke iskustva veličine 64, dok korak optimizacije i parametar interpolacije imaju vrednosti redom $\alpha = 0.00025$ i $\rho = 0.95$. Na slici 3.2 je prikazan pseudokod algoritma optimizacije RMSProp.

Slika 3.2: Algoritam optimizacije RMSProp

Data: Parametri w_t, TD greška δ_t
Result: Ažurirani parametri w_{t+1}
Inicijalizovati vektore u_0 i v_0 nulama
for *t=1,T* **do**
  $g_t = \nabla_w \delta_t^2$
  $u_t = \rho u_{t-1} + (1-\rho) g_t$
  $v_t = \rho v_{t-1} + (1-\rho) g_t^2$
  $w_{t+1} = w_t - \frac{\alpha g_t}{\sqrt{v_t - u_t^2 + \epsilon}}$
end

Koordinate vektora u i v u algoritmu RMSProp predstavljaju srednje vrednosti i kvadrate srednjih vrednosti koordinata gradijenta, dok je deljenje u poslednjem koraku petlje pokoordinatno deljenje, a ϵ mala vrednost koja sprečava deljenje nulom.

Uzorkovanje iskustva osnovnog algoritma se vrši uniformnom raspodelom, dok je eksploracija definisana ε-pohlepnom politikom.

3.5 Napredne tehnike i poboljšanja

Konvergencija polugradijentnih metoda učenja mimo politike predstavlja oblast najsavrmenijeg istraživanja. Puno truda se ulaže u unapređivanje postojećih metoda i učvršćivanje teorije na kojoj se zasnivaju. Postoji veliki broj kontraprimera koji pokazuju nestabilnost i divergenciju polugradijentnih metoda kao što je Q-učenje sa funkcionalnom aproksimacijom. To što se iz tih primera može zaključiti je da opasnost od nestabilnosti i divergencije nastaje kad god algoritmi uključuju: *samoaproksimaciju*, *učenje mimo politike* i *funkcionalnu aproksimaciju*. Rezultati ovih istraživanja su upečatljivi jer su korišćene najjednostavnije i najbolje shvaćene metode funkcionalne aproksimacije sa metodama učenja mimo politike zasnovanim na samoaproksimaciji kao što je Q-učenje, koja pritom ima i najbolje garancije konvergencije [44]. Prema tome, treba imati na umu da opasnost ne proizilazi iz GPI procedure ili učenja u nepoznatom okruženju već je to isključivo posledica kombinacije prethodno navedenih tehnika. Ako su prisutne bilo koje dve, ali ne i sve tri, onda se nestabilnost može izbeći. Međutim sva tri elementa su od suštinskog značaja za primenu algoritama u praksi. Bilo da je to skalabilnost, mogućnost učenja u toku epizode ili učenja više politika odjednom, izostavljanje bilo kojeg od ovih elemenata je nemoguće ako je krajnji cilj napraviti moćne agente čije bi sposobnosti bile na nivou opšte inteligencije.

Izazov polugradijentnih metoda učenja mimo politike se može podeliti na dva dela, jedan koji se javlja u tabelarnom slučaju i drugi koji nastaje samo sa funkcionalnom aproksimacijom. Prvi deo izazova je vezan za ciljne vrednosti TD-popravke i zahteva samo proširenje već postojećih rešenja tabelarnih metoda na metode sa funkcionalnom aproksimacijom, dok je drugi nešto složeniji i odnosi se na neslaganje raspodela TD-popravke i politike ponašanja. Istražena su dva pristupa rešavanju drugog problema. Prvi je da se koriste tehnike za prepravljanje raspodele TD-popravke, a drugi je razvoj pravih gradijentnih metoda kod kojih stabilnost ne zavisi od posebnih raspodela. Ovo je najsavremenija oblast istraživanja i nije jasno koji od ovih pristupa je najefikasniji u praksi.

Pored navedenih problema polugradijentnih metoda koji doprinose nestabilnosti, postoji i određen broj nedostataka osnovnih oblika ovih metoda koji vode do znatno sporije konvergencije. U nastavku biće predstavljene rezličite tehnike koje smanjuju nestabilnost i ubrzavaju konvergenciju i koje ujedno predstavljaju komponente najsavremenije varijante algoritma DQN koja se zove *Rainbow*.

Pristrasnost maksimizacije i dvostruko učenje

Poznato je da metode kao što je Q-učenje često uče nerealno visoke vrednosti akcija jer uključuju korak maksimizacije u odnosu na ocene vrednosti akcija, što teži da preferira precenjene vrednosti pre nego potcenjene. U ovim algoritmima, maksimum ocena vrednosti se koristi implicitno kao ocena maksimalne vrednosti, što može dovesti do značajne pozitivne pristrasnosti. Na primer, neka sve vrednosi $q(s, a)$ akcija a u stanju s imaju vrednost nula i neka su ocene vrednosti $Q(s, a)$ netačne i raspoređene iznad i ispod nule. Tada je pravi maksimum nula, dok je maksimum ocena veći od nule. Ovo precenjivanje pravog maksimuma se zove *pristrasnost maksimizacije*.

Precenjivanje se javlja kada su vrednosti akcije netačne, bez obzira koji je izvor greške aproksimacije [48]. Naravno, neprecizne ocene vrednosti su uobičajene tokom učenja, što ukazuje na to da se precenjivanje može desiti dosta često. Optimistične ocene vrednosti nisu nužno problem same po sebi. Kada bi sve vrednosti bile uniformno veće, relativne preferencije akcija bi bile očuvane i ne bi se očekivalo da će rezultujuća politika biti gora. Štaviše, poznato je da je optimizam u stanju neizvesnosti dobro poznata tehnika istraživanja [23]. Međutim, ako precenjivanja nisu uniformna i nisu koncentrisana na stanjima o kojima se želi više saznati, onda bi one mogle negativno uticati na kvalitet rezultujuće politike. Trun i Švarc [46] daju konkretne primere u kojima to vodi ka neoptimalnim politikama, čak i asimptotski.

Dvostruka DQN

Jedan od načina da se posmatra problem pristrasnosti maksimizacije je taj da se on javlja usled korišćenja istih uzoraka i za određivanje maksimalne akcije i za procenu njene vrednosti. Da bi se ovo sprečilo potrebno je najpre odvojiti izbor akcija od njihove evaluacije. Takođe je potrebno podeliti uzorak iskustva na dva podskupa pomoću kojih se uče dve nezavisne ocene vrednosti akcija, tako da postojaće i dva nezavisna skupa težina $\mathbf{w}$ i $\mathbf{w}'$. Zatim bi se jedna ocena, recimo $Q_{\mathbf{w}}$, koristila za određivanje pohlepne akcije $A^* = \text{argmax}_a Q_{\mathbf{w}}(a)$, a druga, $Q_{\mathbf{w}'}$ za računanje ocene njene vrednosti $Q_{\mathbf{w}'}(A^*) = Q_{\mathbf{w}'}(\text{argmax}_a Q_{\mathbf{w}}(a)))$. Ova ocena će tada biti nepristrasna u smislu da je $\mathbb{E}[Q_{\mathbf{w}'}(A^*)] = q(A^*)$, dok će tada pravilo ažuriranja biti sledećeg oblika:

$$Q_{\mathbf{w}}(S_t, A_t) \leftarrow Q_{\mathbf{w}}(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_{\mathbf{w}'}(S_{t+1}, \text{argmax}_a Q_{\mathbf{w}}(S_{t+1}, a)) - Q_{\mathbf{w}}(S_t, A_t) \Big].$$

Ovaj proces se može ponoviti sa zamenjenim ulogoma dve ocene nakon čega bi se dobila druga nepristrasna ocena $Q_{\mathbf{w}}(\text{argmax}_a Q_{\mathbf{w}'}(a)))$. Uloga ocena bi se smenjivala nasumično sa verovatnoćom 0.5, dok bi se ε-pohlepna politika računala u odnosu na zbir ili srednju vrednost ocena $Q_{\mathbf{w}}$ i $Q_{\mathbf{w}'}$.

Prethodno rešenje predstavlja originalni algoritam *dvostruko Q-učenje* (eng. *double Q-learning*)[47]. Ideje ovog algoritma se mogu donekle proširiti na algoritam DQN bez uvođenja dodatnih mreža. Ciljna mreža u DQN arhitekturi je prirodni kandidat za računanje ocena vrednosti akcija, dok se mreža za predviđanja koristi za ocenu pohlepne politike. Ciljna vrednost u ovom slučaju je sledeća:

$$Y_t^{DDQN} \doteq R_{t+1} + \gamma Q_{\mathbf{w}',t}(S_{t+1}, \text{argmax}_a Q_{\mathbf{w}_t}(S_{t+1}, a))$$

pri čemu se težine ciljne mreže ažuriraju na isti način kao i u slučaju standardne DQN, tako što periodično dobijaju vrednosti težina mreže za ocenu aktuelne politike. Ova verzija algoritma DQN se zove *dvostruka DQN* (eng. *double DQN*, DDQN) i predstavlja minimalnu moguću promena DQN u pravcu dvostrukog Q-učenja [48]. Cilj je dobiti većinu prednosti dvostrukog Q-učenja sa minimalnim računskim troškovima, a pritom zadržati ostatak DQN algoritma netaknutim radi poštenog poređenja.

Funkcija prednosti i dueling arhitektura

Za mnoga stanja nije neophodno proceniti vrednost svakog izbora akcije. U nekim stanjima od najveće je važnosti znati koju akciju preduzeti, ali u mnogim drugim stanjima izbor akcije nema značajne posledice na ono što se dešava u okruženju. Zbog toga je potrebno nekako razdvojiti funkciju vrednosti od važnosti akcije u posmatranom stanju koja se zove *prednost* (eng. *advantage*). Pojam razdvajanja funkcija vrednosti i prednosti prvi put pominje Berd [4]. U njegovom originalnom algoritmu koji uključuje ažuriranje pomoću prednosti, Bellmanova jednačina za ažuriranje je razložena na dva dela. Jedan deo odgovara funkciji vrednosti stanja a drugi funkciji prednosti. Koristeći ovaj pristup se pokazalo da je konvergencija znatno brža nego što je u slučaju standardnog Q-učenja [19].

Definisanjem funkcije prednosti se može predstaviti veza između funkcija vrednosti stanja i akcije u stanju:

$$A^{\pi}(s, a) \doteq Q^{\pi}(s, a) - V^{\pi}(s)$$

pri čemu je $\mathbb{E}_{a\sim\pi(s)}[A^{\pi}(s,a)] = 0$. Intuitivno, funkcija V i Q mere koliko je dobro biti u određenom stanju s i koliko je dobro izabrati određene akcije u ovom stanju, dok funkcija A pokazuje relativnu meru važnosti svake akcije u stanju s.

Koristeći definiciju prednosti, može se konstruisati varijanta arhitekture DQN koja računa vrednosti akcija na sledeći način:

$$Q_{\mathbf{w},\eta,\psi}(s,a) = V_{\mathbf{w},\eta}(s) + A_{\mathbf{w},\psi}(s,a).$$

gde su $\mathbf{w}$ parametri konvolutivnog dela mreže, a η i ψ parametri tokova vrednosti stanja i prednosti akcija. Početni slojevi mreže su konvolutivni kao u originalnoj DQN. Međutim, umesto da nakon toga sledi jedan niz potpuno povezanih slojeva, koriste se dva paralelna toka potpuno povezanih slojeva koji su konstruisani tako da omoguće odvojeno računanje aproksimacija funkcija vrednosti i prednosti. Na izlazu mreže, dva toka se kombinuju kako bi se dobila jedna izlazna funkcija Q.

Ograničenje prethodnog pristupa je u tome što funkcija Q definisana na ovaj način uvodi značajnu nestabilnost. Ovaj nedostatak rezultuje lošim performansama kada se funkcija prednosti koristi za evaluaciju akcija prilkom obučavanja. Problem nestabilnosti se može rešiti malom izmenom prethodne jednačine za računanje funkcije Q tako što se od prednosti akcija oduzme srednja vrednost prednosti. Nakon ove izmene funkcija Q je sledećeg oblika:

$$Q_{\mathbf{w},\eta,\psi}(s,a) = V_{\mathbf{w},\eta}(s) + (A_{\mathbf{w},\psi}(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'\in\mathcal{A}} A_{\mathbf{w},\psi}(s,a'))$$

Tada je $a^* = \text{argmax}_{a'\in\mathcal{A}} Q_{\mathbf{w},\eta,\psi}(s,a') = \text{argmax}_{a'\in\mathcal{A}} A_{\mathbf{w},\psi}(s,a')$. Treba imati na umu da iako oduzimanje srednje vrednosti u prethodnoj jednačini pomaže u smanjenju nestabilnosti, ono ne menja poredak vrednosti A, a samim tim i poredak vrednosti Q, čuvajući bilo kakvu ε-pohlepnu politiku zasnovanu na Q vrednostima. Sa jedne strane, ovo gubi originalnu semantiku V i A jer one sada aproksimiraju vrednosti koje odstupaju od pravih vrednosti stanja i prednosti akcija za konstantu, ali sa druge strane povećava stabilnost optimizacije.

Važno je napomenuti da se prethodna jednačina posmatra i implementira kao deo mreže, a ne kao poseban algoritamski korak. Ova arhitektura se zove *dueling DQN* (eng. *dueling DQN*) i odgovorna je za veliki napredak u primeni algoritama u praksi [49].

Prioritizovana reprodukcija iskustva

Osnovna implementacija DQN upotrebljava memoriju iskustva sa kliznim prozorom koja čuva veliki broj poslednjih prelaza između stanja, nad kojom se vrši reprodukcija iskustva uzorkovanjem nasumičnog podskupa uniformnom raspodelom. U opštem slučaju, reprodukcija iskustva može da smanji količinu interakcije sa okruženjem povećavanjem količine računskih resursa potrebnih za skladištenje i uzorkovanje iskustva koji su u većini slučajeva jeftiniji resurs od interakcije agenta sa okruženjem. Problem sa osnovnom implementacijom je što uzorkovanje uniformnom raspodelom pridaje jednak značaj svim prelazima, bez obzira koliko su oni važni za učenje. Idealno bi bilo davati prednost prelazima iz kojih se može najviše naučiti. Jedna implementacija reprodukcije iskustva koja ovo omogućava je *prioritizovana reprodukcija iskustva* [38].

Kriterijum na osnovu kog se određuje važnost nekog prelaza predstavlja najvažniju komponentu prioritizovane reprodukcije iskustva. Ovaj kriterijum se najčešće definiše pomoću vrednosti TD greške prelaza, koja govori koliko je posmatrani prelaz neočekivan, odnosno koliko se vrednost nekog stanja ili para stanja i akcije razlikuje od njene ocene u sledećem koraku. Upotreba prioritizovane reprodukcije iskustva podrazumeva skladištenje poslednje nastale TD greške zajedno sa odgovarajućim prelazom u memoriju iskustva. Postoji više vrsta prioritizovanog uzorkovanja. Najjednostavniji oblik predstavlja pohlepno uzorkovanje koje podrazumeva izbor prelaza sa najvećom apsolutnom TD greškom. Pošto je TD greška dostupna tek nakon jednog koraka, novi prelazi stizaće bez odgovarajuće TD greške. Zbog toga će oni biti smešteni u memoriju sa maksimalnim prioritetom, kako bi im posećenost bila zagarantovana.

Pohlepno prioritizovano uzorkovanje ima nekoliko nedostataka. Prvo, može se desiti da prelazi sa malom TD greškom neće dugo biti ažurirani, jer se zbog skupih obilazaka cele memorije, prelazi ažuriraju samo prilikom uzorkovanja. Ako se uzme u obzir da je memorija iskustva sa kliznim prozorom, može se desiti da se neki prelazi neće nikad ažurirati. Drugo, pohlepno prioritizovano uzorkovanje može biti osetljivo na šum kada su nagrade stohastičke, kao što je to u slučaju samoaproksimacije. Treće je nedostatak raznovrsnosti u podacima koji se javlja usled fokusiranja na malom podskupu iskustva, što može da vodi preprilagođavanju modela. Posebno kada se koristi funkcionalna aproksimacija, TD greške prelaza se sporo smanjuju, što znači da će pohlepno prioritizovano uzorkovanje u početku da preferira samo prelaze sa visokom TD gerškom.

Svi ovi nedostaci se mogu prevazići uvođenjem stohastičkog prioritizovanog uzorkovanja koje vrši interpolaciju između pohlepnog prioritizovanog uzorkovanja i uniformnog nasumičnog uzorkovanja. Ovaj pristup obezbeđuje da je verovatnoća uzorkovanja monotona u odnosu na prioritet prelaza, dok garantuje verovatnoću različitu od nule čak i za prelaze sa najnižim prioritetima. Verovatnoća uzorkovanja prelaza se može definisati na osnovu prioriteta na sledeći način:

$$P(i) \doteq \frac{p_i^\omega}{\sum_k p_k^\omega}$$

gde je $p_i > 0$ prioritet prelaza i, dok parametar ω određuje koliki značaj se pridaje prioritetu, pri čemu $\omega = 0$ odgovara uzorkovanju uniformnom raspodelom.

U zavisnosti od načina računanja prioriteta, razlikujemo više vrsta stohastičke prioritizovane reprodukcije iskustva. Jedna varijanta je proporcionalno prioritizovano uzorkovanje gde je $p_i = |\delta_i| + \epsilon$, gde je ϵ mala pozitivna konstanta koja omogućava da prelazi budu izabrani čak i ako je njihova greška nula. Raspodela ovakvog skupa vrednosti je monotona u odnosu na $|\delta|$. Ova varijanta stohastičkog prioritizovanog uzorkovanja dovodi do velikih ubrzanja u odnosu na uniformnu varijantu.

Pošto memorija iskustva može biti ogromna, potrebno je implementirati je kao strukturu koja omogućava efikasno skladištenje, uzorkovanje i ažuriranje. To se može postići strukturom koja se zove *stablo sume*, koja predstavlja binarno stablo čiji su listovi prioriteti elemenata iskustva, a unutrašnji čvorovi predstavljaju zbir prioriteta potomaka. Na ovaj način se obezbeđuje efikasno računanje sume svih prioriteta podstabla, što omogućava stratifikovano uzorkovanje jednostavnom podelom memorije iskustva na k podintervala iz kojih će se uniformnom raspodelom izabrati po jedan element, gde je k veličina uzorka. Tada će vremenska složenost ažuriranja, skladištenja i uzorkovanja biti reda $O(\log N)$, gde je N veličina memorije iskustva.

Uklanjanje pristrasnosti

Prioritizovana reprodukcija iskustva bira prelaze iz iskustva na osnovu raspodele koja je različita od raspodele politike, što za posledicu ima uvođenje pristrasnosti prilikom uzorkovanja, zbog čega se menjaju očekivane vrednosti kojima će ocene konvergirati, čak i ako je raspodela politike fiksirana. Ova pristrasnost se može popraviti tehnikom koja se zove *uzorkovanje prema važnosti*. Potrebno je

definisati težine uzorkovanja prema važnosti na sledeći način:

$$w_i \doteq \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}$$

gde je β parametar koji određuje stepen uticaja težina w_i na elemente uzorka. Čak i ako pristrasnost ima značajan uticaj na učenje, njen efekat u ranim fazama obučavanja nije toliko važan zbog nestacionarnosti problema učenja potkrepljivanjem, kada su značajnije promene u politici, raspodeli stanja i samoaproksimaciji ciljnih vrednosti. Zbog toga je korisno inkrementalno prepravljati parametar β od neke inicijalne male vrednosti β_0 na početku obučavanja do $\beta = 1$ pri konvergenciji. Efekat uzorkovanja zavisi u velikoj meri od prave kombinacije vrednosti parametara ω i β, gde prvi određuje uticaj prioriteta na raspodelu a drugi koliko će jako ona biti popravljena.

Uzorkoavanje prema važnosti ima još jednu prednost kada se kombinuje sa prioritizovanim uzorkovanjem u kontekstu nelinearne funkcionalne aproksimacije. Naime, pošto gradijent aproksimira pravac najbržeg rasta samo lokalno, veliki koraci mogu da smetaju pri optimizaciji nelinearnih funkcija. Stoga, ovaj pristup kombinacije uzorkovanja iskustva i prepravljanje raspodele omogućava da se prelazi sa velikom greškom vide mnogo puta, a da se pritom norma gradijenta i veličina koraka optimizacije smanjuju kako bi algoritam mogao da prati krivinu nelinerane površi, jer se time Tejlorov razvoj iz početka aproksimira nakon manjih koraka.

Višekoračno učenje

Kod jednokoračnih TD metoda, isti vremenski korak određuje vremenski interval u kom dobijena nagrada utiče na vrednosti stanja i akcija i nakon kog će se vršiti samoaproksimacija. U većini slučajeva poželjno je imati mogućnost brzog ažuriranja promena nastalih nakon svake preduzete akcije, međutim, upotreba samoaproksimacije daje najbolje rezultate ako je interval između uzastopnih ažuriranja duži, kada je došlo do značajnijih promena stanja.

Jedan nedostatak korišćenja jednokoračnih metoda je taj što dobijanje nagrade r utiče samo na vrednosti neposrednog para akcija stanja (s, a) koje su dovele do nagrade. Na vrednosti drugih parova akcija i stanja utiče samo indirektno preko ažurirane vrednosti $Q(s, a)$. Ovo može usporiti proces učenja jer su potrebna mnoga ažuriranja da bi se nagrada razdelila na prethodna stanja i akcije koje su takođe doprinele dobijanju posmatrane nagrade. Višekoračne (eng. *n-step*)

metode pravazilaze ova ograničenja i omogućavaju računanje dobitka korišćenjem samoaproksimacije nakon proizvoljnog broja koraka. TD metode u kojima se vremenska razlika proteže na n koraka nazivaju se *n-koračne TD metode* (eng. *n-step TD methods*).

Formalnije, za definisanje n-koračnih TD metoda, potrebno je najpre definisati dobitak koji uključuje samoaproksimaciju. Za metode koje se zasnivaju na samoaproksimaciji vrednosti, jednokoračni dobitak se može definisati na sledeći način:

$$G_{t:t+1} \doteq R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1})$$

gde je Q_t ocena funkcije q_π u trenutku t. Oznaka $G_{t:t+1}$ označava da je dobitak skraćen i da uzima u obzira nagrade od trenutka t do $t+1$, dok $\gamma Q_t(S_{t+1}, A_{t+1})$ predstavlja popravku koja zamenjuje odsečeni deo dobitka $\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$. U opštem slučaju, izraz kojim se definiše dobitak koji uzima u obzir n narednih koraka je sledećeg oblika:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

za svako n i t takve da je $n \geq 1$ i $0 \leq t < T - n$, pri čemu ako je $t + n \geq T$, tada je $G_{t:t+n} \doteq G_t$. Odnosno ako veličina n-koračnog dobitka prekorači broj koraka epizode, tada se vrednosti van granice računaju kao nule, a vrednost n-koračnog dobitka jednaka je uobičajenom kompletnom dobitku.

Sada se uz malu izmenu izraza n-koračnog dobitka može definisati pravilo ažuriranja algoritma *n-koračno Q-učenje* na sledeći način:

$$Q_{t+n}(S_t, A_t) \doteq Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)], 0 \leq t < T,$$

gde je $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{t+n-1} R_{t+n} + \gamma^n max_a Q_{t+n-1}(S_{t+n}, a)$. Treba imati na umu da n-koračni dobici, za $n > 1$, uključuju buduća stanja, akcije i nagrade koje nisu dostupne pre trenutka $t + n$. Ne postoji algoritam koji može da koristi n-koračni dobitak $G_{t:t+n}$ pre nego što R_{t+n} postane dostupna i Q_{t+n-1} može biti izračunata. Takođe, kako bi se nadoknadio broj propuštenih ažuriranja u prvih $n-1$ koraka epizode, toliki broj vrednosti će biti ažurirano nakon završnog stanja, pre početka sledeće epizode.

Važno svojstvo n-koračnog dobitka je što je njegovo očekivanje bolja ocena funkcije q^π nego što je to Q_{t+n-1}, u odnosu na najgori mogući par stanja i akcije. Odnosno, najveća greška n-koračnog dobitka je sigurno manja ili jednaka najvećoj

grešci ocene Q_{t+n-1} pomnoženu sa γ^n:

$$\max_{(s,a)} |\mathbb{E}_\pi[G_{t:t+n}|S_t = s] - q^\pi(s,a)| \leq \gamma^n \max_{(s,a)} |Q_{t+n-1}(s,a) - q^\pi(s,a)|,$$

za svako $n \geq 1$. Ovo svojstvo se zove *svojstvo smanjivanja greške* i na osnovu njega se može pokazati da pod odgovarajućim uslovima n-koračne TD metode konvergiraju ka tačnim predviđanjima.

Konstrukcija napredne implementacije algoritma DQN

Ideje predstavljene u prethodnim poglavljima predstavljaju različita proširenja osnovne varijante algoritma DQN koja za cilj imaju smanjenje nestabilnosti i poboljšanje kvaliteta učenja. Ovaj skup tehnika je komplementaran, odnosno svaka tehnika se odnosi na rešavanje po jednog od različitih nedostatka osnovne implementacije. U nastavku biće predstavljena varijanta algoritma DQN koja implementira sve prethodne tehnike koje su ujedno i komponente poznatog algoritma Rainbow.

Ključnu stvar koju treba izmeniti u osnovnoj implementaciji DQN je funkcija greške, odnosno izraz kojim se računa TD greška. Naime, potrebno je kombinovati višekoračno i dvostruko učenje u izrazu za računanje ciljnih vrednosti, tako što se u n-koračnom dobitku, pri samoaproksimaciji, pomoću mreže za predviđanja bira pohlepna akcija u stanju s_{t+1}, koja će biti evaluirana pomoću mreže za računanje ciljnih vrednosti. Zamenom $r_{t+1}+\gamma \max_a Q_{\mathbf{w}'}(s_{t+1}, a)$ sa $G_{t:t+n}$ u y_i dobija se sledeći izraz koji opisuje funkciju greške:

$$L_i(\mathbf{w}_i) \doteq \mathbb{E}_{(s,a,r,s')\sim\mathcal{P}(\mathcal{D})}\left[\left(y_i - Q_{\mathbf{w}_i}(s,a)\right)^2\right]$$

gde je $y_i = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q_{\mathbf{w}'}(s_{t+1}, \text{argmax}_a Q_{\mathbf{w}}(s_{t+1}, a))$, a $\mathcal{P}$ predstavlja raspodelu definisanu stohastičkim prioritizovanim uzorkovanjem. Pošto aloritam ne uči na osnovu trenutnog prelaska između stanja već na osnovu podataka iz memorije iskustva, prelascima će najpre biti pridružena odgovarajuća višekoračna nagrada i maksimalan prioritet pa će onda biti smešteni u memoriju iskustva iz koje će kasnije biti izabrani uzorkovanjem.

Arhitektura mreže je dueling arhitektura, kod koje je računanje vrednosti stanja i prednosti akcija implicitno definisano samom arhitekturom mreže. Vrednosti stanja i prednosti akcija se na izlazu mreže kombinuju tako da se smanji nestabilnost koja se javlja zbog promena u ciljnim vrednostima.

Reprodukcija iskustva je definisana standardnim proporcionalnim prioritizovanim uzorkovanjem kod kojeg se prioriteti računaju pomoću apsolutnih vrednosti TD grešaka odgovarajućih prelazaka. Takođe se za popravljanje raspodele uzorka koriste težine uzorkovanja prema važnosti koje se računaju na osnovu prioriteta i potom noramlizuju kako ne bi previše povećale vrednost koraka optimizacije. Na slici 3.3 je prikazan pseudokod napredne implementacije algoritma DQN.

Slika 3.3: Napredna implementacija algoritma duboka Q mreža

Alocirati memoriju iskustva $\mathcal{D}$ veličine N
Nasumično inicijalizovati parametre w
Inicijalizovati parametre w' na w
for *i=1,M* **do**
  Inicijalizovati početno stanje s_1
  Inicijalizovati nizove S, A i R veličine n
  for *t=1,T* **do**
    $a_t = \begin{cases} \text{nasumična akcija} & \text{sa verovatnoćom } \varepsilon \\ \text{argmax}_a Q_w(\phi(s_t), a) & \text{inače} \end{cases}$
    Izvršiti a_t i opaziti r_{t+1} i s_{t+1}
    Izračunati indeks $k = (t \mod n) + 1$
    Inicijalizovati $\rho \leftarrow 0$
    for *j=1,n* **do**
      Akumulirati nagradu $\rho \leftarrow R[k-j] + \gamma\rho$
      ($k-j$ cirkularno)
    end
    Sačuvati $(S[k], A[k], \rho, s_{t+1})$ u $\mathcal{D}$
    $S[k] \leftarrow s_t; A[k] \leftarrow a_t; R[k] \leftarrow r_{t+1}$
    for *j=1,|$\mathcal{D}'$|* **do**
      Uzorkovati prelaz $j \sim \mathcal{P}(j) = p_j^\omega / \sum_k p_k^\omega$
      Izračunati težinu $z_j = (N \cdot P(j))^{-\beta} / \max_k z_k$
      $y^j = \begin{cases} \rho^j & s_{t+1}^j \text{ je završno} \\ \rho^j + \gamma^n \max_a Q_{w'}(\phi(s_{t+1}^j), a) & \text{inače} \end{cases}$
      Izračunati TD grešku $\delta_j = y^j - Q_w(\phi(s_t^j), a_t^j)$
      Ažurirati prioritet $p_j \leftarrow |\delta_j| + \epsilon$
    end
    Ažurirati $w \leftarrow w + \alpha \cdot z \cdot \delta \cdot \nabla_w Q(\phi(s_t), a_t)$
    Ako je t deljivo sa C postaviti $w' = w$
  end
end

Glava 4

Eksperimenti i rezultati testiranja

Primarna svrha eksperimenata u ovom delu je ispitivanje poznatih metoda koje su primenjivane u 2D okruženjima i utvrđivanje koliko su one moćne kada se primenjuju na složenije probleme koje uključuju percepciju i donošenje odluka u 3D okruženjima. Takođe, cilj je utvrditi čvrstu osnovu za buduća istraživanja na osnovu koje će biti moguće lako izvesti ispitivanje novih metoda na istom problemu ili proširiti problem radi povećanja težine zadatka za naprednije metode.

U nastavku će biti predstavljeni rezultati ispitivanja različitih implementacija algoritma DQN. Najpre će biti opisan postupak evaluacije osnovnog DQN modela koji uključuje rezultate ispitivanja familije konfiguracija ovog algoritma koje nisu dostigle konvergenciju. Zatim, biće predstavljena konfiguracija i rezultati testiranja osnovnog modela koji je korišćen kao referentna tačka za upoređivanje performansi naprednijih implementacija. Ispitivanje naprednih implementacija uključuje rezultate testiranja performansi pojedinačnih komponenti algoritma Rainbow i njihov pojedinačni napredak u odnosu na osnovni model. Konačno, biće predstavljeni rezultati testiranja napredne implentacije algoritma koji obuhvata osnovni model DQN zajedno sa svim komponentama naprednih implementacija, kao i ispitivanje doprinosa pojedinačnih komponenti celokupnom učenju kroz odstranjivanje pojedinačnih komponenti iz najnaprednije implementacije.

Sva ispitivanja su vršena pod operativnim sistemom Manjaro na računaru sa konfiguracijom koja uključuje sledeće hardverske komponente: CPU i7-11800H, RAM veličine 16GB, GPU nVidia GeForce RTX 3060 sa 6GB sopstvene memorije i SSD kapaciteta 512GB. Ceo kod projekta je napisan na jeziku Python pomoću biblioteke PyTorch za implementaciju dubokog učenja i nalazi se zajedno sa svim rezultatima na sledećem linku: https://github.com/MrHofmann/Doom-RL.

4.1 Metrike, evaluacija i opšta podešavanja

Jedan način evaluacije performansi agenta je, kao što je sugerisano u radu [6], pomoću ukupne nagrade koju agent sakupi u toku epizode. Metrika koja se u ovim eksperimentima koristi uključuje prosek nad ukupnom akumuliranom nagradom tokom većeg broja epizoda. Prosek se računa periodično nad 200 testnih epizoda nakon svake epohe obučavanja koja iznosi 50000 koraka. Kako bi se precizno evaluiralo stanje agenta, tokom testnih epizoda je isključena eksploracija tako da se za izbor akcija koristi pohlepna politika izvedena iz odgovarajuće ε-pohlepne.

Druga korišćena metrika u eksperimentima je procenat epizoda u kojima je agent prikupio maksimalnu nagradu od 2100 poena i preživeo. Ova metrika je možda od većeg značaja jer direktno pokazuje koliko često agent ostvaruje cilj, dok je prethodna metrika korisnija pri kraju obučavanja jer omogućava lakše praćenje napretka agenta kada se njegove performanse popravljaju u malim koracima.

Problem kod prikazivanja rezultata koristeći prethodne metrike je što rezultati sadrže mnogo šuma, pa su zbog toga i rezultujući grafici nepregledni. Jedan način kako da se smanji šum je korišćenjem metrike prosečne maksimalne vrednosti akcije u stanju [33]. Međutim, iako se korišćenjem ove metrike dobijaju lepši i pregledniji grafici na kojima se može lakše videti napredak modela tokom obučavanja, nemoguće je upoređivati različite modele jer ocene akcija ne govore o stvarnom napretku agenta. To može da se odredi samo na osnovu nagrade.

Ukupno trajanje obučavanja u svim eksperimentima je podešeno na milion koraka podeljenih na 20 epoha od po 50000 koraka, pri čemu je učenje, odnosno početak izvršavanja propagacije unazad, odloženo za 2500 koraka od početka prve epohe, kako bi se omogućilo učitavanje memorije iskustva sa što više podataka iz što većeg broja epizoda i time obezbedila što veća diversifikacija reprodukcije iskustva od početka obučavanja.

Opšta podešavanja ViZDoom okruženja pod kojima su sprovođeni svi eksperimenti uključuju: originalnu rezoluciju iscrtavanja od 160x120, iscrtavanje van ekrana radi bržeg izvršavanja, onemogućen zvučni bafer jer se ne koristi, onemogućeno iscrtavanje oružja jer se ne koristi, a zaklanja vidno polje agenta, onemogućeno iscrtavanje različitih grafičkih detalja koji su računski zahtevni a nisu važni za učenje. Takođe, podešeno je preskakanje tri uzastopna frejma, tako da okruženje prosleđuje agentu svaki četvrti frejm, dok se u preostalim frejmovima podrazumeva poslednja preduzeta akcija.

4.2 Utvrđivanje osnovnog modela

Model koji je opisan u glavi 3 predstavlja osnovu za dalja unapređivanja. Međutim, postoji veliki broj parametara koje je potrebno najpre podešavati kako bi se utvrdila osnovna konfiguracija algoritma nad kojom bi se vršila nadogradnja i u odnosu na koju bi se upoređivali rezultati naprednijih implementacija. Definisanje osnovnog modela uključuje izbor metoda preprocesiranja, izbor arhitekture mreže, izbor algoritma i parametara optimizacije i podešavanje hiperparametara algoritma učenja kako bi se obezbedili ravnopravni uslovi pod kojima bi se vršilo poređenje sa naprednijim implementacijama.

Polazna tačka postupka utvrđivanja osnovnog modela je bio spoj eksperimenata iz radova [34] i [24]. Korišćena je originalna arhitektura DQN iz rada [34] sa različitim vrednostima veličine filtera i broja jedinica slojeva na scenariju prikupljanja prve pomoći iz rada [24], pri čemu je iz inicijalne implementacije agenta izostavljena kratkoročna istorija od poslednja 4 stanja.

Preprocesiranje se nije primenjivalo pri inicijalnim konfiguracijama. Korišćen je kompletan sadržaj bafera za iscrtavanje kao ulaz za mrežu čija je veličina bila podešena na veličinu rezolucije iscrtavanja od 160 piksela širine, 120 piksela visine i sva tri kanala RGB formata.

Inicijalna arhitektura mreže je bila jednostavna konvolutivna neuronska mreža koja se koristila u ranijim eksperimentima u igrama sa 2D okruženjima. Imala je dva konvolutivna sloja sa 16 filtera širine 8 i 4 piksela, pri čemu se konvolucija primenjivala sa koracima 4 i 2. Zatim, sledila su dva potpuno povezana sloja. Jedan skriveni sloj sa 256 jedinica i izlazni sloj sa 8 jedinica, po jednu za svaku moguću kombinaciju akcija. Sve jedinice su koristile ReLU nelinearnost kao aktivacionu funkciju.

Parametri učenja su incijalno takođe bili izabrani na osnovu ranijih eksperimenata na jednostavnijem scenariju. Vrednost hiperparametra umanjenja je bila podešena na $\gamma = 1.0$, korak učenja je postavljen na $\alpha = 0.01$, kapacitet memorije iskustva na 10000 prelaza i veličina uzorka iskustva na 40. Vrednost hiperparametra eksploracije se kretala od $\varepsilon = 1.0$ na početku učenja i zatim se između 100000 i 200000 koraka ravnomerno smanjivala do $\varepsilon = 0.1$. Preskakanje frejmova je takođe podešeno na osnovu ranijih istraživanja u kojima je utvrđeno da je idealna vrednost ovog parametra između 3 i 10, pri čemu je konkretno u ovom slučaju izabrana vrednost 3.

Prostor hiperparametara i konfiguracija arhitekture algoritma DQN je prevelik kako bi se vršila iscrpna pretraga. Zbog toga je njihova vrednost određena empirijski, ručnom pretragom. Za svaku komponentu algoritma početna tačka pretrage vrednosti hiperparametara je bila neka od vrednosti koje se koristile u prethodnim istraživanjima, nakon čega se vršila ručna pretraga u okolini te vrednosti kako bi se odredila neka lokalna, približno optimalna vrednost hiperparametara. U tabelama 4.1 i 4.2 su prikazane konfiguracije arhitekture mreže i vrednosti hiperparametara svih isprobanih modela, pri čemu je podebljanim brojevima označena inicijalna konfiguracija koja je korišćena u ranijim istraživanjima.

Tabela 4.1: Testirane konfiguracije arhitekture mreže inicijalnog modela - dimenzije ulaza, broj, dimenzije i korak filtera prvog i drugog konvolutivnog sloja, veličina potpuno povezanog sloja i veličina izlaznog sloja

	Arhitektura				
Konfiguracija	Ulaz	Konv1	Konv2	PP	Izlaz
1	**[3,160,120]**	**[16,8,4]**	**[32,4,2]**	**256**	**8**
2	[3,160,120]	[32,8,4]	[32,4,2]	256	8
3	[3,120,120]	[32,7,3]	[32,4,1]	800	8
4	[3,120,120]	[32,7,3]	[32,4,1]	800	8
5	[3,120,120]	[32,7,3]	[64,4,1]	800	8
6	[3,120,120]	[32,7,3]	[64,4,1]	800	8
7	[3,120,120]	[64,7,3]	[64,4,1]	1024	8
8	[3,120,120]	[64,7,3]	[64,4,1]	1024	8
9	[3,120,120]	[64,8,4]	[64,4,2]	1024	8

Tabela 4.2: Testirane vrednosti hiperparametara inicijalnog modela - umanjenje, korak učenja, odmrzavanje težina, početna i krajnja vrednost eksploracije, smanjivanje eksploracije, odložen početak učenja, veličina memorije i uzorka iskustva

Konf.	γ	α	C	ε_{start}	ε_{end}	ε_{decay}	B_{start}	$\|D\|$	$\|D'\|$
1	**1.0**	**0.01**	**100000**	**1.0**	**0.1**	**100000**	**10000**	**50000**	**40**
2	1.0	0.01	10000	1.0	0.1	100000	10000	50000	40
3	1.0	0.01	10000	1.0	0.1	100000	10000	50000	40
4	1.0	0.00025	10000	1.0	0.005	200000	10000	50000	64
5	1.0	0.00025	10000	1.0	0.005	300000	10000	50000	64
6	1.0	0.00025	15000	1.0	0.005	400000	10000	50000	64
7	1.0	0.00025	15000	1.0	0.005	500000	5000	50000	64
8	1.0	0.00025	5000	1.0	0.005	500000	5000	50000	64
9	1.0	0.00025	5000	1.0	0.005	500000	2500	50000	64

Ove konfiguracije, iako su se dobro pokazale na jednostavnijim problemima, nisu uspele da reše kretanje i upravljanje resursima kroz složeniji 3D prostor. Nijedna od ovih konfiguracija nije vodila do konvergencije modela pa je zaključeno da je potrebno implementirati moćniju arhitekturu mreže.

Arhitektura mreže osnovnog modela je bila proširena za još jedan konvolutivni sloj, dok je ulaz mreže bio modifikovan tako da može da prihvati kratkoročnu istoriju od prethodna četiri frejma sa odgovarajućim preduzetim akcijama i stanjem zdravlja agenta. Stoga je bilo odlučeno da će se dimenzije ulaza smanjiti kako se ne bi previše povećali vremenski zahtevi izvršavanja algoritama. Uveden je korak preprocesiranja koji je transformisao ulazne frejmove rezolucije 160x120 i RGB formata u kratkoročnu istoriju od četiri frejma rezolucije 84x84 sa paletom sivih boja, kao što je predloženo u [33]. Nakon toga je bilo potrebno samo još nadovezati niz vrednosti koji predstavlja drugi deo ulaza na izlaz iz poslednjeg konvolutivnog sloja koji je najpre transformisan u jednodimenzioni niz. Ostali slojevi mreže su ostali nepromenjeni, dok su parametri koji određuju veličine slojeva bili preuzeti iz [34]. Kompletna arhitektura mreže konačnog osnovnog modela je predstavljena tabelom 4.3

Nakon isprobavanja velikog broja različitih vrednosti, utvrđen je i skup vrednosti hiperparametara osnovne implementacije algoritma DQN koji će nadalje biti fiksirian radi smislenog upoređivanja sa rezultatima naprednijih implementacija. Konačne vrednosti preostalih hiperparametara osnovnog modela su prikazane u tabeli 4.4.

Tabela 4.3: Arhitektura mreže osnovnog modela

Ulaz	Dimenzije	
Prvi konvolutivni sloj	$[c, h, w]$	$[4, 84, 84]$
Prvi potpuno povezani sloj	$[c, l]$	$[4, 36]$
Konvolucija	Kanali, Veličina Filtera, Korak	
Prvi konvolutivni sloj	$[c, f, s]$	$[32, 8, 4]$
Drugi konvolutivni sloj	$[c, f, s]$	$[64, 8, 2]$
Treći konvolutivni sloj	$[c, f, s]$	$[64, 8, 1]$
Propagacija unapred	Broj Jedinica	
Potpuno povezani sloj	$[l]$	$[1024]$
Izlazni sloj	$[l]$	$[8]$

Tabela 4.4: Hiperparametri osnovnog modela

Parametar	Oznaka	Vrednost
Umanjenje	γ	1.0
Korak učenja	α	0.00025
Eksploracije	ε	$1.0 \rightarrow 0.005$
Smanjivanje eksploracije	ε_{decay}	500000
Odmrzavanje	C	5000
Početak učenja	B	2500
Veličina iskustva	$\|D\|$	50000
Uzorak iskustva	$\|D'\|$	64

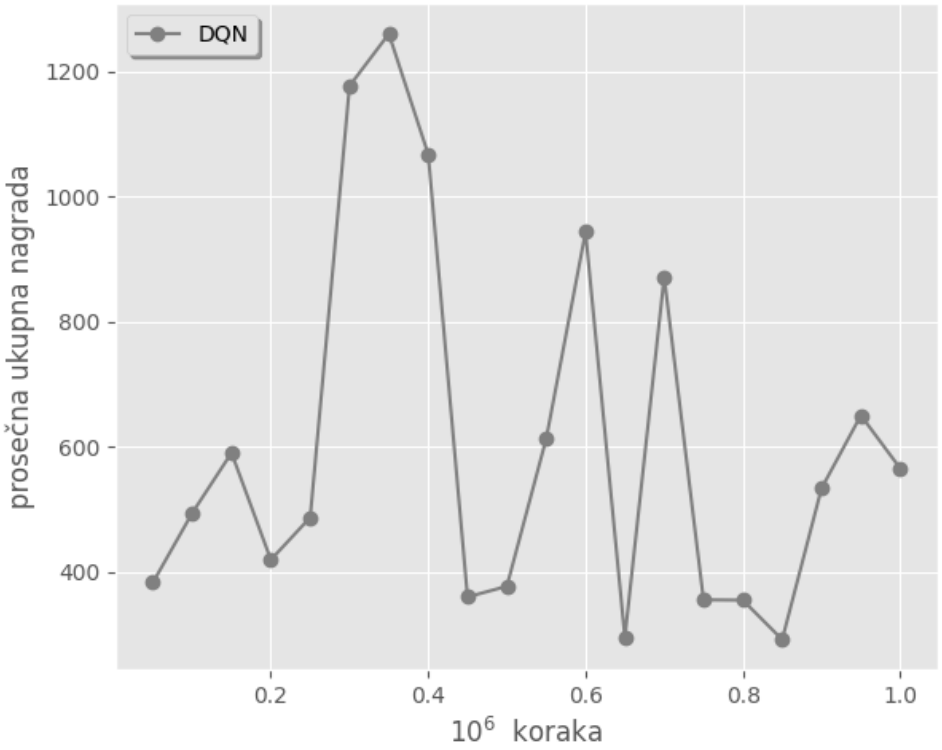
Obučavanje osnovnog modela

Osnovni model je bio obučavan na milion iteracija, što je trajalo oko 54 minuta. Ponašanje agenta tokom različitih faza testiranja je uglavnom pokazivalo preferenciju jedne akcije, posebno u početnim fazama obučavanja. Tokom većeg broja početnih iteracija agent nije uspevao da nauči osnovni zadatak kretanja kroz okruženje, sve do sredine obučavanja kada je tokom većeg dela vremena pokazivao logično ponašanje.

Nakon 400000 iteracija, agent je postigao solidnu veštinu navigacije kroz okruženje. Međutim, na osnovu izbora akcija, zaključeno je da agent nije uspeo da reši teži zadatak prepoznavanja i upravljanje resursima. Ključna stvar koju je naučio je da treba da se kreće da bi preživio, kao i da je približavanje zidovima i zaglavljivanje u ćoškovima loša situacija za njega, dok se skupljanje paketa prve pomoći dešavalo slučajno, kada bi mu se našli na putu. Takođe, bočice sa otrovom u njegovoj neposrednoj blizini nije uspeo da prepozna kao opasnost, čak i kada je vrednost njegovog stanja zdravlja bila niska, pa je uvek prelazio i preko njih, što ga je u velikom broju slučajeva i ubilo. Ipak, agent je i u ovakvom stanju uspeo u više slučajeva da sakupi maksimalnu nagradu od 2100 poena i preživi epizodu.

U trećoj trećini obučavanja je nastala nagla degradacija performansi agenta, što je pri kraju dovelo do divergentnog ponašanja, koje je u različitim fazama testiranja rezultovalo izborom samo jedne od kombinacija akcija, slično kao na početku obučavanja, kada je mreža nasumično inicijalizovana. Pošto su stanja koja se prosleđuju agentu dosta slična, agent u ovakvim situacijama ne pravi veliku razliku između njih, pa će zbog toga i vrednosti tih stanja i preduzetih akcija biti dosta slične. Stoga je agent uvek izabrao istu akciju, jer je eksploracija isključena tokom testiranja. Na slici 4.1 su prikazane performanse agenta tokom obučavanja.

Slika 4.1: Performanse osnovnog modela



4.3 Ispitivanje pojedinačnih komponenti

U ovom delu će biti predstavljeni i analizirani rezultati obučavanja četiri različita agenta. Svaki od njih nadograđuje na osnovni model po jednu komponentu napredne implementacije, što u isto vreme omogućava nezavisno testiranje i upoređivanje pojedinačnih komponenti napredne implementacije sa rezultatima testiranja osnovnog modela.

Konfiguracija i vrednosti hiperparametara osnovnog modela koje su određene u prethodnom poglavlju su bile fiksirane tokom izvođenja ovih eksperimenata, s tim što su neki elementi osnovne implementacije modifikovani kako bi se omogućila sama integracija komponenti u osnovni model. Modifikacije osnovnog modela koje su izvršene prilikom unapređivanja komponenti su bile minimalne kako bi se osigurala što manje odstupanje i pošteno poređenje sa rezultatima prethodnih i budućih testiranja.

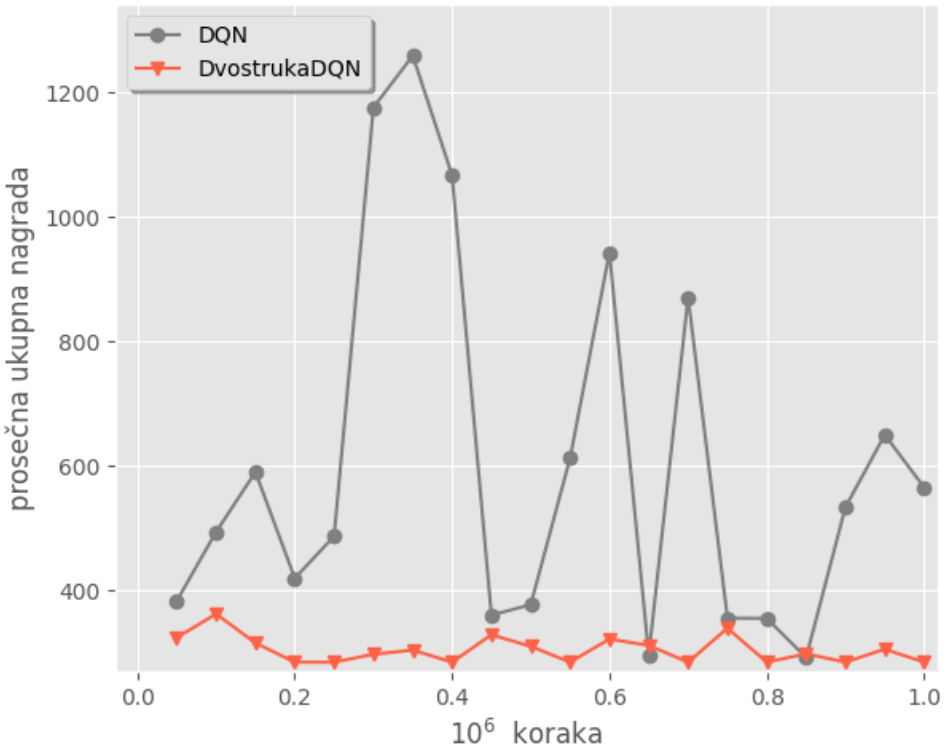
Rezultati obučavanja ovih agenata takođe obuhvataju ispitivanja i pretragu optimalnih konfiguracija hiperparametara komponenti kako bi se obezbedio maksimalan napredak u odnosu na osnovni model.

Dvostruka DQN

Performanse dvostruke DQN se pokazale kao ubedljivo najlošije od svih implementacija. Model nije ni u jednoj fazi testiranja pokazivao napredak. U početnim fazama testiranja, dok je istraživanje bilo dovoljno često, u ponašanju agenta je mogao da se vidi izbor različitih akcija. Međutim, kako se obučavanje približavalo sredini, agent je počeo da konvergira ka izboru samo jedne od kombinacija akcija, jer je nedostatak eksploracije tokom obučavanja onemogućavao agentu da pronađe bolje opcije od trenutne koju poznaje. Od ovog trenutka pa do kraja obučavanja, nije se ništa promenilo. Agent je u svakoj fazi testiranja preferirao samo jednu od mogućih kombinacija akcija. Obučavanje dvostruke DQN je trajalo 52 minuta. Na slici 4.2 su prikazane performanse agenta tokom različitih faza obučavanja.

U radu [21] pokazano je kako u nekim slučajevima dvostruko učenje ne doprinosi učenju, a nekada čak i šteti. To se javlja u situacijama kada su vrednosti akcija već potcenjene, bez uticaja dvostrukog učenja. Na primer, kao rezultat tehnike odsecanja nagrada. Pošto se u ovom slučaju nije koristilo odsecanje nagrada, pretpostavlja se da je uzrok loših performansi dvostrukog učenja nedostatak eksploracije prouzrokovan nedostatkom precenjenih vrednosti akcija.

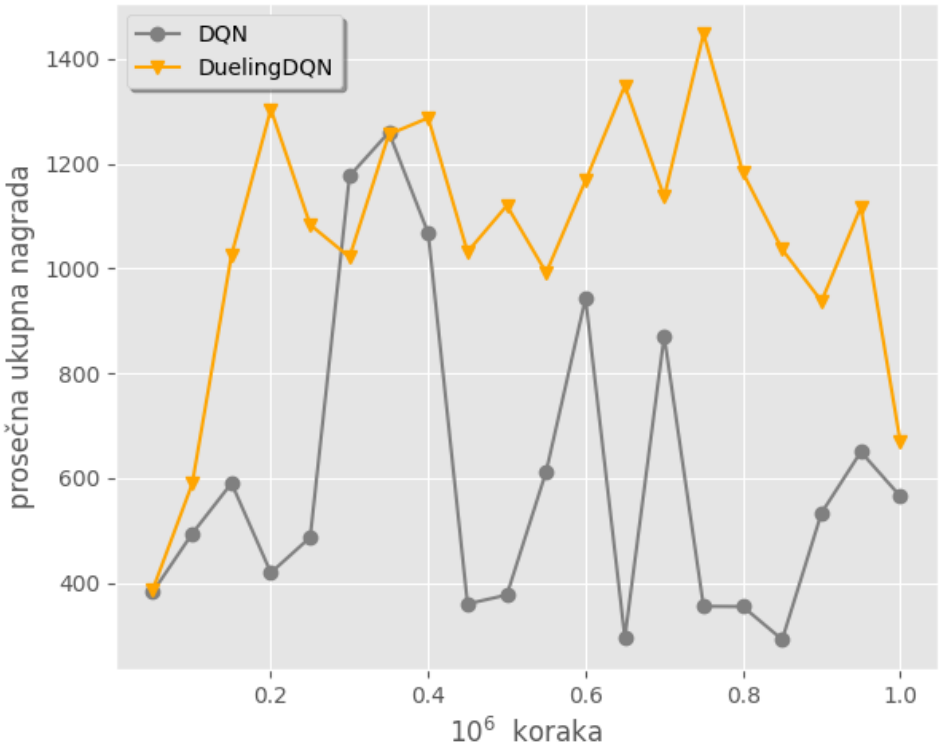
Slika 4.2: Performanse modela sa dvostrukim učenjem



Dueling arhitektura

Dueling arhitektura je jedna od komponenti koja je najviše doprinela učenju. Model koji uključuje dueling arhitekturu najviše je naučio u početnim fazama obučavanja. Već u prvoj četvrtini obučavanja je prestigao najveću prosečnu vrednost akumulirane nagrade koju je ostvario osnovni model. Međutim, zbog nestabilnosti, performanse ovog modela počinju naglo da opadaju pri kraju obučavanja, slično kao u slučaju osnovnog modela. Obučavanje ovog agenta je trajalo 63 minuta. Specijalna podešavanja implementacije dueling arhitekture izostavljaju podelu izlaza iz poslednjeg konvolutivnog sloja na dva jednaka dela kao što je navedeno u literaturi. Pošto se na izlaz iz poslednjeg konvolutivnog sloja nadovezuju ulazne vrednosti koje predstavljaju stanje zdravlja agenta i kartkoročnu istoriju poslednjih preduzetih akcija, te vrednosti bi se propagirale samo kroz jedan tok dueling arhitekture ako bi se izlaz iz konvolutivnog sloja podelio, što je na jednom od prethodnih eksperimenata vodilo do divergencije. Poređenje performansi tokom obučavanja osnovnog modela i modela koji uključuje dueling arhitekturu je prikazano na slici 4.3.

Slika 4.3: Performanse modela sa dueling arhitekturom

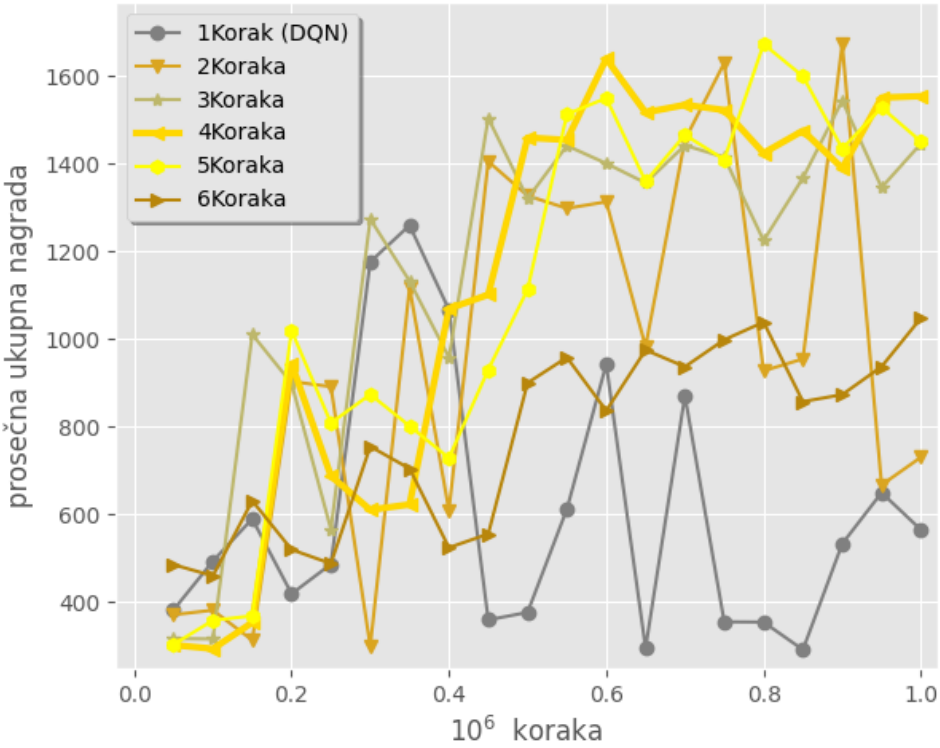


Višekoračno učenje

Jedan od glavnih nedostataka osnovnog modela je nestabilnost konvergencije. Višekoračno učenje, pored toga što je ubrzalo konvergenciju, takođe je uspelo da je u velikoj meri stabilizuje. Najvažniji doprinos ove tehnike je stabilizacija pri kraju konvergencije, u trenutku kada postoji najveća verovatnoća da će performanse modela naglo da opadnu. Ispitano je učenje za različite vrednosti broja koraka n. Na slici 4.4 je prikazano poređenje rezultata eksperimenata za šest različitih vrednosti koraka n. Na osnovu grafika se može zaključiti da najbolji kompromis između performansi i stabilnosti nude implementacije višekoračnog učenja sa vrednošću koraka između 3 i 5. Za dalja testiranja izabrana je varijanata višekoračnog učenja sa 4 koraka. Obučavanje svakog od agenata višekoračnog učenja je trajalo između 55 i 60 minuta.

Glavno poboljšanje koje se očekuje od višekoračnog učenja je veća brzina konvergencije. Međutim, na grafiku može da se vidi da u ranijim fazama nema mnogo velike razlike u brzini, gde osnovni model na trenutak prestiže višekoračne metode, sve dok ne počnu performanse da mu opadaju zbog nestabilnosti.

Slika 4.4: Performanse modela sa različitim konfiguracijama višekoračnog učenja



Prioritizovana reprodukcija iskustva

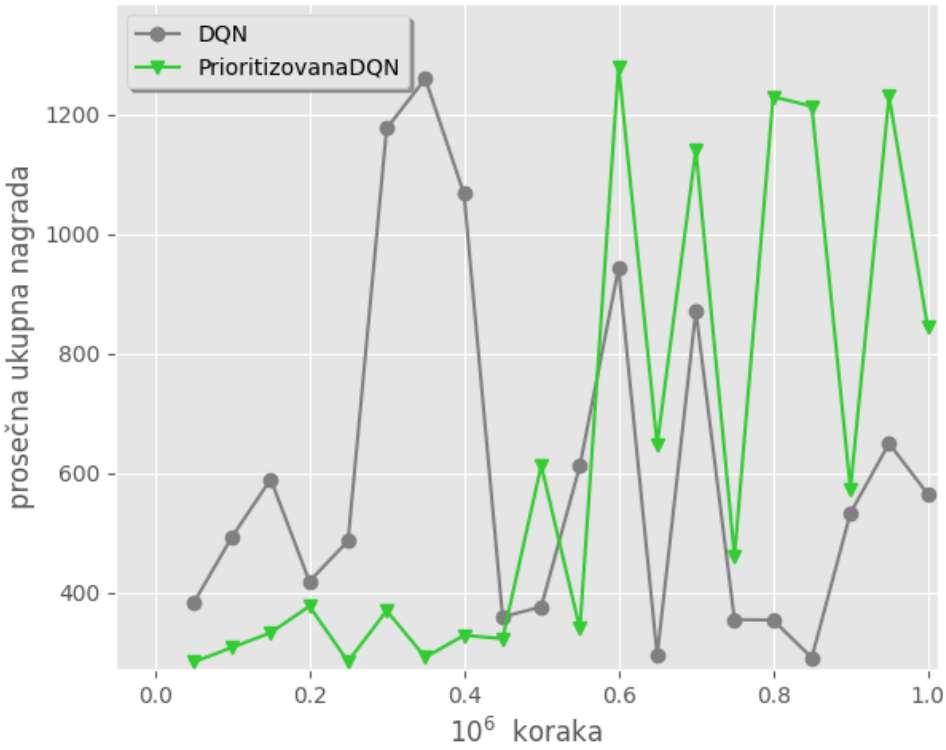
Pokrenuto je više eksperimenata za različite vrednosti hiperparametara prioritizovane reprodukcije iskustva. Podešavanje hiperparametara je bilo vršeno na osnovu već isprobanih konfiguracija iz ranijih istraživanja. Za predložene vrednosti hiperparametara nije bilo značajnih razlika u performansama modela tokom obučavanja i testiranja. Jedna stvar koja je bila karakteristična za sve ispitane modele je da je proporcionalno prioritizovano uzorkovanje usporavalo konvergenciju u početnim fazama učenja, dok je u drugoj polovini obučavanja ostvarilo veliki napredak i nadmašilo rezultate osnovne implementacije. Takođe, bio je ispitan uticaj višekoračnog učenja na prioritizovanu reprodukciju iskustva i primećeno je da proporcionalno uzorkovanje najbolje radi u kombinaciji sa 3 koraka višekoračnog učenja, ali je zbog generalno boljih rezultata koje pokazuju višekoračne metode učenja pri implementacijama koje uključuju i preostale komponente, ipak izabrano 4 koraka višekoračnog učenja. Obučavanja modela sa različitim konfiguracijama reprodukcije iskustva je trajalo između 53 i 58 minuta.

U tabeli 4.5 su prikazane vrednosti hiperparametara na različitim eksperimentima, gde su podebljanim brojevima predstavljene konačne vrednosti koje su se koristile i u preostalim ispitivanjima. Parametar ϵ predstavlja sigurnosnu vrednost koja sprečava deljenje nulom u računanju prioriteta, parametar ω uticaj prioriteta na uzorkovanje, a β_0, β_{end} i β_{steps} početnu i krajnju vrednost i broj koraka smanjivanja vrednosti parametra koji upravlja uticajem težina uzorkovanja prema važnosti na učenje. Takođe, na slici 4.5 je predstavljeno poređenje performansi osnovnog modela i modela sa proporcionalnim uzorkovanjem tokom faza testiranja u terminima prosečne prikupljene nagrade.

Tabela 4.5: Hiperparametri prioritizovane reprodukcije iskustva

ϵ	ω	β_0	β_{end}	β_{steps}
0.01	0.4	0.4	1.0	500000
0.01	0.4	0.4	1.0	1000000
0.01	0.5	0.4	1.0	700000
0.01	**0.6**	**0.4**	**1.0**	**800000**
0.01	0.7	0.4	1.0	600000
0.01	0.6	0.5	1.0	600000
0.01	0.8	0.6	1.0	500000
0.01	0.8	0.6	1.0	900000

Slika 4.5: Performanse implementacije sa prioritizovanom reprodukcijom iskustva



4.4 Ispitivanje napredne implementacije

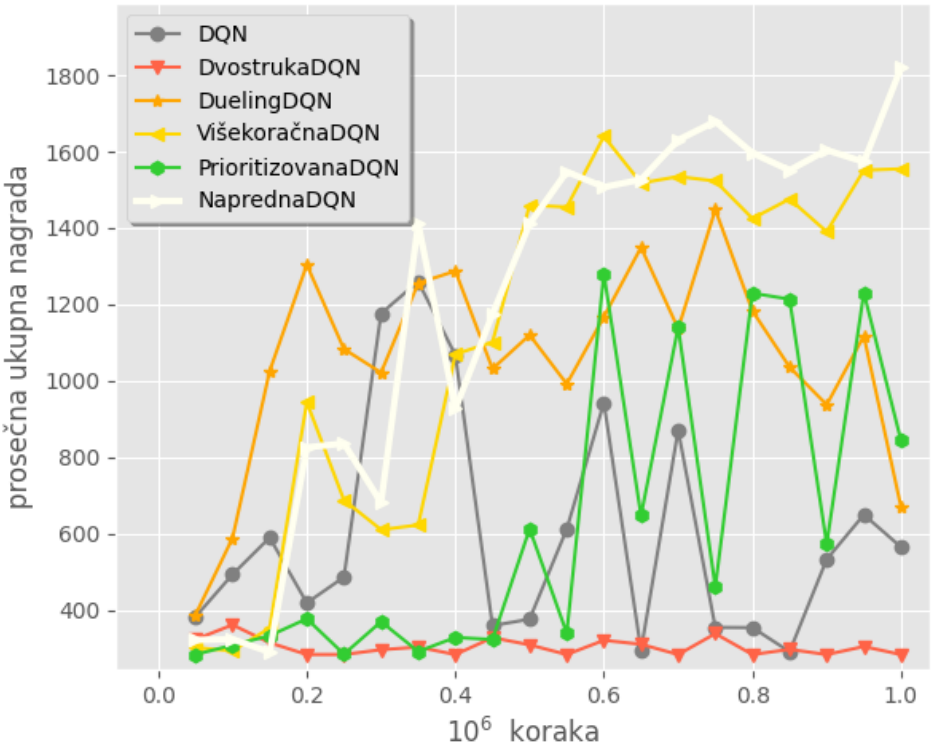
Prilikom testiranja napredne implementacije koja sadrži sva unapređenja prethodnih modela, uočeno je više zanimljivih stvari. Pre svega, od velikog je značaja uspeh koji je postigla napredna implementacija u brzini i stabilnosti konvergencije kada se poredi sa modelima koji uključuju pojedinačne komponente, kao i napredak koji je ostvarila u odnosu na osnovni model, kako u rezultatima metrika tako i u ponašanju u samoj igri.

Na samom početku obučavanja, u prvih sto hiljada iteracija, agent je jako malo uspeo da nauči. Njegovo ponašanje je pokazivalo izbor iste kombinacije akcija u svakom stanju. Pre nego što je obučavanje dostiglo dvesta hiljada koraka, agent je počeo povremeno da preduzima i preostale kombinacije akcija i od ove tačke njegovo ponašanje je počelo naglo da se popravlja. U narednih dvesta hiljada iteracija, agent je dostigao nivo performansi osnovnog modela pri konvergenciji. Počeo je da donosi smislenije odluke, ali je njegovim ponašanjem u ovoj fazi obučavanja preovladala visoka nesigurnost. U velikom broju situacija je stajao u mestu i šetao pogledom levo-desno, ne znajući u koju stranu je najbolje krenuti.

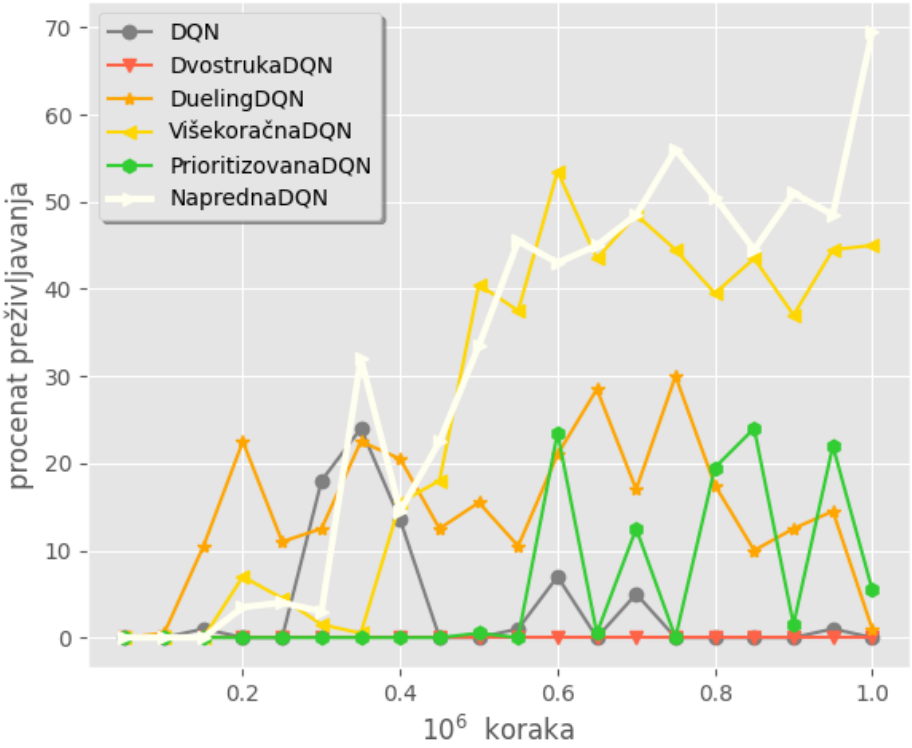
U periodu između četiristo hiljada i šeststo hiljada iteracija, agent je naučio da njegov opstanak zavisi od prikupljanja paketa prve pomoći i više ih ne prikuplja slučajno kada bi mu se našli na putu, već je njegovo kretanje uvek bilo usmereno ka njima. U ovoj fazi obučavanja takođe je počeo naglo da raste broj epizoda u kojima je agent uspeo da preživi. U preostalom delu obučavanja, agent se približavao konvergenciji. Počeo je sve više da izbegava bočice sa otrovom i dostigao je vrhunac u količini prikupljene nagrade i procentu epizoda u kojima je uspeo da preživi. Obučavanje napredne implementacije je trajalo 68 minuta.

Kompletno poređenje napredne implementacije sa svim prethodnim modelima je prikazano na slici 4.6, na kojoj je predstavljena prosečna prikupljena nagrada tokom različitih faza testiranja. Takođe, na slici 4.7 je prikazan procenat epizoda u kojima je agent prikupio maksimalnu nagradu i preživeo.

Slika 4.6: Prosečna prikupljena nagrada naprednih agenata



Slika 4.7: Procenat epizoda u kojima su agenti prikupili maksimalnu nagradu

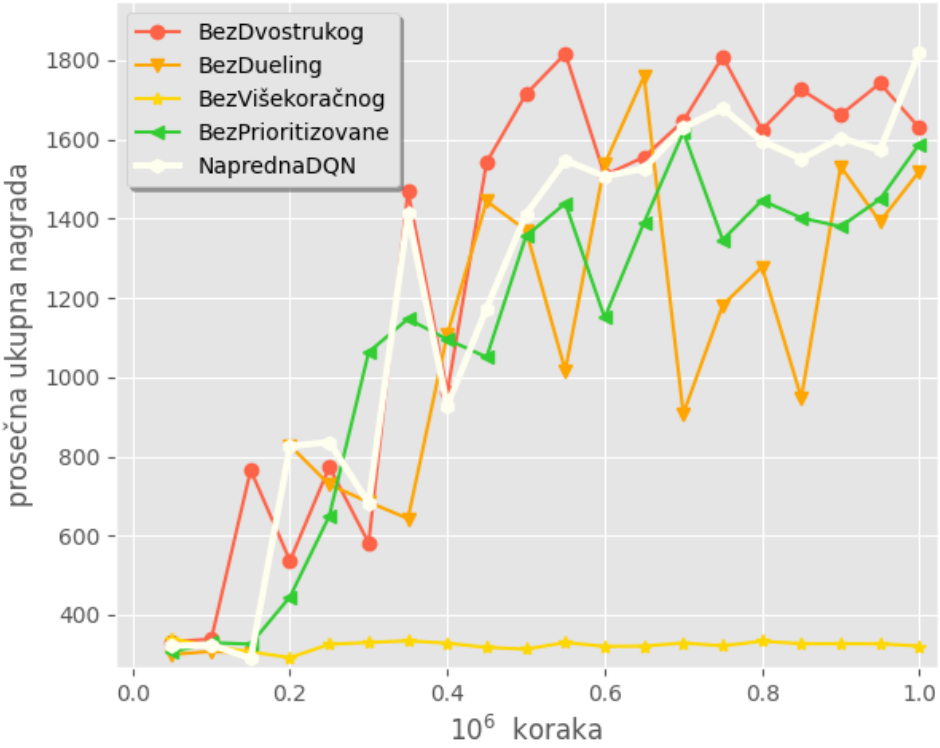


Ispitivanje komplementarnosti

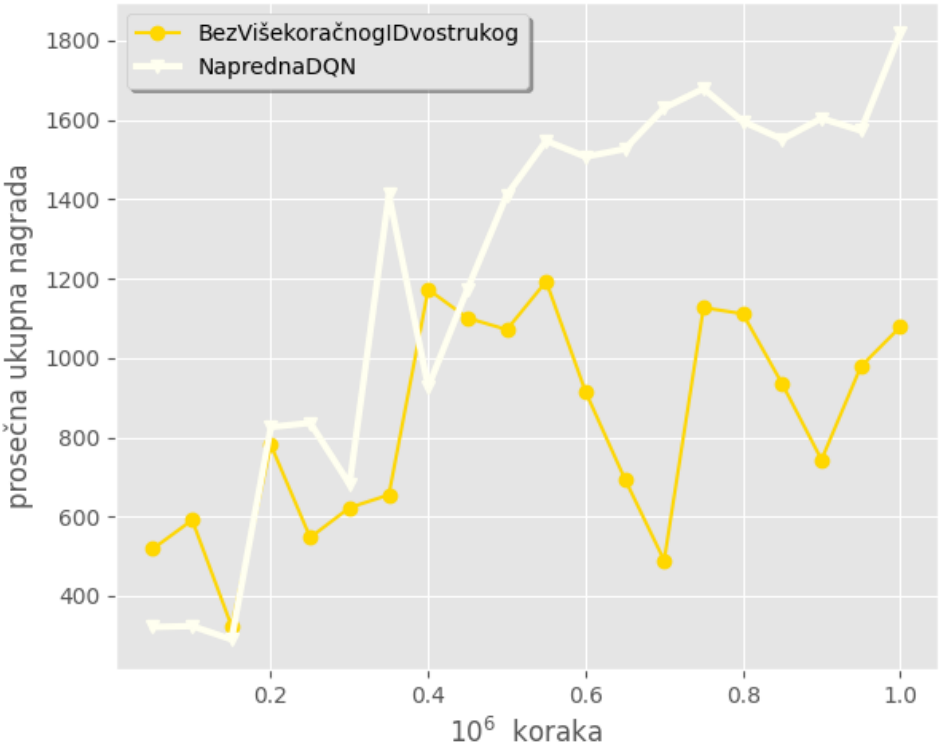
Posebno je bio uočljiv doprinos i komplementarnost komponenti prilikom izvođenja eksperimenata koji uključuju odstranjivanje komponenti. Naime, mogao je da se vidi pad performansi nakon uklanjanja bilo koje komponente izuzev dvostrukog učenja, kada su se rezultati popravili u proseku za 100 poena. Eliminacija jedne od preostalih komponenti prouzrokovala je drastičan pad brzine i stabilnosti konvergencije. Obučavanje svih ovih modela je trajalo između 60 i 72 minuta. Na slici 4.8 je predstavljen grafik koji prikazuje performanse napredne implementacije zajedno sa svojim varijantama kojima nedostaje po jedna od komponenti.

U slučaju ispitivanja modela koji ne uključuje višekoračno učenje, na osnovu grafika se može činiti da doprinos višekoračnog učenja pokriva doprinos preostalih komponenti, na osnovu čega se može zaključiti da ispitane komponente nisu komplementarne. Međutim, to nije slučaj. Uzrok loših performansi prilikom odstranjivanja višekoračnog učenja je većim delom uticaj dvostrukog učenja. Na slici 4.9 su prikazani rezultati obučavanja napredne implementacije nakon odstranjivanja višekoračnog i dvostrukog učenja.

Slika 4.8: Rezultati ispitivanja komplementarnosti



Slika 4.9: Rezultati odstranjivanja višekoračnog i dvostrukog učenja



4.5 Dodatna zapažanja, problemi i hipoteze

Analiziranjem rezultata prethodnih ispitivanja primećuju se neki problemi koji su zajednički za više modela. Jedan od glavnih problema predstavljaju nestabilnost i oscilacije. Postoji više mogućih uzroka velikih oscilacija prilikom konvergencije, ali jedan od najčešćih predstavljaju veliki koraci optimizacije. U obučavanju prethodnih modela, na veličinu koraka optimizacije može da utiče veći broj vrednosti. To su korak učenja, TD greška, gradijent i eventualno težine uzorkovanja prema važnosti. Smanjivanje bilo koje od ovih vrednosti bi moglo da pomogne u smanjivanju koraka optimizacije. Jedina vrednost od prethodno navedenih koja može značajno da utiče na povećanje koraka optimizacije je TD greška, i to prilikom smrti agenta kada postoji najveća verovatnoća da joj je apsolutna vrednost velika. Ovaj problem može lako da se reši tehnikom odsecanja nagrade.

Drugi važan problem koji se javlja je divergencija dvostrukog učenja, ali je još važniji efekat njegovog odstranjivanja iz napredne implementacije. Može se videti da se odstranjivanjem dvostrukog učenja povećavaju performanse napredne implementacije za u proseku 100 poena, što implicira da u opštem slučaju postoji još prostora za napredak u odnosu na naprednu implementaciju.

Prilikom podešavanja različitih parametara okruženja utvrđeno je da postoje konfiguracije pod kojima je agent mogao mnogo lakše da reši zadatak, kao što je definisanje parametra preskakanja frejmova na 7, ali bi onda bilo nemoguće pokazati razlike između algoritama jer bi i osnovna implementacija mogla da reši zadatak relativno lako. Takođe, nije se mogla izabrati osnovna konfiguracija parametara koja bi obezbedila povoljne uslove koji bi maksimalno odgovarali svim komponentama napredne implementacije. U velikom broju slučajeva je bilo potrebno naći kompromis. Na primer, na osnovu pretpostavke da dvostruko učenje ne postiže zadovoljavajuće rezultate zbog toga što se smanjivanjem precenjivanja akcija takođe smanjuje i istraživanje stanja i akcija iz kojih potencijalno ima šta da se nauči, moglo je da se odluči za drugačija podešavanja parmetra eksploracije, ali bi se time pokvarile performanse drugih komponenti u okviru napredne implementacije. Ili da se, na osnovu činjenice da proporcionalno uzorkovanje loše utiče na učenje kada se kombinuje sa velikim brojem koraka višekoračnog učenja, izaberu manji koraci koji akumuliraju manju nagradu, ili da se normalizuje vrednost nagrade i time obezbede bolji uslovi za uzorkovanje prema važnosti. Sva ova pitanja zahtevaju više ispitivanja.

Glava 5

Zaključak

Nakon više stotina provedenih sati i nekoliko desetina izvršenih eksperimenata u okviru ovog istraživanja, došlo se do većeg broja zaključaka vezanih za ciljeve rada, kao i do novih pitanja koja treba razmatrati u budućim istraživanjima.

Putem ovog istraživanja predstavljene su raznovrsnost i fleksibilnost u definisanju problema koje okruženje ViZDoom nudi. Time je demonstrirano da ova platforma predstavlja korisnu alternativu realističnim 3D okruženjima za izvođenje eksperimenata u oblasti vizuelnog učenja potkrepljivanjem. Takođe, detaljno su ispitane primene naprednih TD metoda koje su ranije korišćene u problemima autonomnog upravljanja u igrama sa 2D okruženjima na probleme autonomnog upravljanja u igrama sa perspektivom iz prvog lica, gde je utvrđeno da komponente naprednih implementacija algoritma DQN predstavljaju sastojke čija sinergija omogućava kreiranje moćnih agenata koji mogu da se uhvate u koštac sa ovakvim problemima.

Na osnovu ličnog uvida prilikom posmatranja igre, utisak autora je da su sposobnosti agenta nakon što je naučio da igra bile na nivou sedmogodišnjeg deteta. Treba imati na umu da scenario koji je agent naučio da reši nije jednostavan i zahteva određeni stepen koncentracije čak i za prosečnu odraslu osobu. Paketi prve pomoći se ne pojavljuju preterano često, pa agent nema puno vremena za donošenje odluka. Takođe, važno je napomenuti da je agent, pored navigacije kroz 3D okruženje, naučio da rešava teži zadatak upravljanja resursima. Međutim, postoje situacije u kojima nije jasno da li je agent sposoban da donese strateške odluke višeg nivoa kao što je uzimanje bočica sa otrovom ako se odmah iza njih nalazi paket prve pomoći, a pritom vrednost stanja zdravlja agenta nije niska. To što jeste sigurno je da je agent naučio da su bočice sa otrovom opasne i u opštem

slučaju ih treba izbegavati.

Ispitivanjem pojedinačnih komponenti napredne implementacije algoritma DQN i upoređivanjem sa rezultatima ispitivanja osnovnog modela, zabeležen je veliki napredak u performansama u većini slučajeva. Uticaj svake komponente je naročito bio vidljiv nakon odstranjivanja komponenti iz napredne implementacije. Naime, primećena je veća stabilnost višekoračnog učenja, brže učenje dueling arhitekture i sprečavanje preuranjene konvergencije proporcionalnim uzorkovanjem. Najvažnije od svega je da je odstranjivanjem dvostrukog učenja utvrđeno postojanje dodatnog prostora za napredak u odnosu na naprednu implementaciju. Međutim, otkriven je i veći broj problema kao što je visoka nestabilnost tokom obučavanja, loše performanse prilikom kombinovanja različitih konfiguracija komponenti i u nekim slučajevima i divergencija algoritma. Sva ova pitanja zahtevaju detaljnije istraživanje i analizu.

Glavno pitanje je kuda krenuti dalje? Ovim istraživanjem je utvrđena čvrsta osnova koja omogućava lako poređenje sa prethodnim istraživanjima i koja ujedno predstavlja odskočnu dasku za buduće poduhvate u ovoj oblasti. Naredni korak bi bio pozabaviti se jednim od mnogih problema koji su otkriveni tokom prethodnih eksperimenata i pokušati ih eliminisati pre nego što bi se uvodile nove stvari. Takođe, poželjno je pronaći čista algoritamska rešenja koja se ne oslanjaju na tehnike poput kratkoročne memorije stanja na ulazu mreže ili preskakanje frejmova. Zatim, naredna istraživanja bi mogla da uključuju ispitivanje preostalih komponenti algoritma Rainbow, kao i pretragu optimalne konfiguracije algoritma koja postiže najbolji mogući rezultat, dok bi jedan od dugoročnih ciljeva mogao biti definisanje novih scenarija, povećavanje težine zadatka i proširivanje skupa akcija koje agent može da preduzme, ili čak uvesti veći broj zadataka i ispitati mogućnost učenja više poslova odjednom pomoću složenijih arhitektura koje uključuju upotrebu rekurentnih neuronskih mreža. Mogućnosti su beskrajne...

Literatura

[1] David Abel, Alekh Agarwal, Fernando Diaz, Akshay Krishnamurthy, and Robert E. Schapire. Exploratory gradient boosting for reinforcement learning in complex domains. *arXiv:1605.02097v2*, 2016.

[2] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Recent Advances in Robot Learning*, pages 163–187, 1996.

[3] Minoru Asada, Eiji Uchibe, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. A vision-based reinforcement learning for coordination of soccer playing behaviors. *Proceedings of AAAI-94 Workshop on AI and A-life and Entertainment*, pages 16–21, 1994.

[4] Leemon C. Baird. Advantage Updating. *Technical Report WL-TR-93-1146*, 1993.

[5] Etienne Barnard. Temporal-Difference Methods and Markov Models. *IEEE Transactions on Systems Man and Cybernetics*, 23(2):357–365, 1993.

[6] Marc Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *arXiv:1207.4708v2*, 2012.

[7] Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning. *ArXiv:1707.06887v1*, 2017.

[8] Yoshua Bengio. Learning Deep Architectures for AI. *Fondations and Trends in Machine Learning*, 2(1):1–55, 2009.

[9] Nicolas Cole, Sushil J Louis, and Chris Miles. Using a genetic algorithm to tune first-person shooter bots. *Evolutionary Computation*, 1:139–145, 2004.

[10] Giuseppe Cuccu, Matthew Luciw, Jürgen Schmidhuber, and Faustino Gomez. Intrinsically motivated neuroevolution for vision-based reinforcement learning. *Development and Learning (ICDL), 2011 IEEE International Conference on*, 2:1–7, 2011.

[11] George Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[12] Mark Dawes and Richard Hall. Towards using first-person shooter computer games as an artificial intelligence testbed. *Knowledge-Based Intelligent Information and Engineering Systems*, pages 276–282, 2005.

[13] Abdennour El Rhalibi and Madjid Merabti. A hybrid fuzzy ANN system for agent adaptation in a first person shooter. *International Journal of Computer Games Technology*, 2008.

[14] Meire Fortunato, Mohamm Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Graves Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration. *ArXiv:1706.10295v3*, 2017.

[15] Chris Gaskett, Luke Fletcher, and Alexander Zelinsky. Reinforcement learning for a vision based mobile robot. *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, 1:403–409, 2000.

[16] Benjamin Geisler. *An empirical study of machine learning algorithms applied to modeling player behavior in a first person shooter video game.* PhD thesis, University of Winsocnsin-Madison, 2002.

[17] Frank G. Glavin and Michael G. Madden. DRE-Bot: A hierarchical First Person Shooter bot using multiple Sarsa(λ) reinforcement learners. *Computer Games (CGAMES), 2012 17th International Conference on*, pages 148–152, 2012.

[18] Frank G. Glavin and Michael G. Madden. Adaptive Shooting for Bots in First Person Shooter Games Using Reinforcement Learning. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1:180–192, 2015.

[19] Mance Harmon, Leemon C. Baird, and A. Harry Klopf. Advantage Updating Applied to a Differrential Game. *NIPS*, 1996.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852v1*, 2015.

[21] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *ArXiv:1710.02298v1*, 2017.

[22] Kurt Hornik. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, Vol. 2:359–366, 1989.

[23] Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[24] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning. *arXiv:1605.02097v2*, 2016.

[25] Nate Kohl and Peter Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. *ICRA*, volume 3, 2004.

[26] Jan Koutnı⊠k, Jürgen Schmidhuber, and Faustino Gomez. Intrinsically motivated neuroevolution for vision-based reinforcement learning. *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 541–548, 2014.

[27] Guillaume Lample and Devendra Singh Chaplot. Playing FPS Games with Deep Reinforcement Learning. *arXiv:1609.0552v2*, 2018.

[28] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. *IJCNN*, pages 1–8, 2010.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.

[30] Long-Ji Lin. Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3-4):293–321, 1992.

[31] Michelle McPartland and Marcus Gallagher. Reinforcement Learning in First Person Shooter Games. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(1):43–56, 2011.

[32] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *ArXiv:1602.01783v2*, 2016.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *ArXiv:1312.5602*, 2013.

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[35] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous Inverted Helicopter Flight via Reinforcement Learning. *Experimental Robotics*, 2006.

[36] Andrew Y. Ng, Daishi Harada, and Stuart Russel. Policy Invariance Under Rewad Transformations: Theory and Application to Reward Shaping. *ICML*, 1999.

[37] Martin Riedmiller. Neural Fitted Q Iteration-First Experience With a Data Efficient Neural Reinforcement Learning Method. *Machine Learning: ECML 2005*, pages 317–328, 2005.

[38] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *ArXiv:1511.05952v4*, 2016.

[39] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. *ICANN*, 4131:632–640, 2006.

[40] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. *JAIR*, 16:105–133, 2002.

[41] Megan Smith, Stephen Lee-Urban, and Héctor Muñoz-Avila. RETALIATE: learning winning policies in first-person shooter games. *Proceedings of the National Conference on Artificial Intelligence*, 22:1801, 2007.

[42] Tony C Smith and Jonathan Miles. Continuous and Reinforcement Learning Methods for First-Person Shooter Games. *Journal on Computing (JoC)*, 1(1), 20014.

[43] Alexander L. Strehl, Lihonh Li, Eric Wiewiora, John Langford, and Michael L. Littman. PAC Model-Free Reinforcement Learning. *ICML*, 2006.

[44] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. The MIT Press, Cambridge, Massachussetts, 2018.

[45] Gerald Tesauro. TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, 6(2):215–219, 1994.

[46] Sebastian Thrun and Anton Schwartz. Issues in Using Function Approximation for Reinforcement Learning. *Proceedings of the 1993 Connectionist Models Summer School*, 1993.

[47] Hado van Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems 23 (NIPS 2010)*, 23:2613–2621, 2010.

[48] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *ArXivi:1509.06461v3*, 2015.

[49] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. *ArXiv:1511.06581v3*, 2016.

[50] Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[51] Dimitry Yarotski. Universal Approximations of Invariant Maps by Neural Networks. *Constructive Approximation*, 55:407–474, 2022.

Biografija autora

Zovem se Matei Jon Stanču i rođen sam 2.8.1990. godine u Vršcu. Završio sam osnovnu školu Olga Petrov Radišić u Vršcu sa skroz odličnim uspehom i gimnaziju Borislav Petrov Braca u Vršcu sa odličnim uspehom. Osnovne studije smera informatika na matematičkom fakultetu u Beogradu sam završio 2015. godine sa ocenom 7.52. Iste godine sam upisao master studije smera informatika, takođe na matematičkom fakultetu u Beogradu. Trenutno radim kao sistemski i mrežni administrator u data centru u Vršcu, a radnog iskustva imam i kao programer u oblasti mašinskog učenja. U slobodno vreme volim da čitam, programiram i igram video igre. Od svoje desete godine se aktivno bavim sportom, najveći deo vremena američkim fudbalom, a trenutno samo trčanjem. Pored srpskog jezika, tečno govorim engleski i rumunski.