Implementacija i validacija algoritma za poticano strojno učenje

Mikan, Bruno

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1404

IMPLEMENTACIJA I VALIDACIJA ALGORITMA ZA POTICANO STROJNO UČENJE

Bruno Mikan

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1404

IMPLEMENTACIJA I VALIDACIJA ALGORITMA ZA POTICANO STROJNO UČENJE

Bruno Mikan

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1404

Pristupnik:	**Bruno Mikan (0036543114)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Damir Pintar

Zadatak: **Implementacija i validacija algoritma za poticano strojno učenje**

Opis zadatka:

Poticano učenje je grana strojnog učenja koja se bavi razvijanjem algoritama i tehnika koje omogućuju računalima da autonomno uče kako donositi odluke ili izvoditi određene zadatke na temelju iskustva, bez eksplicitnog programiranja. Ova vrsta učenja temelji se na sustavima nagrađivanja ili kaznjavanja na temelju rezultata njihovih akcija te iterativnom procesu poboljšanja ponašanja na osnovi povratnih informacija iz okoline. Vaš zadatak jest proučiti moderne algoritme za poticano strojno učenje te kroz vlastiti programski kod implementirati inačicu odabranog algoritma te provesti komparativnu analizu dobivenih rezultata na odabranoj domeni uporabe sa onima dobivenim preko gotovih implementacija.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

1. Uvod

Kroz posljednjih nekoliko desetljeća razvoj umjetne inteligencije doživio je znatne napretke. Kroz želju za rješavanjem sve složenijih problema posvetili smo znatnu pozornost razvoju metoda koje imaju sposobnost učenja te nam se predstavljaju kao alati kojima možemo postići rezultate koje ne bismo mogli klasičnim algoritmima i metodama programiranja. Kroz takav brzi razvoj, današnje strojno učenje se uspostavilo u tri glavne grane: nadzirano učenje, nenadzirano učenje te, kao tema ovog rada, poticano učenje.

Iako metodama nadziranog i nenadziranog učenja možemo postići vrlo dobre rezultate u rješavanju raznolikih problema, pri pokušaju korištenja tih modela u situacijama koje zahtijevaju direktnu interakciju s okolinom te učenjem sličnom učenju bioloških organizama nailazimo na prepreke. U takvim problemima poticano učenje se pokazuje kao vrlo korisna i sposobna metoda strojnog učenja.

Dok nadzirano učenje možemo efektivno primijeniti na situacijama gdje imamo već označeni skup podataka kako bismo mogli generalizirati na neviđene podatke, te nenadzirano u situacijama gdje iz skupa podataka želimo pronaći ranije nepoznate strukture, u problemima interakcije s okolinom nam se takvi obrasci u pravilu ne pokazuju. Potreban nam je pristup gdje iskustvom, kroz uspjehe i neuspjehe, donosimo zaključke o optimalnim postupcima u ranije viđenim ili sličnim situacijama. Na taj način algoritam može kroz interakciju s okolinom postepeno učiti kako bi naučio ponašanje koje optimalno rješava neki problem koji mu okolina predstavlja svojim odgovorima.

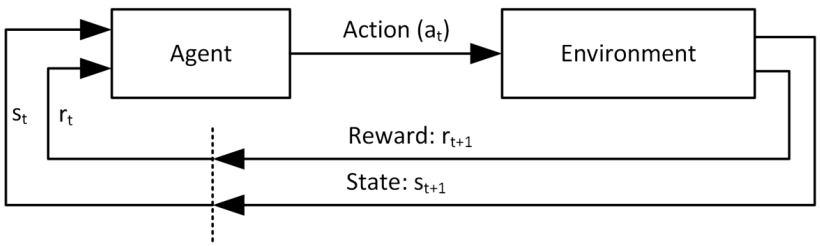
U prvom poglavlju dan je uvod u problematiku. Kroz drugo poglavlje ovaj rad će se fokusirati na definiranje te objašnjavanje osnovnih pojmova i metoda poticanog učenja. Korištenjem konačnog Markovljevog procesa odlučivanja, postavit će se matematički temelji koji čine osnovu poticanog učenja. Također, predstavit će se metode kojima se

mogu riješiti problemi poticanog učenja na diskretnim skupovima stanja. U trećem poglavlju opisat će se primjena metode poticanog učenja na praktičnom primjeru. Dati će se pregled implementacije okoline zajedno sa modelom nagrade te će se opisati implementacija algoritma poticanog učenja. U četvrtom poglavlju predstavit će se konfiguracija algoritma zajedno sa modelom evaluacije i ocijene uspješnosti. Provesti će se usporedba implementiranog algoritma s već razvijenim algoritmom preuzetim iz programske biblioteke. Kroz model evaluacije i ocijene uspješnosti donijeti će se zaključci o uspješnosti implementiranog algoritma. U petom poglavlju navesti će se mogući smjerovi daljnjeg razvoja implementirane okoline. Na kraju, u šestom poglavlju, predstavit će se sažetak rezultata zajedno sa zaključkom rada.

2. Poticano učenje

2.1. Pregled osnovnih pojmova

Poticano učenje zasnovano je na interakciji modela s okolinom. Sustav je moguće predočiti agentom koji predstavlja algoritam poticanog učenja. Agent se kroz interakciju s okolinom može pronaći u nekom stanju s_t iz konačnog ili beskonačnog skupa stanja S, gdje t predstavlja neki trenutak. U svakom stanju agent može odabrati i poduzeti neku akciju a iz skupa stanja $A(s)$. Odabirom akcije a agent deterministički ili stohastički s nekom vjerojatnosti p prelazi u stanje s_{t+1}, često označeno kao s', te pritom od okoline dobiva nagradu r iz skupa R. Navedeno ponašanje može se vidjeti na slici 2.1. Skup stanja S može biti diskretan ili kontinuiran. Problemi na diskretnim skupovima stanja, pretpostavljeno da kardinalitet skupa stanja ne nadilazi raspoložive računalne sposobnosti, učinkovito se mogu riješiti korištenjem tabličnih metoda poticanog učenja koje su primaran naglasak ovoga rada. U slučaju kontinuiranih skupova stanja bolji pristup je korištenje aproksimacijskih metoda.



Slika 2.1. Model interakcije agenta i okoline, preuzeto iz [1]

Glavni cilj agenta je maksimizirati očekivanu dugoročnu nagradu odabirom odgovarajućih akcija. Strategija odabira akcije u danom stanju naziva se politikom [2], te ona predstavlja vjerojatnost odabira akcije a u stanju s, označujemo ju s $\pi(a|s)$. Ako je vjerojatnost odabira akcije u svakom stanju uvijek jednaka 1, kažemo da je politika de-

deterministička. Agent za određenu politiku ima cilj izračunati i procijeniti vrijednosnu funkciju $v(s)$ svakog stanja s. Vrijednosna funkcija $v(s)$ predstavlja očekivanu povratnu vrijednost koju agent može ostvariti iz stanja s slijedeći politiku π, ona efektivno predstavlja agentovu mogućnost da uzima u obzir budućnost te odabire akcije koje možda u tom trenutku ne daju najveću nagradu već rezultiraju većom povratnom vrijednošću i govori koliko je dobro agentu biti u tome stanju. Također, uz vrijednosnu funkciju stanja može se razmatrati i vrijednosna funkcija uređenog para nekog stanja i akcije $q_\pi(s,a)$ za određenu politiku π, koja ima sličnu interpretaciju. Tablične metode imaju mogućnost izračuna točno optimalne vrijednosne funkcije za optimalnu politiku π dok aproksimacijske metode na temelju karakteristika stanja procjenjuju vrijednosnu funkciju za to stanje. Opisan proces se zove evaluacija politike i ključan je dio poticanog učenja.

Paralelno evaluaciji politike bitan je proces poboljšanja politike. Naime, nakon što je procijenjena vrijednosna funkcija v_π za određenu politiku π, agent ima mogućnost odabrati akciju $\pi'(s) = a$ u stanju s za koju vrijedi $q_\pi(s,a) \geq v_\pi(s)$. Odabir akcije $a \neq \pi(s)$ svaki puta kada se agent pronađe u stanju s, ili drugim riječima zamjene politike π s π' bolje je ili jednako dobro od slijeđenja politike π. Dokaz ove činjenice može se pronaći u [3]. Iteracija evaluacije i poboljšanja politike temeljni je proces u algoritmima poticanog učenja te omogućuje pronalaženje optimalne politike.

2.2. Konačan Markovljev proces odlučivanja

Kako bismo mogli formalno opisati metodu poticanog učenja problem opisujemo u idealiziranom pogledu korištenjem konačnog Markovljevog procesa odlučivanja, (eng. *Markov Decision Process* - MDP). Konačni MDP karakteriziraju konačan skup stanja S, akcija A i nagrada R te definirane vjerojatnosti prijelaza.

Prijelaze u Markovljevom procesu odlučivanja moguće je opisati na formalan način koristeći sljedeću funkciju.

$$\sum_{s' \in S} \sum_{r \in R} p(s', r|s, a) = 1, \forall s \in S, \forall a \in A(s) \tag{2.1}$$

Funkcija p prima 4 argumenta koje preslikava na interval $[0,1]$. Argumenti s', r, s

i a redom predstavljaju: sljedeće stanje u koje okolina može prijeći, nagradu dobivenu prijelazom, trenutno stanje te akciju koja se može poduzeti u trenutnom stanju. Tako definirane vjerojatnosti p Markovljevog procesa odlučivanja u potpunosti karakteriziraju ponašanje okoline [3]. Iz jednadžbe 2.1 vidljiv je zahtjev da sljedeće stanje s' i nagrada r ovise samo o trenutnom stanju s i akciji a. Može se primijetiti kako ponašanje takvog sustava zadovoljava Markovljevo svojstvo.

Kao što je već navedeno, cilj agenta je maksimizirati očekivanu povratnu vrijednost. Povratnu vrijednost definiramo s pomoću dobivenih nagrada na sljedeći način:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

Gdje parametar γ predstavlja faktor popusta ili umanjenja nagrade za kojeg vrijedi $0 \leq \gamma \leq 1$, a R_{t+k+1} predstavlja nagradu u trenutku $t + k + 1$. Raspisivanjem 2.2 kroz nekoliko koraka i izlučivanjem faktora γ dobivamo jednadžbu u rekurzivnom obliku:

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2.3}$$

Rekurzivni oblik lijepo ilustrira kako povratna vrijednost u trenutku t ovisi o dobivenoj nagradi u sljedećem trenutku $t + 1$ i povratnoj vrijednosti u tom sljedećem trenutku te kako faktorom popusta kontroliramo koliko agent pridružuje važnost nagradama u budućnosti. Što se faktor popusta više približava vrijednosti 1 agent više promatra odvijanje procesa u budućnost, dok u slučaju gdje faktor popusta poprima vrijednost jednaku 0 agent promatra samo trenutnu nagradu. Optimalna vrijednost faktora popusta ovisi o problemu te pri izgradnji modela vrijedi isprobavati različite vrijednosti kako bi se agent što bolje ponašao.

Interakcije agenta s okolinom dijele se u epizodične i kontinuirane. Epizodične slučajeve karakterizira terminalno stanje u kojem epizoda završava. Epizodične se mogu zamisliti kao igru agenta i protivnika gdje epizoda prirodno završava pobjedom ili gubitkom agenta te nova započinje u početnom stanju igre. Kontinuirana interakcija se može zamisliti kao agent koji kontrolira iznos napona na nekom elektromotoru gdje se rad

sustava smatra neprestanim. U epizodičnom slučaju faktor popusta je moguće izostaviti ili postaviti na 1 budući da je suma konačna, a sve nagrade nakon terminacije smatrane su jednakim 0. Također, u epizodičnim slučajevima, jednadžba 2.1 se redefinira tako da s ne može biti terminalno stanje. U kontinuiranim slučajevima faktor popusta se mora postaviti na vrijednost manju od 1 za koju suma konvergira pod uvjetom da je niz R_{t+k+1} omeđen. Ovo je istina budući da je red geometrijski te za njega vrijedi da konvergira u slučaju kada je $|\gamma| < 1$.

Već je navedeno da vrijednosna funkcija $\upsilon(s)$ predstavlja očekivanu povratnu vrijednost u nekom stanju s. Formalno se može zapisati na sljedeći način:

$$\upsilon_\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right], \forall s \in S \tag{2.4}$$

Slično za uređeni par stanja i akcije $q_\pi(s, a)$ v rijedi:

$$q_\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right], \forall s \in S, \forall a \in A(s) \tag{2.5}$$

Vrijednosne funkcije za koje vrijedi da maksimiziraju povratnu vrijednost se nazivaju optimalnim vrijednosnim funkcijama i označavaju se s $\upsilon^*(s)$ i $q^*(s, a)$.

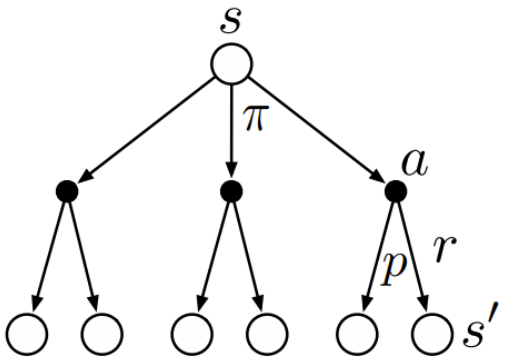
Vrijednosnu funkciju se može zapisati i u sljedećem obliku:

$$\upsilon_\pi(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma \upsilon_\pi(s')], \forall s \in S \tag{2.6}$$

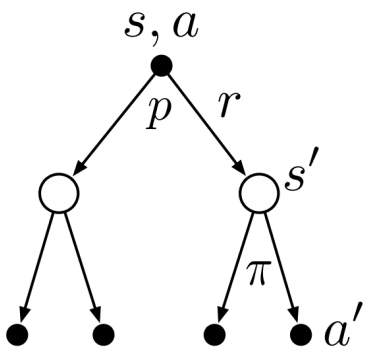
Ovako zapisana vrijednosna funkcija zove se Bellmanova jednadžbom za vrijednosnu funkciju stanja. Ona zapravo predstavlja sustav linearnih jednadžbi. Analogno, može se zapisati Bellmanova jednadžba za vrijednosnu funkciju stanja i akcije na sljedeći način:

$$q_\pi(s, a) = \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma \sum_{a' \in A(s')} \pi(a'|s') q_\pi(s', a')], \forall s \in S, \forall a \in A(s) \tag{2.7}$$

Ove jednadžbe ponovno pokazuju rekurzivnu prirodu procesa poticanog učenja. Vrijednost stanja u trenutku t moguće je izračunati na temelju vrijednosti svih dostižnih stanja u trenutku $t+1$. Detaljnije ove jednadžbe se mogu ilustrirati na temelju takozvanih povratnih dijagrama 2.2. i 2.3.



Slika 2.2. Povratni dijagram za v_π, preuzeto iz [3]



Slika 2.3. Povratni dijagram za q_π, preuzeto iz [3]

Stanja su označena praznim čvorovima, dok akcije punim. Promatranjem ovih dijagrama ponašanje procesa postaje jasnije. Politika π svakoj od akcija a pridodaje vjerojatnost odabira $\pi(a|s)$. Također, odabirom svake akcije, prema nekoj vjerojatnosti sustav može prijeći u jedno od sljedećih stanja i pritom agentu vratiti nagradu r.

Činjenicu kako Bellmanova jednadžba za vrijednosnu funkciju stanja stvarno predstavlja očekivanje može se ilustrirati sljedećom konstrukcijom. Može se uzeti da slučajna varijabla X poprima vrijednosti $[r+\gamma v_\pi(s')]$, tada vjerojatnost slučajne varijable X iznosi $\pi(a|s)p(s',r|s,a)$. Konstruira se tablica:

$$X \sim \begin{pmatrix} r_1+\gamma v_\pi(s'_1) & r_2+\gamma v_\pi(s'_2) & \cdots & r_n+\gamma v_\pi(s'_n) \\ \pi(a_1|s)p(s'_1,r_1|s,a_1) & \pi(a_2|s)p(s'_2,r_2|s,a_2) & \cdots & \pi(a_n|s)p(s'_n,r_n|s,a_n) \end{pmatrix}$$

Gdje $r_i, r_j, i \neq j$ ne moraju nužno biti različiti, $s'_1, s'_2, ..., s'_n$ predstavljaju sva moguća sljedeća stanja te je zajedno uz svaki a_i dopušteno pojavljivanje više sljedećih stanja. Množenjem vjerojatnosti i vrijednosti slučajne varijable po svakom stupcu i zbrojem dobivenih vrijednosti dobije se standardni oblik očekivanja zapisan kao težinska aritmetička sredina.

Kako bi zaključili opis Markovljevog procesa odlučivanja valja još spomenuti izraze

za optimalne vrijednosne funkcije stanja te uređenog para stanja i akcije. Optimalna vrijednosna funkcija stanja definirana je kao:

$$\upsilon^*(s) = \max_{\pi} \upsilon_\pi(s), \forall s \in S \tag{2.8}$$

dalje ovu vrijednost je moguće izraziti promatrajući ju na način da je vrijednost stanja pod optimalnom politikom jednaka najvećoj očekivanoj povratnoj vrijednosti za najbolju akciju u tom stanju. [3] Pišemo:

$$\begin{aligned} \upsilon^*(s) &= \max_{a \in A(s)} q_{\pi^*}(s,a) \\ &= \max_{a \in A(s)} \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma \upsilon^*(s')], \forall s \in S \end{aligned} \tag{2.9}$$

Analogno, za optimalnu vrijednosnu funkciju stanja i akcije vrijedi:

$$q^*(s,a) = \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma \max_{a' \in A(s')} q^*(s', a')], \forall s \in S, \forall a \in A(s) \tag{2.10}$$

Ovako izražene optimalne vrijednosne funkcije predstavljaju Bellmanove jednadžbe za optimalne vrijednosne funkcije stanja i uređenog para stanja i akcije. One se mogu riješiti kao sustav linearnih jednadžbi, i za konačni Markovljev proces odlučivanja imaju jedinstveno rješenje. Rješavanjem sustava dolazi se do egzaktnih vrijednosti optimalnih vrijednosnih funkcija. Naravno, lako je naslutiti da u praksi takav pristup porastom broja stanja vrlo brzo postaje vrlo neučinkovit. Iz tog razloga razvijene su mnoge metode koje iterativno procjenjuju vrijednosne funkcije te konvergiraju prema optimalnim, konačnim vrijednostima.

Ponašanje sustava nakon izračunatih optimalnih vrijednosnih funkcija odvija se prema optimalnoj politici. Njih može biti više, politika se smatra optimalnom ako odabire akcije a za koje vrijedi da je za njih postignut maksimum prema Bellmanovim jednadžbama

optimalnosti. [3] Dakle, agent koji iz trenutnog stanja provjeri sve akcije te odabere one koje mu se čine najboljima ponašat će se optimalno.

2.3. Tablične metode rješavanja MDP problema

Spomenuto je da se optimalne vrijednosne funkcije, ako je poznat model sustava, mogu riješiti kao sustav linearnih jednadžbi. Također je spomenuto kako taj pristup u praksi nije efektivan. Kao efektivno rješenje na diskretnim i konačnim skupovima stanja predstavljaju se tablične metode poticanog učenja. Tablične metode, dobile su ime po tome što se vrijednosti stanja, ili stanja i akcije pohranjuju u tablicu. Algoritmi u suštini funkcioniraju tako da iterativno, svakim prolaskom kroz stanje, procjenjuju vrijednosti stanja, ili stanja i akcije. Tako, kako broj posjeta teži u beskonačnost, procijene konvergiraju prema optimalnim vrijednostima. Nakon procijene vrijednosnih funkcija, politika se poboljšava te se postupak ponavlja. Tablične metode poticanog učenja mogu se podijeliti u tri glavne grupe:

- Metode dinamičkog programiranja
- Monte Carlo metode
- Metode temporalnih razlika

2.3.1. Dinamičko programiranje

Dinamičko programiranje je metoda koja se jako oslanja na teoriju postavljenu u poglavlju 2.2. Kao takva i dalje pati od računske složenosti no svejedno teorijski postavlja temelj poticanog učenja. Kako bi se dinamičko programiranje moglo primijeniti potreban je poznati model sustava. Ta karakteristika znatno ograničava primjenu dinamičkog programiranja na većem broju problema. Kako bi se taj problem zaobišao razvijene su metode, kao naprimjer Monte Carlo, koje mogu procijeniti vrijednosnu funkciju stanja bez poznavanja čitavog modela sustava ili poznavanjem tek ograničenog modela.

Dinamičko programiranje generalno se temelji na procesu iteracije politike i njegovim varijacijama. Proces iteracije politike sastoji se od dva već ranije spomenuta koraka: evaluacija politike i poboljšanje politike.

Proces evaluacije politike zapravo se sastoji od procijene vrijednosne funkcije. Konkretno za vrijednosnu funkciju stanja, ona se računa prema Bellmanovoj jednadžbi 2.6 Pravilo ažuriranja vrijednosti stanja pri nekoj iteraciji se može izraziti kao:

$$v_{k+1}(s) = \sum_{a \in A(s)} \pi(a|s) \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma v_k(s')] \tag{2.11}$$

Algoritam evaluacije politike iterira po svim stanjima te ažurira vrijednosti svakog stanja. Iteracija se ponavlja sve dok se maksimalna promjena vrijednosti u nekom koraku ne smanji ispod neke unaprijed definirane granice.

Nakon što je vrijednosna funkcija procijenjena, slijedi korak poboljšanja politike. Politika π se poboljšava tako da se konstruira politika π' koja se ponaša pohlepno s obzirom na procijenjene vrijednosti stanja i akcije. Drugim riječima, politika π' odabire akcije koje u stanju s izgledaju najbolje prema procijenjenim vrijednostima.

Formalno, politika se poboljšava na sljedeći način:

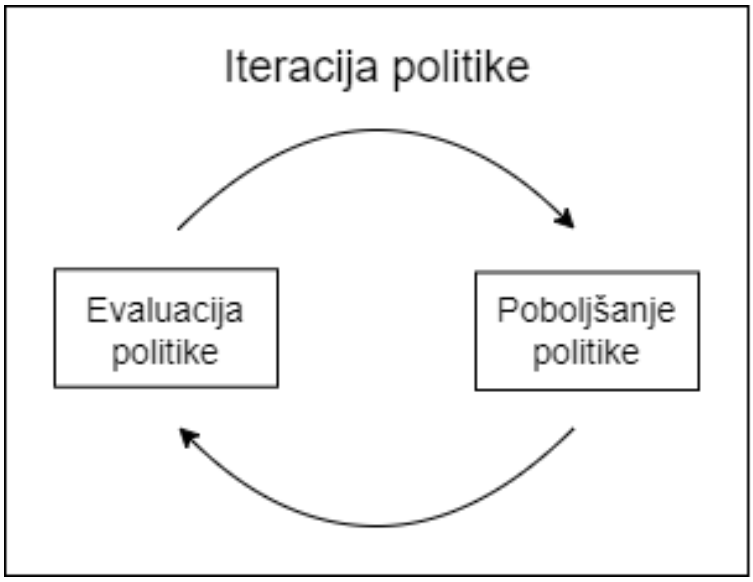
$$\pi'(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.12}$$

U slučaju da se više vrijednosti ispostave jednakima, akcije se između njih odabiru slučajno.

Ovako opisan proces iteracije politike prikazan je na slici 2.4.

Iteracija politike provodi se sve dok se politika ne stabilizira, odnosno dok se ne dogodi da je $\pi = \pi'$. U tom trenutku je poznato da je proces konvergirao te da je pronađena optimalna politika. Jedno poboljšanje ovog algoritma je iteracija vrijednosti. Iteracija vrijednosti objedinjuje evaluaciju i poboljšanje politike tako da se u svakom koraku vrijednost stanja ažurira prema sljedećem izrazu:

$$v_{k+1}(s) = \max_{a \in A(s)} \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma v_k(s')] \tag{2.13}$$



Slika 2.4. Proces iteracije politike

Može se pokazati da ovakvo pravilo ažuriranja također konvergira prema optimalnoj vrijednosnoj funkciji kada broj iteracija kroz sva stanja teži u beskonačnost. Nakon konvergencije vrijednosti, politika se jednostavno konstruira tako da pohlepno odabire akcije prema istom izrazu:

$$\pi'(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s' \in S, r \in R} p(s', r|s, a)[r + \gamma v_k(s')] \tag{2.14}$$

Budući da je broj koraka u procesu evaluacije politike smanjen na samo jedan, algoritam u ovom obliku postiže bolje performanse.

2.3.2. Monte Carlo metode

Iako dinamičko programiranje predstavlja sposoban način rješavanja problema poticanog učenja, činjenica da metoda zahtijeva model sustava kako bi mogla naučiti optimalnu politiku predstavlja znatno ograničenje. U ovom pregledu razmatrat će se samo epizodične primjene Monte Carlo metoda. Monte Carlo metode omogućuju rješavanje problema poticanog učenja bez poznavanja kompletnog modela sustava. Ovo omogućava činjenica da Monte Carlo metode ažuriraju vrijednosti tek nakon završetka epizode propagacijom nagrade unatrag svim stanjima koja su vodila do konačnog ishoda. Takvim načinom rada, Monte Carlo metode mogu raditi bez eksplicitnog poznavanja svih vjerojatnosti prijelaza za odabranu akciju.

Konkretno, Monte Carlo metode procjenjuju vrijednosne funkcije stanja tako da u svakom posjetu računaju aritmetičku sredinu povratnih vrijednosti zabilježenih u tom stanju. Dakle, nakon završetka epizode, za svako stanje s_{T-k} počevši od stanja s_T povratna vrijednost se propagira unatrag na sljedeći način: $G_t = R_{t+1} + \gamma G_{t+1}$, gdje se G u početku postavlja na nulu. Zatim se ta povratna vrijednost uračunava u aritmetičku sredinu. Aritmetička sredina se iterativno može ažurirati koristeći sljedeći izraz:

$$V_n(s) = \frac{V_{n-1}(s)(n-1) + (R_{t+1} + \gamma G_{t+1})}{n} = V_{n-1}(s) + \frac{1}{n}[(R_{t+1} + \gamma G_{t+1}) - V_{n-1}(s)] \quad (2.15)$$

gdje n predstavlja broj ažuriranja srednje vrijednosti, a t trenutak u epizodi. Može se primijetiti kako se u jednoj epizodi neko stanje može posjetiti više puta te se time aritmetička sredina vrijednosti ažurira više puta unutar epizode. Monte Carlo metode se dijele u Monte Carlo metode prvog posjeta i svakog posjeta prema tome dozvoljava li se više ažuriranja srednje vrijednosti u epizodi, ili samo ažuriranje pri prvom posjetu.

Ovaj pristup može se još dodatno unaprijediti. Trenutno, kako bi znali ažurirati politiku na temelju vrijednosti stanja, potrebno je imati djelomični model sustava koji nam govori u koja sljedeća stanja se može doći. Kako bi se skroz zaobišla ova potreba može se računati vrijednosna funkcija za uređene parove stanja i akcije. Ovako postavljena metoda u određenom stanju politiku može odrediti odabirom akcije s maksimalnom vrijednošću:

$$\pi(s) = argmax_a q(s, a) \quad (2.16)$$

Problem s ovakvim pristupom je taj da nakon što se jednom u nekom stanju dobije akcija s većom vrijednosti od svih ostalih akcija uvijek će se u tom stanju odabrati točno ta akcija. Ta činjenica je problematična jer je moguće da je neka druga akcija u tom stanju zapravo optimalna, no agent to neće saznati budući da će uvijek odabrati akciju za koju je prvu pronašao veću vrijednost. Mogući način rješavanja ovakvog problema je korištenje takozvane ϵ-pohlepne politike. ϵ-pohlepna politika ima svojstvo da akciji s najvećom vrijednosti dodijeli najveću vjerojatnost odabira dok ostatak raspodijeli jed-

nako kroz ostale akcije. Konkretno, svim ne pohlepnim akcijama dodijeli se vjerojatnost $\frac{\epsilon}{|A(s)|}$, dok se pohlepnoj akciji dodijeli vjerojatnost $1 - \epsilon + \frac{\epsilon}{|A(s)|}$. Na ovaj način politika je i dalje relativno pohlepna prema najviše procijenjenim vrijednostima stanja i akcija no ostavlja mogućnost odabira akcija koje se u tom trenutku možda ne čine optimalnima. Najbitnije, moguće je pokazati da nakon jednog kruga evaluacije politike π, konstrukcijom politike koja je ϵ-pohlepna s obzirom na vrijednosnu funkciju stanja i akcije dobiva se politika π' koja je nužno bolja ili jednako dobra kao politika π. Dokaz je moguće pronaći u [3].

Dakle, vrijednosti stanja i akcija se pod politikom π procjenjuju aritmetičkom sredinom temelju povratnih vrijednosti dobivenih posjetima stanju kroz broj epizoda, a paralelno politika se poboljšava konstrukcijom ϵ-pohlepne politike s obzirom na procijenjene vrijednosti stanja i akcija.

Ovako intuitivno formuliran proces spada pod učenje na politici. Problem kod ovakvog učenja, iako možda ne očit, je taj da se optimalna politika uči na zapravo neoptimalan način poduzimanjem akcija koje nisu nužno optimalne. Iako je ovakvo učenje efektivno, poboljšanje se može pronaći u učenju izvan ciljane politike, to jest učenju ciljane politike na temelju ponašajne politike. Ciljanom politikom se zapravo smatra optimalna politika, dok ponašajna politika predstavlja politiku kojom agent trenutno djeluje. Jedan nužni uvjet je da ponašajna politika svakoj akciji ciljane politike koja ima vjerojatnost odabira veću od nule također dodjeljuje neku pozitivnu vjerojatnost.

Realizacija učenja izvan ciljane politike ostvaruje se tako da se povratne vrijednosti prilikom posjete stanju i uračunavanja u aritmetičku sredinu skaliraju s omjerom vjerojatnosti odabira akcije pod ciljanom politikom i ponašajnom politikom: $\frac{\pi_c(a|s)}{\pi_p(a|s)}$

Dakle vrijednost stanja pri učenju izvan ciljane politike ažurira se prema sljedećem izrazu:

$$V(s) = \frac{1}{n} \sum_{i=1}^{n} \frac{\pi_c(A_i|S_i)}{\pi_p(A_i|S_i)} G_i \tag{2.17}$$

gdje n predstavlja broj posjeta stanju, a i konkretan posjet. Ovakav pristup zove se uzrokovanje po važnosti.

2.3.3. Metoda učenja temporalnih razlika

Opisujući Monte Carlo metode i dinamičko programiranje dobiven je uvid u neke njihove nedostatke. Problem Monte Carlo metoda je taj što moraju čekati kraj epizode kako bi mogle dobiti informaciju o procijeni povratne vrijednosti za svako stanje. Takvo ponašanje može znatno utjecati na brzinu učenja. S druge strane, metode dinamičkog programiranja, iako ne moraju čekati kraj epizode za procjenu povratne vrijednosti, su ograničene jer zahtijevaju potpun model sustava. Kao spoj najboljeg ovih dviju metoda i rješenje spomenutih nedostataka predstavlja se metoda učenja temporalnih razlika.

Metoda temporalnih razlika generira prijelaz iz jednog stanja u drugo te vrijednost prijašnjeg stanja procjenjuje na temelju procijene stanja u koje se dogodio prijelaz. Iz toga proizlazi sličnost Monte Carlo metodama jer nije potreban model sustava budući da agent uči iz generiranih prijelaza, dok sličnost metodama dinamičkog programiranja se sastoji od toga da se vrijednosti jednog stanja računaju na temelju vrijednosti drugih stanja. Vidljivo je da ovako postavljenom agentu nije potreban model sustava niti on mora čekati kraj epizode za procjenu vrijednosti.

Konkretno, pravilo ažuriranja vrijednosti stanja u metodi učenja temporalnih razlika može se zapisati na sljedeći način:

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.18}$$

gdje $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ predstavlja grešku procjene povratne vrijednosti, govori nam razliku između bolje procjene $R_{t+1} + \gamma V(S_{t+1})$ i prijašnje procjene $V(S_t)$. Ovako postavljena metoda učenja temporalnih razlika naziva se TD(0) metoda. TD(0) metoda je zapravo slučaj TD(n) metode gdje se za procjenu koristi samo jedno sljedeće stanje. Određivanje optimalne politike u metodi temporalnih razlika također se može ostvariti učenjem na politici i učenjem izvan ciljane politike.

Učenje na politici u metodi temporalnih razlika ostvaruje se na intuitivan način sličan već spomenutome. Opisan način određivanja vrijednosti za stanja vrijedi i za određivanje vrijednosti parova stanja i akcija. Jednadžba za ažuriranje vrijednosti stanja izmjenjuje se kako bi vrijedila za parove stanja i akcija na sljedeći način:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.19}$$

Kako bi se ostvarilo učenje na politici, paralelno uz ažuriranje vrijednosti konstruira se ϵ -pohlepna politika koja ϵ-pohlepno odabire akcije s obzirom na procijenjene vrijednosti stanja i akcija. Vrijedi da ovako formulirano pravilo sigurno konvergira prema optimalnoj politici i vrijednosnoj funkciji parova stanja i akcija. Budući da metoda ovisi o vrijednostima $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ dobila je ime Sarsa.

Učenje izvan ciljane politike u metodi temporalnih razlika zove se Q-učenje. U Q-učenju, koriste se dvije politike, ponašajna i ciljna. Podsjetimo se, ponašajna politika može biti bilo koja politika koja akcijama koje poduzima ciljana politika daje pozitivne vjerojatnosti odabira. Konkretno kao ponašanja politika može se odabrati ϵ-pohlepna politika. Kao ciljanu politiku korisno je odabrati politiku koja deterministički i pohlepno odabire akcije s obzirom na procijenjene vrijednosti stanja i akcija. Izraz za ažuriranje vrijednosti parova stanja i akcija u metodi Q-učenja može se zapisati na sljedeći način:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.20}$$

Dakle, u jednom koraku agent prema ϵ-pohlepnoj politici odabire akciju, a okolina mu odgovara sa nagradom i novim stanjem. Nakon toga agent ažurira vrijednost prethodnog stanja prema gore navedenom izrazu tako da se vrijednost ažurira prema maksimalnoj procijenjenoj vrijednosti sljedećeg stanja i akcije. Ciljana politika je nakon toga pohlepna prema procijenjenim vrijednostima stanja i akcija. Ovako postavljena metoda konvergira prema optimalnoj politici i vrijednostima stanja i akcija.

Budući da je metoda Q-učenja korištena prilikom implementacije agenta poticanog učenja u nastavku slijedi pseudokod metode Q-učenja.

Algorithm 1 Q-učenje

Inicijaliziraj $Q(s, a)$ proizvoljno, s $Q(teminalno, *) = 0$
Inicijaliziraj π proizvoljno
for epizoda **do** konačna epizoda
    Inicijaliziraj s
    while nije terminalno stanje **do**
        Odaberi akciju a prema ponašajnoj politici, npr. $(\epsilon - pohlepna)$
        Poduzmi akciju a
        Ažuriraj vrijednost $Q(s, a)$ prema izrazu 2.20
        $s \leftarrow s'$
    end while
end for

3. Pregled implementacije okoline i agenta

3.1. Okolina

Radi boljeg razumijevanja rada agenta poticanog učenja korisno je prvo upoznati se s okolinom za koju je agent razvijen. Kako bi se ostvarila mogućnost veće kontrole i podešavanja razvijena je vlastita okolina. Za izradu okoline korišten je programski jezik Python te biblioteka PyGame.

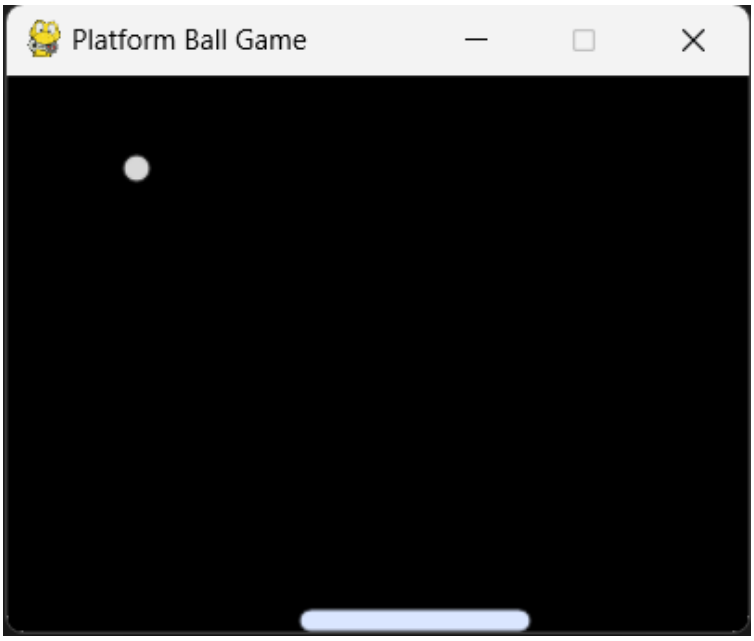
3.1.1. Opis okoline

Okolina je implementirana kao igra gdje postoji loptica koja se kreće u jednom od 4 dijagonalna smjera: gore-lijevo, gore-desno, dolje-lijevo, dolje-desno. Uz lopticu postoji platforma s kojom je cilj igre uhvatiti lopticu kako bi se ona odbila te spriječiti da loptica padne. Kolizijom loptice s platformom ili rubom okvira igre smjer se mijenja intuitivno tako da se odgovarajuće promijeni vertikalna ili horizontalna komponenta vektora smjera. Završetak igre definiran je slučajem kada loptica dostigne dno okvira igre. Što se tiče ponašanja okoline nakon što agent odabere akciju, ono je deterministično, za akciju a u nekom stanju s okolina uvijek vraća isto sljedeće stanje s' i nagradu r.

Okolina pruža dva sučelja kojima je moguće kontrolirati platformu: korisničko sučelje i programsko sučelje. Sučelje se može odabrati korištenjem odgovarajućih metoda: $run\_tick(action)$ i $reset()$ predstavljaju programsko sučelje, dok se korisničko sučelje koristi pozivom metode $run\_continuous()$ i korištenjem tipki lijeve i desne strelice. Također je moguće uključiti ili isključiti grafičko prikazivanje igre, to se ostvaruje pri instanciranju okoline putem argumenta konstruktora. Okolina se može instancirati na sljedeći način:

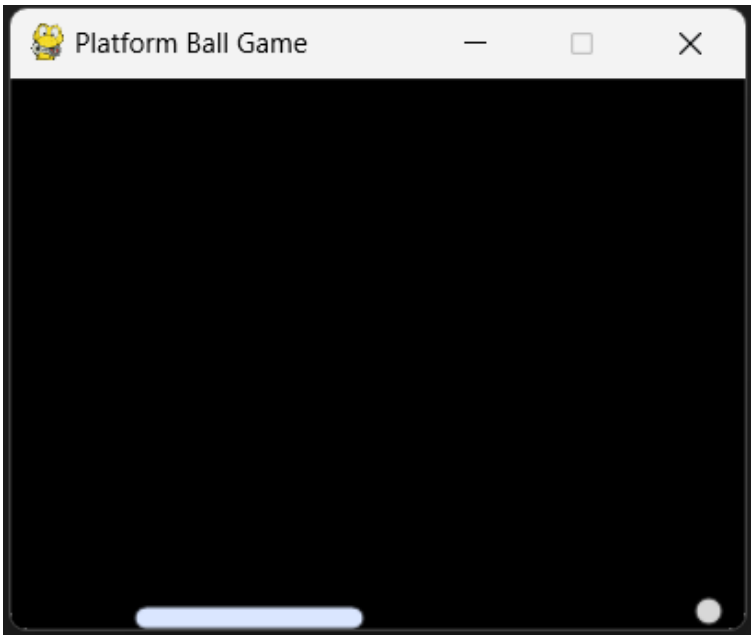
env = PlatformBallGame(PlatformBallGame.MODE.DISPLAY)

gdje argument *PlatformBallGame.MODE.DISPLAY* specificira grafički prikaz igre, uglavnom korišten za testiranje i podešavanje, dok *PlatformBallGame.MODE.TRAINING* specificira odvijanje igre bez grafičkog prikaza koje je pogodno za treniranje agenta budući da računalo ne mora trošiti resurse na prikaz igre. Grafički izgled okoline prikazan je slikom 3.1.



Slika 3.1. Prikaz izgleda okoline

Korisničko sučelje se odvija u kontinuiranoj petlji koja se izvodi sve dok se ne postigne uvjet kraja igre, uvjet kraja igre postiže se ulaskom okoline u terminalno stanje. Jedno od terminalnih stanja prikazano je slikom 3.2. U svakom krugu petlje okolina preuzima akciju od korisnika, u ovom slučaju pritisak tipke lijeve ili desne strelice, te pomiče platformu u odgovarajućem smjeru ako je korisnik pružio akciju, ili ju ostavlja na mjestu ako je akcija izostala. Programsko sučelje djeluje na sličan način, no umjesto da se igra interno odvija u kontinuiranoj petlji, sučelje pruža metodu *run_tick(action)* koja izvršava jedan korak igre s određenom akcijom te vraća informaciju o nagradi i stanju u kojemu se pronašla igra i metodu *reset()* koju je potrebno pozvati nakon kraja epizode kako bi se okolina postavila u početno stanje. Dužnost je na implementaciji agenta da se brine o kontinuiranom izvođenju igre tako da poziva metodu *run_tick* u petlji. Akcije koje okolina očekuje na programskom sučelju su -1, 0 i 1, gdje -1 i 1 odgovarajuće predstavljaju



Slika 3.2. Jedno od terminalnih stanja okoline

pomicanje ulijevo i udesno, a 0 označava da platforma ostaje na istome mjestu.

3.1.2. Stanja okoline

Budući da su fokus ovog rada tablične metode poticanog učenja, okolina je razvijena s diskretiziranim skupom stanja, pri čemu je ostavljena opcija za daljnje proširenje okoline s modeliranim kontinuiranim skupom stanja. Pojedino stanje sastoji se od 4 komponente:

- x koordinata lijevog kraja platforme
- x koordinata gornjeg, lijevog kuta loptice
- y koordinata gornjeg, lijevog kuta loptice
- smjer loptice

Kako bi se smanjio broj stanja, loptica i platform kreću se u koracima od 4 piksela. Pri inicijalizaciji okoline smjer loptice odabire se nasumično između gore-lijevo i gore-desno.

3.1.3. Model nagrade

Prilikom svakog koraka igre, okolina vraća nagradu agentu. Budući da je cilj igre spriječiti lopticu da padne odabran je model nagrade gdje se prilikom svake kolizije agenta i loptice agentu vraća pozitivna nagrada, proporcionalna udaljenosti loptice od središta platforme, dok se u jednom od terminalnih stanja igre, kada loptica dodirne dno okvira igre, agentu vraća negativna nagrada, ili kazna, čiji iznos ovisi o udaljenosti loptice od platforme u terminalnom stanju, inače nagrada iznosi 0. Što je platforma više udaljena od loptice u terminalnom stanju to je negativna nagrada veća.

Razlog odabira pozitivne nagrade pri uspješnom kontaktu s lopticom je intuitivan, agenta se želi nagraditi za uspješno izvedenu akciju kako bi se postiglo željeno ponašanje odbijanja loptice. Razlog odabira negativne nagrade u terminalnom stanju je proizašao iz problema koji se javio prilikom terminacije igre gdje su platforma i loptica na različitim stranama okvira igre. U tom slučaju problem proizlazi iz male vjerojatnosti da agent slučajnim pomicanjem platforme prijeđe na drugu stranu okvira igre gdje se nalazi loptica. Na prvi pogled kao rješenje se nameće povećanje faktora istraživanja ϵ, nedostatak takvog rješenja je dubina epizoda. Velika dubina epizoda uzrokuje da se agent, ako puno istražuje, često pronađe u nepoznatim stanjima u početku epizode gdje se ponaša nasumično, to uzrokuje da što epizoda dulje traje to je veća vjerojatnost da će se agent *"izgubiti"* ranije u epizodi gdje je zapravo već naučio uhvatiti lopticu. Rješenje koje se pokazalo učinkovitim je bilo dodavanje negativne nagrade u terminalnom stanju. Iznos negativne nagrade računa se po formuli:

$$kazna = -\left(\frac{|x_{platforme} - x_{loptice}|}{\text{širina okvira igre} - \text{širina platforme} - \text{širina loptice}}\right)^2 \tag{3.1}$$

Omjer udaljenosti i širine okvira igre služi za skaliranje negativne nagrade tako da je negativna nagrada između -1 i 0. Kvadrat daje veću važnost većim udaljenostima između platforme i loptice, a umanjuje negativnu nagradu kada igra završi u blizini platforme i loptice.

Razlog većeg iznosa nagrade pri pogađanju loptice što bliže sredini platforme je smanjenje rizika neuspješnog hvatanja loptice pri slučajnom istraživanju. Naime ako je raz-

dioba nagrade konstantna s obzirom na lokaciju kontakta platforme i loptice agent često nauči hvatati lopticu krajem platforme što kod slučajnog istraživanja povećava broj ranih neuspjeha što produljuje vrijeme učenja. Nagrada pri kontaktu platforme i loptice se računa na sljedeći način:

$$\text{nagrada} = c \times \left(1 - \frac{|x_{pc} - x_{lc}|}{w_p/2 + w_l/2}\right) \tag{3.2}$$

gdje c predstavlja neku konstantu koja skalira nagradu i omogućuje podešavanje učenja, x_{pc} predstavlja x koordinatu sredine platforme, x_{lc} x koordinatu sredine loptice, a w_p i w_l širinu platforme i loptice odgovarajuće. Izraz $|x_{pc} - x_{lc}|$ predstavlja udaljenost sredina platforme i loptice, tu udaljenost skaliramo na interval $[0, 1]$ dijeljenjem s polovicom zbroja širina platforme i loptice budući da je to maksimalna udaljenost sredina za koju se može dogoditi kontakt. Ta vrijednost iznosi 1 kada se sudar dogodi na rubu platforme, a 0 kada se sudar dogodi u sredini platforme. Budući da mi želimo nagradu obrnuto od toga, nagrada se oduzima od 1 te množi s pozitivnom konstantom c. Na ovaj način agent će prioritizirati kontakt s lopticom što bliže sredini platforme što je bio i cilj.

Načini na koje bi se problem mogao produbiti je pretvaranje skupa stanja u kontinuirani, beskonačni skup stanja. Konkretno, vrednovanjem smjera loptice prema vektoru koji može poprimiti bilo koji smjer, dodavanjem slučajnosti pri odbijanju loptice i time dodavanjem svojstva stohastičnosti okolini te dodavanjem prepreka loptici. Ovo bi uzrokovalo potrebu korištenja agenata koji se koriste aproksimacijskim metodama poticanog učenja.

3.2. Implementacija agenta

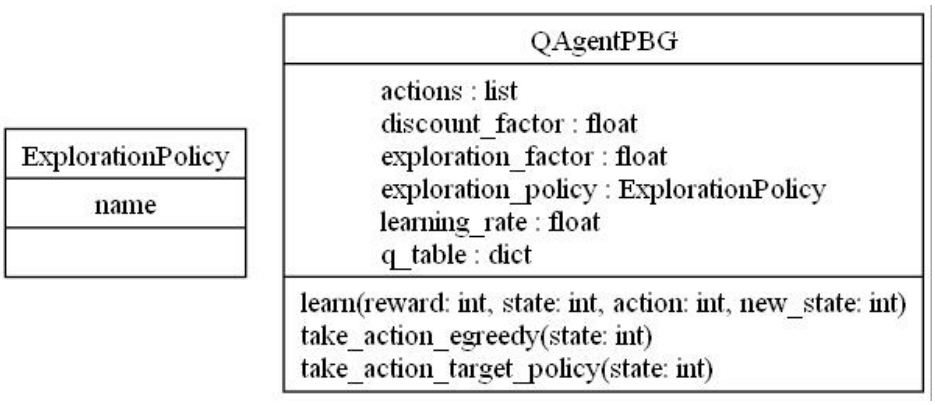
Kao što je već spomenuto, metoda poticanog učenja korištena za rješavanje problema platforme i loptice je Q-učenje. Q-učenje je odabrano zbog svoje efikasnosti i jednostavnosti. Detalji metode Q-učenja opisani su u poglavlju 2.3.3. Spomenuto je kako metoda Q-učenja spada u kategoriju metoda koje uče izvan ciljane politike, to jest koriste ponašajnu politiku kako bi se kretali po prostoru stanja, a vrijednosti parova stanja i akcija procjenjuju na temelju ciljne politike. U ovom slučaju korištena ponašajna politika je ϵ-pohlepna politika za koju se pri učenju, vrijednost faktora istraživanja postepeno sma-

njuje prema izrazu:

$$\epsilon = \epsilon - \frac{\epsilon}{\text{broj epizoda}} \tag{3.3}$$

ovakvo skaliranje faktora istraživanja ublažilo je problem terminacija u ranim dijelovima epizoda, gdje je agent već naučio hvatati lopticu, kada je dubina epizoda postala velika. Ciljana politika odabrana je kao politika pohlepna s obzirom na najbolje vrijednosti parova stanja i akcija u Q-tablici, ako više akcija u nekom stanju s ima istu vrijednost, akcija se odabire slučajno. Pseudokod metode Q-učenja već je ranije naveden te se može pronaći u poglavlju 2.3.3.

Agent je realiziran kao izdvojena klasa koja sadrži sve potrebne metode i strukture podataka potrebne za učenje i upravljanje agentom. Između okoline i agenta djeluje skripta koja kontrolira učenje. Ovakav dizajn je odabran kako bi se ostvarila odvojenost odgovornosti te kako bi se agent mogao serijalizirati i pohraniti za buduće korištenje nakon učenja. Izgled klase agenta prikazan je UML dijagramom 3.3.



Slika 3.3. UML dijagram klase agenta

Akcije su predstavljene kao lista s elementima 0, 1, 2. Vrijednosti predstavljaju sljedeće akcije:

- 0 - pomicanje platforme ulijevo
- 1 - ostavljanje platforme na mjestu
- 2 - pomicanje platforme udesno

Razlog odabira akcija u ovakvom obliku je pojednostavljenje indeksiranja u q-tablici. Prilikom prosljeđivanja okolini, akcije se mapiraju oduzimanjem 1 od dobivene akcije kako bi se dobila akcija iz skupa vrijednosti $-1, 0, 1$ koje se mogu proslijediti okolini. Klasa sadrži standardne parametre Q-učenja: stopu učenja, faktor istraživanja i faktor popusta.

Q-tablica je ostvarena kao Python struktura podataka dictionary. Kao ključevi koriste se vrijednosti sažetaka podatkovne strukture tuple u koji su pohranjene komponente stanja, ključevi se mogu ovako konstruirati budući da je svako stanje nužno jedinstveno. Kao vrijednosti u strukturu dictionary pohranjene su liste vrijednosti parova akcija i stanja, gdje vrijednost na i-tom mjestu u listi odgovara vrijednosti akcije *i* u nekom stanju *s*. Tablica je postavljena na ovaj način budući da omogućava pristup vrijednostima stanja i akcija u vremenu $O(1)$ što je bitno za brzinu učenja i rada agenta.

Agent nije pred-treniran na bilo koji način već su sve inicijalne vrijednosti parova akcija i stanja smatrane kao 0. Tablica se kreira dinamički prilikom prvog posjeta stanju kako bi agent zadržao fleksibilnost.

Enumeracija *ExplorationPolicy* služi za odabir politike istraživanja, trenutno je implementirana ϵ-pohlepna politika s podesivim faktorom istraživanja te je ostavljena mogućnost proširivanja funkcionalnosti implementacijom proizvoljne $\epsilon - mekane$ politike. $\epsilon - mekana$ politika je svaka politika π koja osigurava da svaka akcija ima pozitivnu vjerojatnost odabira, to jest $\pi(a|s) > 0, \forall a \in A(s), \forall s \in S$.

Agent pruža metode:

- *take_action_egreedy(state)* - s vjerojatnošću $1 - \epsilon$ vraća akciju prema ciljnoj politici, inače akciju odabire slučajno
- *take_action_target_policy(state)* - vraća akciju prema ciljanoj politici, konkretno u ovom slučaju akciju sa maksimalnom vrijednosti funkcije $Q(s, a)$, ako sve akcije imaju istu vrijednost, onda akciju odabire slučajno.
- *learn(state, action, new_state)* - ažurira vrijednost para stanja i akcije

Skripta koja je odgovorna za učenje i koordinaciju između okoline i agenta je imple-

mentirana kao petlja koja se izvodi za odabran broj epizoda, u svakoj epizodi okolina se postavlja u početno stanje, te se odvija standardna razmjena akcije, stanja i nagrade između okoline i agenta. Skripta također prati razne dijagnostičke podatke.

Konkretan pseudokod skripte izgleda ovako:

Algorithm 2 Kontrola učenja

```
Inicijaliziraj okolinu i agenta
N = broj epizoda učenja
broj epizoda = 0
while broj epizoda < N do
    Postavi okolinu u početno stanje
    broj epizoda = broj epizoda + 1
    Smanji faktor istraživanja
    while epizoda nije u terminalnom stanju do
        Dohvati akciju od agenta
        Proslijedi akciju okolini
        Proslijedi nagradu, stanje, akciju i novo stanje agentu na učenje
    end while
    Prati dijagnostiku
end while
Serijaliziraj i pohrani klasu agetna
```

4. Validacija i rezultati

4.1. Provedba učenja

Učenje agenta, algoritmom Q-učenja, igranju igre platforme i loptice provedeno je iterativnim postupkom. Uz podešavanje parametara učenja, paralelno je razvijan model nagrade okoline te su se rješavale razne propuštene greške pronađene u implementaciji okoline.

Glavni problem koji se javio pri učenju agenta bio je veliki kardinalitet prostora skupa stanja. Problem se ublažio smanjenjem kretanja loptice i platforme s koraka veličine jednog piksela na korake od 4 piksela. Veliki prostor stanja uzrokovao je i postepeno produbljivanje trajanja epizode što je uzrokovalo da agent slučajnim istraživanjem često završi u terminalnom stanju u ranijim dijelovima epizode gdje je već naučio lokaciju na kojoj treba pozicioniran kako bi uhvatio lopticu. Stoga je sve rjeđe dolazio do frontalnog dijela epizode gdje je tek trebao naučiti kako uhvatiti lopticu. Podešavanjem modela nagrade na način opisan u poglavlju 3.1.3. te podešavanjem parametara agenta ovaj problem je riješen te je agent naučio igrati igru. Konkretne postavke agenta koje su rezultirale uspješnim rješavanjem problema su:

- stopa učenja = 0.05
- faktor popusta = 0.9
- faktor istraživanja = 0.005
- broj epizoda učenja = 440 000

Iako je agent učio do 440 000. epizode, agent je naučio igrati igru tako da uvijek uhvati lopticu već oko 150 000. epizode. Agent je uspješno naučio igrati igru no vrijedi

napomenuti kako bi se znatna poboljšanja mogla pronaći ili u značajnijoj diskretizaciji stanja igre ili u korištenju agenata koji koriste aproksimacijske metode poticanog učenja.

4.2. Opis postupka validacije

Neovisno što je agent uspješno naučio igrati igru, formalno se provodi validacija agenta. Validacija se provodi puštanjem 3 različita agenta da igraju igru. Agenti su sljedeći:

- agent koji akcije odabire slučajno s jednakim vjerojatnostima
- vlastita implementacija agenta Q-učenja opisana u ovome radu
- gotova implementacija agenta poticanog učenja. Konkretno ...

Uspješnost i usporedba se provodi na temelju sljedećih kriterija:

1. Agent pokazuje sposobnost učenja ako u prosjeku uspijeva postići bolju nagradu od agenta koji akcije bira sasvim slučajno s jednakom vjerojatnošću
2. Agent a je bolji od agenta b ako agent a kroz 1000 epizoda, gdje svaka epizoda staje nakon 50 udaraca loptice o platformu ili dolaskom okoline u terminalno stanje, u prosjeku postigne veću kumulativnu nagradu.
3. Agent je naučio igrati igru ako kroz 1000 epizoda, u svakoj uspijeva postići broj udaraca loptice o platformu veći od 100

Ako implementirani agent uspješno zadovolji kriterij 1, te pokaže rezultate koji nisu značajno lošiji od gotovog algoritma implementirani agent je zadovoljiv. Ako implementirani agent zadovolji kriterij 3, agent je potpuno uspješan. Kriteriji su odabrani intuitivno kako bi pokazali razinu uspješnosti agenta pri igranju igre loptice i platforme.

4.3. Usporedba sa gotovim modelima

4.3.1. Opis korištenog gotovog agenta

Kao što je navedeno, usporedba se provodi između slučajnog agenta, implementiranog agenta i gotovog modela. Za gotovu implementaciju agenta odabrana je aproksimacijska metoda, specifično PPO (eng. *Proximal Policy Optimization*) tip algoritma iz biblioteke

Stable Baselines 3 [4]. PPO algoritmi spadaju u obitelj metoda poticanog učenja temeljenih na gradijentu, koje izmjenjuju između uzorkovanja podataka kroz interakciju s okolinom i optimiziranja "zamjenske" funkcije cilja korištenjem stohastičkog gradijentnog uspona [5]. Konkretno tip politike koji koristi PPO metoda biblioteke Stable Baselines 3 je ActorCritic politika. Metode su spomenute informativno te se neće detaljnije opisivati u ovom radu budući da je naglasak na tabličnim metodama i algoritmu Q-učenja.

Kako bi se razvijena okolina mogla koristiti u okviru biblioteke Stable Baselines 3, potrebno je implementirati sučelje kojime gotovi agent može koristiti okolinu igre platforme i loptice. Stable Baselines 3 zahtjeva da svaka vlastito implementirana okolina prati sučelje definirano u biblioteci OpenAI Gym [6]. Sučelje je implementirano kao klasa koja nasljeđuje klasu *gym.Env* te implementira metode *step*, *reset*, *render* te *close*. U spomenutim metodama, uz prilagodbu parametara, pozivaju se odgovarajuće metode okoline igre platforme i loptice.

4.3.2. Pretpostavke usporedbe

Prema prvom kriteriju ranije navedenih kriterija usporedbe 4.2., budući da su i implementirana metoda Q-učenja i gotova PPO metoda dokazano efektivne metode poticanog učenja, očekivano je pretpostaviti da će obje metode biti znatno uspješnije od kontrolnog agenta koji se ponaša slučajno.

Prema drugom kriteriju očekivano je da će PPO metoda biti uspješnija od metode Q-učenja te da će obje metode biti bolje od agenta koji se ponaša slučajno.

Prema trećem kriteriju razumno je vjerovati da će oba agenta uspješno naučiti igrati igru. Budući da je okolina relativno jednostavnog oblika, metoda Q-učenja može egzaktno naučiti igrati igru stoga je uspješnost vrlo vjerojatna. PPO metoda je također vrlo učinkovita metoda poticanog učenja, iako metoda ovisi o aproksimacijskoj funkciji što može uzrokovati varijacije u uspješnosti, očekuje se da će agent naučiti igrati igru.

4.3.3. Provedba usporedbe

Kako bi provjerili prvi kriterij, agenti su pušteni da igraju igru 1000 puta, u svakoj epizodi igra se zaustavlja nakon 10 udaraca loptice o platformu ili stizanjem okoline u terminalno

stanje. Prvo je pušten kontrolni agent te su zabilježeni rezultati. Nakon toga je pušten agent Q-učenja te PPO agent. Nakon oba pokusa zabilježeni su rezultati te su uspoređeni s kontrolnim agentom.

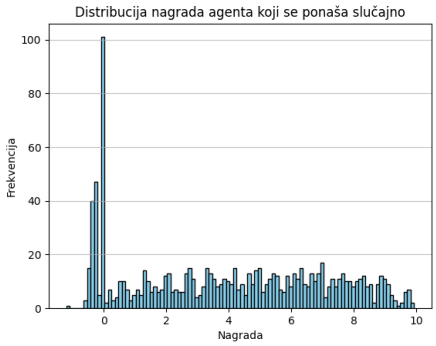
Provjera drugog kriterija je jasno opisana u samom kriteriju, agenti su pušteni da igraju prema kriteriju te je zatim izračunata prosječna nagrada.

Provjera trećeg kriterija također je jasno opisana kriterijem. Agent je pušten da igra igru, ako u svakoj epizodi uspješno dostigne 100 udaraca loptice o platformu naučio je igrati igru.

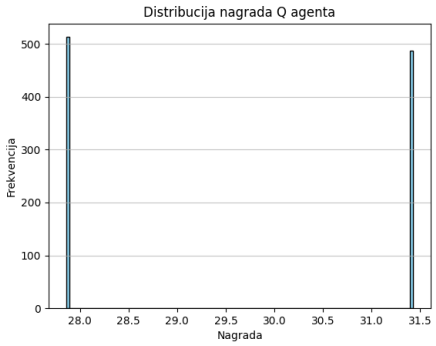
4.3.4. Rezultati

Provođenjem opisanog mjerenja, za provjeru prvog kriterija dobiveni su sljedeći rezultati.

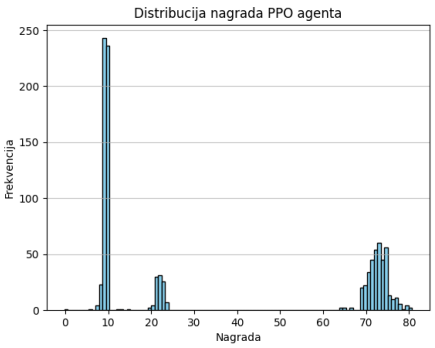
Pojedinačna distribucija nagrada svakog agenta prikazana je histogramima 4.1., 4.2. i 4.3.



Slika 4.1. Distribucija nagrada slučajnog agenta

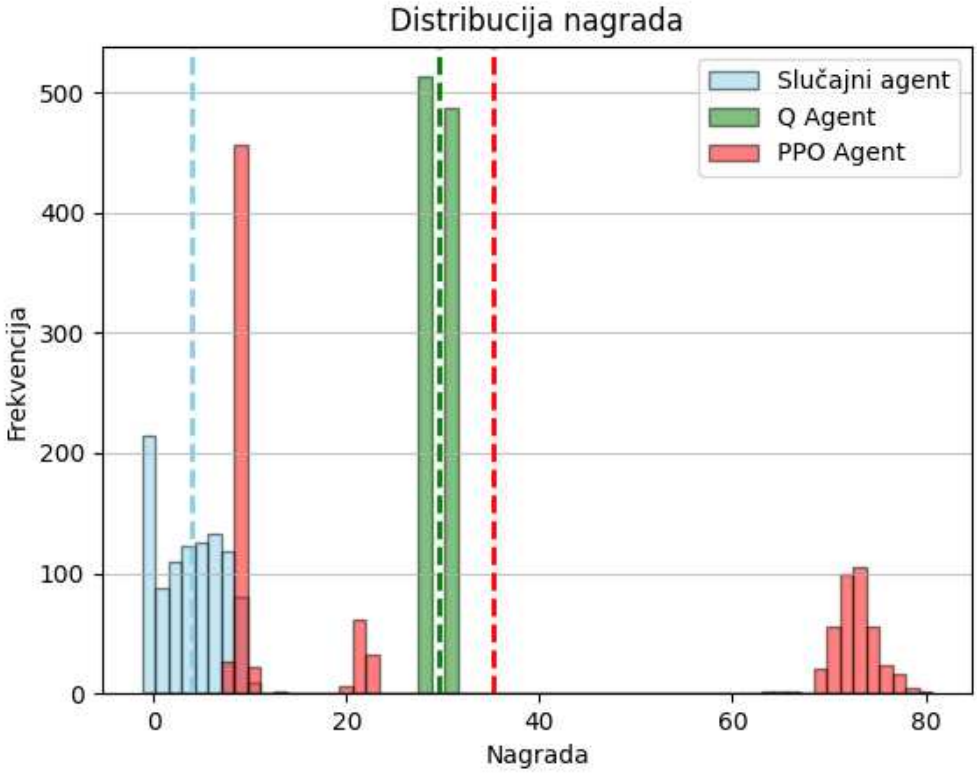


Slika 4.2. Distribucija nagrada agenta Q-učenja



Slika 4.3. Distribucija nagrada PPO agenta

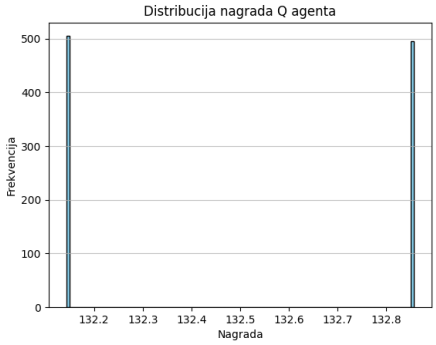
Distribucije, zajedno sa njihovim aritmetičkim sredinama međusobno su preklopljene i prikazane histogramom 4.4.



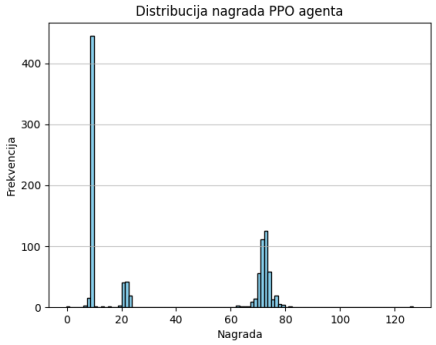
Slika 4.4. Uspoređene distribucije nagrada i aritmetičke sredine

Aritmetičke sredine prikazane su isprekidanim vertikalnim linijama u odgovarajućoj boji. Oba agenta, i agent Q-učenja i PPO agent, pokazuju rezultate u prosjeku bolje od slučajnog agenta. Prema tome oba agenta zadovoljavaju prvi kriterij i pokazuju sposobnost učenja što je i u skladu s očekivanom pretpostavkom.

Radi provjere drugog kriterija agenti Q učenja i PPO agent su pušteni da igraju igru 1000 puta do maksimalno 50 udaraca loptice o platformu. Pojedinačni rezultati prikazani su histogramima 4.5. i 4.6.

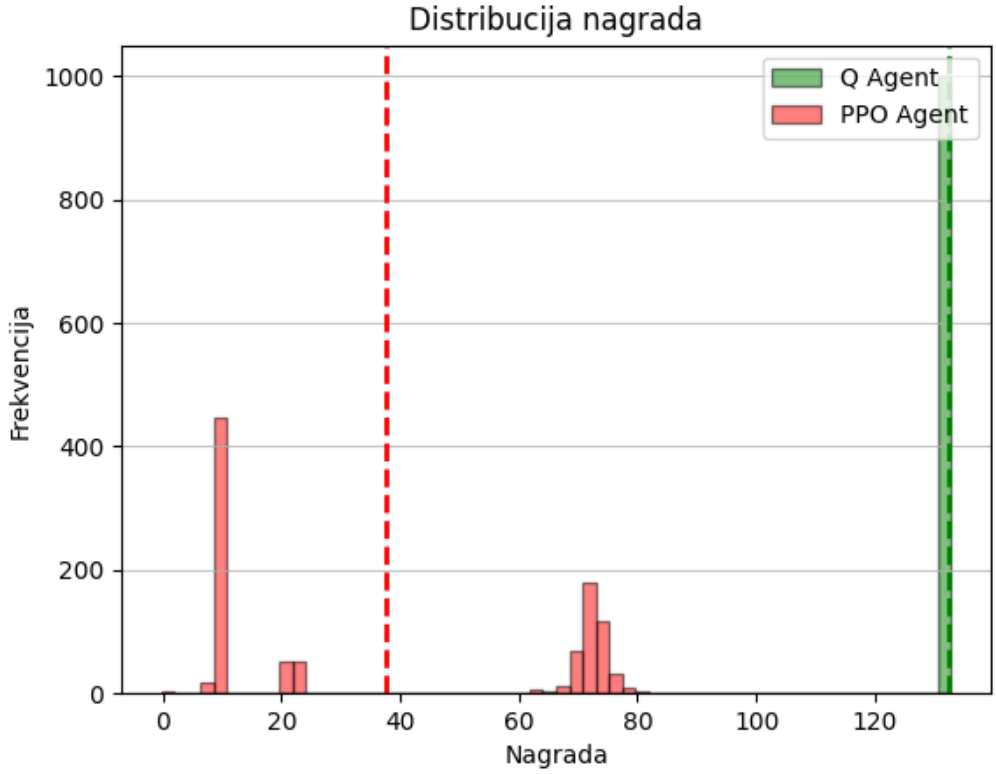


Slika 4.5. Distribucija nagrada agenta Q-učenja



Slika 4.6. Distribucija nagrada PPO agenta

Aritmetičke sredine s distribucijama prikazane su zajedno grafom 4.7.



Slika 4.7. Uspoređene distribucije nagrada i aritmetičke sredine

Zanimljivo je primijetiti kako Q agent zapravo uvjerljivo postiže konzistentne rezultate te je očigledno uspješniji od PPO agenta koji je postigao plato oko 10 pogodaka loptice. Ovo je suprotno očekivanoj pretpostavci budući da na prostorima stanja s velikim kardinalitetom aproksimacijske metode predstavljaju pouzdana rješenja. Analizirajući

rezultat može se zaključiti kako je to vjerojatno posljedica jednostavnosti okoline igre platforme i loptice gdje je agent Q učenja uspio egzaktno naučiti kako igrati igru. Zanimljivo je također primijetiti kako je PPO agent puno bolje naučio hvatati lopticu sredinom platforme što je agentu tabličnog Q učenja stvaralo veće probleme. U slučaju promjene parametara igranja igre razumno je pretpostaviti da bi PPO agent postigao bolje rezultate od Q agenta budući da ima sposobnost generalizirati na temelju svojstava stanja. U ovom slučaju performanse PPO agenta mogle bi se poboljšati ugođavanjem parametara metode.

Preostaje samo provjeriti treći kriterij, to jest pokazati da je Q agent naučio uspješno igrati igru. Agent je pušten da igra igru 1000 puta, ako svaki puta uspije doći do 100 udaraca loptice o platformu, agent je uspješno naučio igrati igru. Provodom mjerenja agent je u 1000 epizoda uspješno postigao 100 udaraca loptice o platformu 100% puta

5. Dodatne konfiguracije okoline

U trenutačnoj konfiguraciji okolina je relativno jednostavna, prostor stanja je konačan i diskretan. Skup akcija sastoji se od samo 3 akcije te su prijelazi za svaku akciju deterministički. Većina složenosti okoline proizlazi iz veličine skupa stanja.

Proširenje okoline može se ostvariti na razne načine. Prvi način je modeliranje prostora stanja kao kontinuirani i beskonačni prostor. To znači shvaćanje lokacija platforme i loptice kao kontinuirane vrijednosti. Ovdje problem predstavlja činjenica da se prilikom grafičkog prikazivanja okoline koriste diskretne vrijednosti lokacija platforme i loptice. Ovo znači da bi se interno trebao koristiti kontinuirani model prostora te bi se prilikom prikaza lokacije trebale diskretizirati. Dodatno, korištenje kontinuiranog prostora stanja omogućuje korištenje različitih iznosa veličine pomaka u jednom vremenskom koraku što omogućuje precizniju kontrolu brzina platforme i loptice. Zadatak agenta se može otežati postavljanjem brzine platforme na manji iznos od brzine loptice. Još jedna mogućnost u kontinuiranom prostoru stanja je predstavljanje smjera kretanja loptice jediničnim vektorom čija orijentacija nije ograničena. Na ovaj način bi se vrijednostima koordinata loptice pridodavale odgovarajuće vrijednosti komponenta vektora smjera pomnoženog nekom konstantom koja predstavlja brzinu loptice. Prilikom inicijalizacije igre mogao bi se koristiti slučajan početak gdje bi se komponente vektora smjera loptice odabrale iz uniformne razdiobe.

Drugi način proširenja okoline može se ostvariti korištenjem kontinuiranog prostora akcija. Korištenje kontinuiranog prostora akcija znatno otežava problem podržanog učenja te ograničuje primjenjive metode. U ovom slučaju prostor akcija bi se mogao ostvariti kao kontinuirani prostor tako da agent ne kontrolira sami pomak platforme već brzinu pomicanja. Na primjer, brzina platforme može se ograničiti na interval $-5, 5$ jedinica po koraku vremena te bi u tom slučaju agent mogao imati mogućnost odabira bilo koje

vrijednosti iz tog intervala.

Konačno, treći način na koji bi se okolina mogla proširiti je modeliranje stvarnog, fizikalnog ponašanja platforme i loptice. Ostvarivanje fizikalnog ponašanja obuhvaća modeliranje gravitacije koja utječe na kretanje loptice, zatim dodavanje gubitka energije koji loptica osjeća prilikom udaraca o platformu ili rub okvira igre i prilikom samog kretanja kroz prostor. Ovo bi modificiralo svojstvo epizodičnosti okoline budući da se gubi mogućnost beskonačnog trajanja epizode u slučaju da agent savršeno igra igru. Dodatno, potrebno bi bilo simulirati trenje koje se javlja prilikom kontakta loptice s platformom te stohastičnost prilikom odbijanja. Također važan faktor bila bi implementacija tromosti platforme prilikom promjene brzine. Složenost okoline mogla bi se kontrolirati uključivanjem ili isključivanjem pojedinih svojstava. Vrlo zanimljiv korak koji se otvara nakon implementacije simuliranog fizikalnog ponašanja je prelazak u stvarnu domenu konstrukcijom fizičke igre platforme i loptice. Agent bi se mogao pred-trenirati na simuliranoj okolini te zatim prenijeti te usavršiti na stvarnoj okolini.

6. Zaključak

Uz sve veću primjenu i razvoj umjetne inteligencije u većini grana ljudskog djelovanja poticano učenje uspostavlja se kao sposobna i pouzdana metoda rješavanja raznolikih problema sposobna pridonijeti razvoju društva. U ovom radu dan je pregled osnovnih koncepta i algoritama poticanog učenja, praktično je prikazana implementacija jednog od algoritama, specifično Q-učenja, na samostalno predstavljenom problemu igre platforme i loptice te je provedena usporedba s gotovim i kontrolnim agentom.

Pokazano je kako poticano učenje uspješno može rješavati probleme oblikovane kao interakcija agenta i okoline. Uspješnost metode Q-učenja dokazana je usporedbom s kontrolnim agentom gdje je agent uvjerljivo bio bolji od agenta koji se ponaša slučajno. Provedena je usporedba s gotovim modelom gdje je na ovom specifičnom problemu agent Q-učenja pokazao bolje rezultate od gotovog modela temeljenog na PPO metodi. Također pokazano je kako je agent Q-učenja uspješno naučio igrati igru bez ijednog neuspjeha.

Problematiku je moguće proširiti i nastaviti na razne načine. Jedan jednostavan smjer je primjena poticanog učenja na bilo koji drugi, potencijalno rješivi problem, ne nužno virtualni. Mogućnost je također proširiti igru platforme i loptice uvođenjem kontinuiranog prostora stanja, dodavanjem prepreka, stohastičnosti ili gravitacije i ponašanja zasnovanog na stvarnim fizikalnim zakonima. Uvođenje složenosti u okolinu povlači potrebu za korištenje aproksimacijskih metoda poticanog učenja.

Literatura

[1] Árpád Fehér, S. Aradi, i T. Bécsi, "Q-learning based reinforcement learning approach for lane keeping", u *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*. Budapest, Hungary: IEEE, November 2018. https://doi.org/10.1109/CINTI.2018.8928230

[2] B. D. Bašić, M. Čupić, i J. Šnajder, "Uvod u umjetnu inteligenciju: Podržano učenje", Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2021., creative Commons Imenovanje-Nekomercijalno-Bez prerada 3.0 v3.10. [Mrežno]. Adresa: https://www.fer.unizg.hr/_download/repository/predavanje.pdf

[3] R. S. Sutton i A. G. Barto, *Reinforcement Learning: An Introduction*, 2. izd. The MIT Press, 2018. [Mrežno]. Adresa: http://incompleteideas.net/book/the-book-2nd.html

[4] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, i N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations", *Journal of Machine Learning Research*, sv. 22, br. 268, str. 1–8, 2021. [Mrežno]. Adresa: http://jmlr.org/papers/v22/20-1364.html

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, i O. Klimov, "Proximal policy optimization algorithms", 2017.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, i W. Zaremba, "Openai gym", 2016.

Sažetak

Implementacija i validacija algoritma za poticano strojno učenje

Bruno Mikan

Poticano učenje postaje sve važnija grana umjetne inteligencije te se koristi za rješavanje sve složenijih problema. Matematička osnova poticanog učenja formalno je opisana konačnim Markovljevim procesima odlučivanja te smjerova za daljnji razvoj i istraživanja ima puno. U ovom radu dan je pregled osnove teorijske podloge poticanog učenja, implementirana je vlastita okolina s konačnim i diskretnim skupom stanja i akcija na kojoj se može trenirati agent poticanog učenja. Također, opisana je vlastita implementacija algoritma Q-učenja te je uspješno provedena usporedba s kontrolnim agentom i već gotovim agentom. Usporedbom je dokazana uspješnost implementacije algoritma Q-učenja na problemu igre platforme i loptice. Konačno, dan je pregled mogućih smjerova daljnjeg razvoja okoline.

Ključne riječi: Poticano učenje; Umjetna inteligencija; Markovljev proces odlučivanja; Q-učenje; Okolina; Agent

Abstract

Implementation and validation of reinforcement learning algorithms

Bruno Mikan

Reinforcement learning is becoming an increasingly important branch of artificial intelligence and is used to solve increasingly complex problems. The mathematical basis of reinforcement learning is formally described by the finite Markov decision process, and there are many directions for further development and research. In this work, an overview of the theoretical basis of supported learning is given. Also, an own environment with a finite and discrete state set and action set is implemented on which a reinforcement learning agent can be trained. Also, the implementation of the Q-learning algorithm was described, and a comparison was successfully carried out with a control agent and an already finished agent. The comparison proved the success of the implementation of the Q-learning algorithm on the platform and ball game problem. Finally, an overview of some possible directions for further development of the environment is given.

Keywords: Reinforcement learning; Artificial intelligence; Markov decision process; Q-learning; Environment; Agent