VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Baigiamasis bakalauro darbas

Q algoritmų pritaikymas pastiprintame agentų mokyme

(Applications of Q Algorithms in Reinforcement Learning for Artificial Agents)

Atliko: 4 kurso 2 grupės studentas
Ilja Jurčenko (parašas)

Darbo vadovas:
partn. doc. Rokas Masiulis (parašas)

Recenzentas:
asist. dr. Linas Litvinas (parašas)

Vilnius
2019

Turinys

1 Įvadas

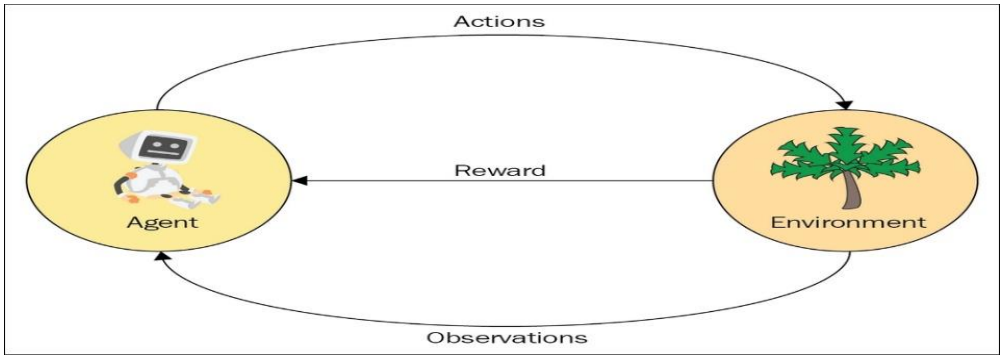
Idėja, kad mes mokomės sąveikaujant su mūsų aplinka, tikriausiai pirmiausiai atsiranda, kai galvojame apie mokymosi pagrindą. Kai kūdikis žaidžia, moja rankomis ar žiūri, jis neturi aiškaus mokytojo, bet jis turi tiesioginį ryšį su savo aplinka. Naudojant šį ryšį gaunama daug informacijos apie veiksmo priežastį ir pasekmes ir apie tai, ką daryti siekiant tikslų. Visą gyvenimą žmogus mokosi iš sąveikos su erdve ir tai yra pagrindinis žinių apie mūsų aplinką ir save šaltinis. Nesvarbu, ar mokomės vairuoti automobilį ar parengti pokalbį, mes puikiai suprantame, kaip mūsų aplinka reaguoja į tai, ką mes darome, ir siekiame koreguoti savo veiksmus priklausomai nuo atgalinio aplinkos ryšio ir tai leidžia keisti ne tik mūsų supratimą, bet ir mūsų pasaulį. Mokymasis iš sąveikos yra pagrindinė idėja, kuria grindžiamos beveik visos mokymosi ir intelekto teorijos. Vienas iš tokių teorijų yra pastiprintas kompiuterinis mokymas, kuris yra nagrinėjamas šiame darbe. Tema yra aktuali tuo, kad pastiprintas mokymas yra taikomas įvairiose srityse: robotikoje, medicinoje, chemijoje, ekonomikoje, inžinerijoje.

Bakalaurinio darbo tikslas skirtingose aplinkose panaudoti pagrindinę pastiprinto mokymo idėją ir algoritmus dirbtinio intelekto kūrime, bei jei įmanoma surasti optimaliausią sprendimą dirbtinio intelekto pritaikymo atveju. Tikslui pasiekti bakalaurinio darbo metu yra iškelti uždaviniai:

- Atlikti pastiprinto mokymo pagrindų apžvalgą ir analizę.
- Atlikti pagrindinių pastiprinto mokymo algoritmų analizę.
- Praktiškai realizuoti pastiprinto mokymo algoritmus skirtingose aplinkose.
- Sukurti dirbtinį intelektą ir atlikti įgyvendintos sistemos analizę, testavimą ir suformuluoti hipotezes dėl optimalaus dirbtinio intelekto konkrečioje aplinkoje.

2 Pastiprintas mokymas

Pastiprintas mokymas (angl. *reinforcement learning*) – tai kompiuterinio mokymo sritis (*angl. machine learning*), kurioje mokymas vyksta sąveikaujant su aplinka, kur agentai nėra mokomi kokių veiksmų reikia imtis [Rnd16]. Vietoj to, agentas mokosi iš savo pasekmių. Agentas turi išmokti elgtis dinamiškoje aplinkoje per bandymų ir klaidų sąveiką. [Klm96]. Įprastame modelyje aplinkos interpretatorius siunčia agentui duomenis apie gautas būsenas ir atpildas, o agentas pasirenka atitinkantį veiksmą, kuris įtakoja aplinkai [Sb16]. Pastiprintas mokymas susideda iš subjektų: agento, aplinkos (angl. *agent, environment*) ir jų ryšių: veiksmų (angl. *actions*), atpildų (angl. *reward*) ir stebėjimų (angl. *observations*) (pav. 1).



pav. 1 Pastiprinto mokymo subjektai ir ryšiai [Lap18]

2.1 Atpildas

Pastiprintame mokyme atpildas yra vienas iš pagrindinių ryšių tarp aplinkos ir agento. Tai yra tiesiog skaliarinė reikšmė, kuri gali būti teigiama arba neigiama, didelė arba maža. Atpildas žymimas *r*. Pagrindinė atpildo reikšmė pastiprintame mokyme yra pasakyti kaip agentas gerai arba blogai elgėsi. Mes nenustatome, kaip dažnai agentas gauna atpildą, bet praktikoje įprastai agentas gauna atpildą po kiekvienos sąveikos su aplinką. Iš esmės, terminas pastiprintas (angl. *reinforcement*) kyla iš to, kad gautas atpildas skatina agentą sustiprinti arba patobulinti savo elgesį į teigiamą arba neigiamą pusę [Lap18]. Atpildas yra lokalus, jis atvaizduoja paskutinio veiksmo rezultatą, bet ne visas agento iki šiol pasiektas sėkmes. Aišku, kad jeigu gausim didelį atpildą iš veiksmo, tai ne reiškia, kad po to nebus didelių blogų pasekmių dėl ankstesnių sprendimų. Pavyzdys būtų banko apiplėšimas: jeigu tai pavyks, tai agentas gaus didelį atpildą, bet po sėkmingo veiksmo atsiranda daug pasekmių [Lap18]. Agentas atliekant veiksmus bando pasiekti didžiausią sukauptą atpildą [Sb16]. Keletą pavyzdžių:

- Finansinė prekyba: pelno suma yra didžiausias sukauptas atpildas prekybininkui perkant ir parduodant atsargas [Lap18]
- Dopamino sistema smegenyse: smegenyse yra dalis (limbinė sistema), kuri gamina dopaminą kiekvieną kartą, kai jai reikia siųsti teigimą signalą. Didesnė dopamino koncentracija sukelia malonumą, o tai sustiprina ir skatina veiklą [Lap18].
- Kompiuteriniai žaidimai: žaidėjas žaidimo metu gauna atpildą, dažniausiai tai yra surinkti taškai. Šiame pavyzdyje atpildas jau yra įprastai sukaupiamas [Lap18].

Kaip matome iš pavyzdžių, atpildo sąvoka bendrai atvaizduoja agento veiklos rezultatą, tokią schemą galima rasti ir įdėti į daugelį praktinių problemų.

2.2 Dirbtinio intelekto agentas

Dirbtinio intelekto agentas (angl. *agent*) yra subjektas kuris sąveikauja su aplinka, atlieka tam tikrus veiksmus, analizuoja stebėjimus ir gauna atpildas, bei keičia savo būsenas [Sb16]. Kiekviename bendravimo etape su aplinka, agentas pasirenka veiksmą a, priklausomai nuo

būsenos s ir gauto atpildo r [Klm96]. Agentas turi pasirinkti veiksmus, didinančius ilgalaikę atpildų sumą. Nesunku pastebėti, kad agentą galima pavaizduoti kaip funkciją su argumentais, o funkcijos reikšmė yra agento veiksmo rezultatas aplinkai [Klm96].

2.3 Aplinka ir jos tipai

Aplinka (angl. *environment*) yra viskas kas egzistuoja už agento ribų ir su kuo agentas gali sąveikauti. Agento ryšis su aplinka riboja atpildai (kurie yra gaunami iš aplinkos), veiksmai (agentas atlieka veiksmus ir keičia aplinką) ir stebėjimai (tam tikra papildoma informacija apie aplinką neįskaitant atpildų) [Lap18]. Egzistuoja keletą aplinkos tipų:

- Determinuota aplinka (angl. *deterministic environment*) – aplinka kur mes, tiksliai žinome visų veiksmų atomazgą. Pavyzdžiui, šachmatų žaidime mes žinome, kiekvieno ėjimo rezultatą [Rnd16].
- Stochastinė aplinka (angl. *stochastic environment*) – aplinka yra stochastinė, kai negalime nustatyti rezultato pagal dabartinę būseną. Pavyzdžiui, mes niekada nežinome tiksliai kokį skaičių gausime kai mesim kauliuką [Rnd16].
- Pilnai permatoma aplinka (angl. *fully observable environment*) – kada agentas nepriklausomai nuo savo būsenos gali apibrėžti visą sistemos būseną. Pavyzdžiui, šachmatų žaidime, sistemos būsena (pozicija visų figūrų ant lentos) yra prieinama visiems žaidėjams [Rnd16].
- Dalinai permatoma aplinka (angl. *partially observable environment*) – kada agentui yra dalinai prieinama visos sistemos būseną [Rnd16]. Pavyzdžiui, žaidžiant pokerį neįmanoma tiksliai pasakyti oponento kortų kombinaciją. [Asb08].
- Vienos ir kelių agentų aplinka (angl. *single and multi-agent environment*) – aplinka vadiname kelių agentų aplinka kai skirtingi agentai, veikiantys visiškai skirtingose aplinkose tarpusavyje bendrauja [Rnd16]. Daugialypė aplinka bus dažniausiai stochastinė [Asb08]. Pavyzdys tokios aplinkos yra realaus laiko strateginis žaidimas, kur egzistuoja žemo lygio agentas, kuris kontroliuoja vienetus ir aukšto, strateginio lygio agentas, kuris kontroliuoja žaidimo strategiją [Bur03].
- Epizodinė ir ne epizodinė aplinka (angl. *episodic and non-episodic environment*). Epizodinė aplinka taip pat vadinama nenuoseklia aplinka. Epizodinėje aplinkoje agento dabartinis veiksmas neturės įtakos būsimam veiksmui, o ne epizodinėje aplinkoje atvirkščiai [Rnd16]. Tai reiškia, kad agentas atlieka nepriklausomas užduotis epizodinėje aplinkoje, o ne epizodinėje aplinkoje visi agento veiksmai yra susiję [Rnd16].

2.4 Veiksmai

Veiksmai pasako, kaip agentas gali keisti aplinką arba ką jis gali padaryti aplinkoje, priklausomai nuo aplinkos taisyklių ir apribojimų [Lap18]. Pastiprintame mokyme veiksmus galima padalinti į du tipus: diskretus veiksmas (angl. *discrete*) ir nuolatinis veiksmas (angl. *continuous*) [Sb16]. Diskretūs veiksmai - veiksmai iš baigtinės aibės nepriklausomų veiksmų, kuriuos agentas gali pasirinkti atlikti. Pavyzdžiui, eiti į kairę arba į dešinę. Nuolatiniai veiksmai turi papildomą reikšmę, pvz., automobilio veiksmas „vairuoti“ turi kampą ir kryptį [Lap18]. Skirtingas vairavimo kampas ir kryptis duoda skirtingus rezultatus.

2.5 Stebėjimai

Aplinkos stebėjimas yra antras agento informacijos šaltinis, kur pirmas yra atpildas [Lap18]. Aplinkos stebėjimas duoda agentui informaciją apie aplinką ir pasako sistemos būseną. Pavyzdžiui, šachmatų žaidime aplinka yra lenta ir oponentas, kuris susideda iš įgūdžių, taktikų, o

stebėjimo rezultatas yra figūrų padėtis [Sb16]. Gali būti taip, kad agentas gauna informaciją apie atpildą per aplinkos stebėjimą, pavyzdžiui toks atvejis yra įmanomas, kai agentas mokosi žaisti žaidimą naudojant žaidimo ekrano kopiją (angl. *screenshot*), kur yra atvaizduojami agento žaidimo taškai [Lap18].

2.6 Aplinkos modelis

Aplinkos modelis yra agento supratimas apie aplinką. Pastiprintas mokymas gali būti dviejų rūšių – mokymas pagal modelį (angl. *model-based learning*) arba be jo (angl. *model-free learning*). Mokyme pagal modelį agentas išnaudoja anksčiau įgytą informaciją tikslui pasiekti (kitaip tariant turi informaciją apie aplinką prieš mokymą), o mokyme be modelio agentas tikslui pasiekti tiesiog sukaupia informaciją apie aplinką mokymo metu. Vienas iš pavyzdžių būtų nukeliauti iš taško A į tašką B. Mokyme pagal modelį agentas tikslui pasiekti (nukeliauti iš taško A į tašką B) paprasčiausiai naudos anksčiau įgytą informaciją – žemėlapį ir bandys pasiekti tikslą kuo greičiau. Pastiprintas mokymas be modelio nenaudos ankstesnės informacijos apie aplinką, tiesiog bandys visus skirtingus maršrutus ir pasirinks greičiausią [Rnd16].

2.7 Pastiprinto mokymo išskirtinės savybės

Kompiuterinis mokymas (angl. *machine learning*) susideda iš prižiūrimojo mokymo (angl. *supervised learning*), neprižiūrimojo mokymo (angl. *unsupervised learning*) ir pastiprinto mokymo. Prižiūrimajame mokyme agentas mokosi iš aibės duomenų, kur duomenys yra sužymėti (angl. *labeled*). Tikslas tokio modelio apmokyti agentą taip, kad jį būtų galima tinkamai pritaikyti prie nematytų duomenų. Iš esmės, prižiūrimajame mokyme egzistuoja subjektas kuris koreguoja agentą ir padeda agentui pasiekti galutinį tikslą. Prižiūrimajame mokyme yra sprendžiamos problemos kaip teksto, ženklų, veidų, objektų atpažinimas, prognozavimas (regresija) ir kitos. Iš esmės visos prižiūrimojo mokymo sprendžiamos problemos turi vieną pagrindinę idėją: mokymas vyksta iš aibės žinomų sužymėtų duomenų gautų iš prižiūrėtojo (angl. *supervisor*). Iš kitos pusės egzistuoja kitas mokymo būdas, kuris neturi prižiūrėtojo – neprižiūrimasis mokymas. Šio mokymo pagrindinė idėja yra išmokti atpažinti paslėptą struktūrą iš aibės duomenų. Kitaip sakant, mes pateikiame duomenis be papildomos informacijos agentui kaip įvestį ir bandome konfigūruoti agentą taip, kad jis sugebėtų atpažinti duomenų aibėje paslėptą modelį, šabloną. Pastiprintas mokymas yra kažkur tarp šių mokymo būdų. Agentas pastiprintame mokyme mokosi iš atpildų maksimizavimo ir naudoja mokymosi tikslams nusistovėjusius metodus (neuroniniai tinklai, funkcijos aproksimacija, dinaminis programavimas, gradientinis nusileidimas), bet pastiprintame mokyme metodai yra pritaikomi šiek tiek kitaip [Lap18].

2.8 Pagrindinė agento apmokymo idėja

Pastiprinimas mokymas leidžia mašinai ar programinės įrangos agentui išmokti elgtis erdvėje, naudojant grįžtamąjį ryšį iš aplinkos. Tai yra pagrindinė apmokymo idėja. Mokymo metu gautas elgesys gali būti įsisavintas vieną kartą arba dinamiškai keistis. Jei sistema yra teisingai sumodeliuojama kai kurie pastiprinto mokymo algoritmai gali rasti optimaliausią strategiją - idealus elgesys, kuris maksimaliai padidina atpildą. Šios automatizuotos sistemos panaudojimas reiškia, kad žmogiškojo eksperto, kuris žino apie taikymo sritį, jau nebereikia. Agentas gali pats išmokti atlikti reikalingus veiksmus, jei bus sukurtas korektiškas atpildo mechanizmas. Yra daug skirtingų algoritmų, kurie sprendžia šią problemą. Uždavinyje agentas turėtų pasirinkti geriausią veiksmą dabartinėje būsenoje. Pakartojant tokį mokymo žingsnį problema susiveda į MDP (angl. *Markov decision process*) [Sb16].

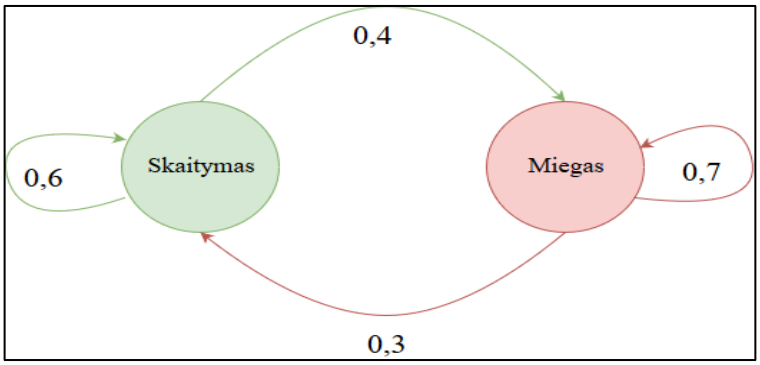
3 Markovo procesas

Markovo procesas, yra apibrėžimas kaip 2 elementų rinkinys $(\boldsymbol{S}, \boldsymbol{T})$ [Lap18], kur

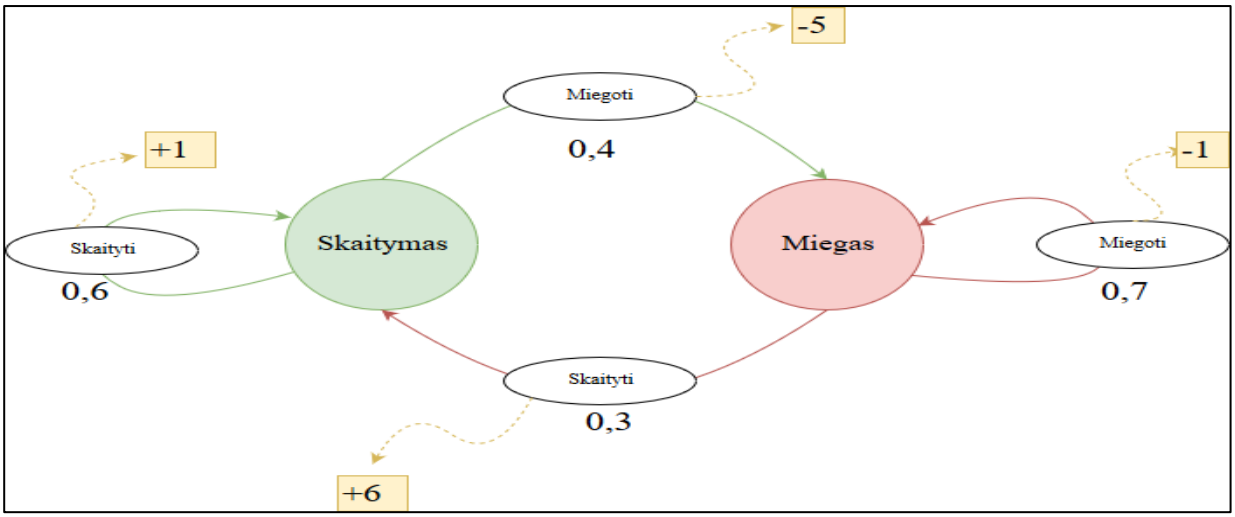
- $\boldsymbol{S}$ baigtinė aibė būsenų

- $\boldsymbol{T}$ perėjimo matrica, kurios reikšmės yra tikimybės pereiti iš būsenos s į būseną s'

Markovo grandinė (angl. *Markov chain*) tai yra Markovo proceso rūšis, turinti diskrečią būsenų erdvę arba diskretų indeksų rinkinį (dažniausiai indeksas pasako laiką) [Asm03]. Markovo procesas arba grandinė yra stochastinis procesas, kuris atitinka Markovo savybę [Roz82]. Markovo savybė pasako, kad būsimos sistemos dinamika ir pakeitimai iš bet kurios būsenos turi priklausyti tik nuo šios būsenos, (kartais apibūdinimas kaip „atminties nebuvimas“). [Roz82]. Kitaip tariant, Markovo savybė reikalauja, kad sistemos būsenos būtų viena nuo kitos atskiriamos ir unikalios, bei būsima būsena turi nepriklausyti nuo praeitų. [Lap18]. Markovo procese įmanoma tik tai stebėti sistemos perėjimą iš vienos būsenos į kitą ir tokios sistemos stebėjimo rezultatas yra būsenų seka, vadinama būsenų istoriją (angl. *state history*) [Lap18]. Pavyzdžiui, turime tokią sistemą, kur agentas gali būti būsenoje miegoti (M) arba skaityti (S) ir turime matricą T su perėjimo tikimybėmis (1 lentelė). Agentas gali skaityti knygą ir likti šioje būsenoje su tikimybe 60% arba pereiti į būseną „miegas“ su tikimybe 40% ir analogiškai būsenoje „miegas“ agentas gali likti šioje būsenoje su tikimybe 70% arba pereiti į būseną „skaitymas“ su tikimybe 30% (pav. 2). Nesunku pastebėti, kad būsenos istorija bus [*skaitymas, miegas, miegas, skaitymas* ...].



pav. 2 Markovo procesas pavaizduotas kaip baigtinis automatas



pav. 3 Markovo sprendimo procesas, kur geltonai yra pažymėti atpildai

Pastiprintam mokymui reikalingas matematinis modelis, kur galima sąveikauti su aplinka ir gauti atgalinį ryšį. Patobulintas Markovo procesas pastiprintame mokyme vadinasi Markovo sprendimo procesas.

Markovo sprendimo procesas yra apibrėžimas kaip 4 elementų rinkinys $(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}_a, \boldsymbol{R}_a)$ [Sb16].

- $\boldsymbol{S}$ baigtinė aibė būsenų.
- $\boldsymbol{A}$ baigtinė aibė veiksmų (analogiškai $\boldsymbol{A}_s$ baigtinė aibė galimų veiksmų būsenoje s).

- $\boldsymbol{P_a(s, s')} = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ tai yra tikimybė pereiti į būseną s' laiko momentu $t + 1$, atliekant veiksmą a būsenoje s laiko momentu t.
- $\boldsymbol{R_a(s, s')}$ tai yra atpildas, gautas pereinant iš būsenos s į būseną s' dėl veiksmo a

Markovo sprendimo procese atsiranda veiksmas ir atpildas, kuris yra gaunamas, kada veiksmas įtakoja aplinką. Tada atsiranda dvi naujos esybės, kaip atpildų ir būsenų perėjimo matricos. Mūsų pavyzdyje (pav. 2) agentas gali miegoti arba skaityti, iš to kyla klausimas koks yra pastiprinto mokymo tikslas. Tegul tikslas bus skatinti agentą kuo daugiau skaityti ir kuo mažiau miegoti. Be to, kiekvieną kartą agentas pasirenka veiksmą iš aibės veiksmų $\boldsymbol{A}$ [*skaityti*, *miegoti*]. Priklausomai nuo pasirinkto veiksmo ir tikimybės agentas pereis iš vienos būsenos į kitą arba liks senoje būsenoje ir gaus atpildą iš aplinkos. Kada agentas skaito jis gauna teigiamą atpildą +1, nes mes norime, kad agentas daugiau skaitytų. Kai agentas pereina iš būsenos „skaitymas“ į būseną „miegas“ jis yra baudžiamas -5, o kai agentas sustoja miegoti ir pradeda skaityti jis yra skatinamas +6 (pav. 3).

	Miegoti	Skaityti
Miegoti	0,7	0,3
Skaityti	0,4	0,6

lentelė 1 Markovo proceso matrica T, kur agentas turi dvi būsenas

3.1 Markovo atpildas

Markovo procese galima stebėti perėjimą iš vienos būsenos į kitą, o patobulintame modelyje atsiranda atpildas kiekvienam perėjimui, kuris yra kaupiamas. Iš to seka, kad galutinis atpildas G bus lygus:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad kur\ 0 \leq \gamma \leq 1$$

form.1 Galutinio atpildo formulė [Sb16]

Kiekviename etape yra apskaičiuojama suma visų praeitų atpildų, bet tolesni atpildai padauginami iš koeficiento γ, kuris yra keliamas laipsniui k (form. 1). Laipsnis pasako, kaip toli mes esame nuo pradinio taško laiko momentu t arba kiek žingsnių padarėme nuo pradinio taško [Rnd16]. Koeficientas γ pasako, kaip daug agentas žiūri į ateitį. Jeigu $\gamma = 1$, tai galutinis atpildas bus lygus visų vėlesnių atpildų sumai ir agentas pastiprintame mokyme įskaitys visus atpildus, kai jam reikės pasirinkti veiksmą. Jeigu $\gamma = 0$, mūsų galutinis atpildas apibūdins tiktai einamąjį atpildą, be jokių būsimų būsenų ir tai apibrėžia agento trumparegiškumą [Lap18]. Praktiškai koeficiento reikšmė yra nustatoma $0{,}99 > \gamma > 0{,}9$ [Lap18]. Tada agentas žiūrės į būsimus atpildus, bet ne taip stipriai į ateitį. Apibendrinant, γ pasako kiek agentas išsaugo tolimesnės informacijos, kai reikšmė artėja iki 1. Toks parametras, kaip γ leidžia modifikuoti svarbumą artimiausių ir tolimiausių tikslų. Kai kuriais atvejais reikia atsižvelgti labiau į dabartinius atpildus, o kitais atvirkščiai. Pavyzdžiui, šachmatų žaidime tikslas yra nugalėti oponento karalių. Jei mes suteikiame prioritetą dabartinėms, artimiausiems tikslams, tai agentas visada kers priešininko figūras ir tai ne reiškia, kad agentas pasieks galutinį tikslą: nugalėti oponento karalių. Su tokia konfigūracija agentas išmoks atlikti tik tai dalį galutinio tikslo.

3.2 Taisyklės ir vertės funkcija

Taisyklės (angl. *policy*) apibrėžia agento elgesį ir strategiją tam tikru laiku aplinkoje. Kokį veiksmą pasirinkti agentui priklauso nuo taisyklių aibės. Taisyklė (angl. *policy*) dažnai žymima simboliu $\boldsymbol{\pi}$**.** Iš esmės, taisyklė $\boldsymbol{\pi}$ yra atvaizdavimas iš būsenos į veiksmą, kitaip sakant, taisyklė pasako kokį veiksmą reikia atlikti tam tikroje būsenoje ir matematiškai (form. 2) apibrėžiama kaip

funkcija:

$$\pi(s) : S \rightarrow A.$$

form.2 Taisyklės atvaizdis [Sb16]

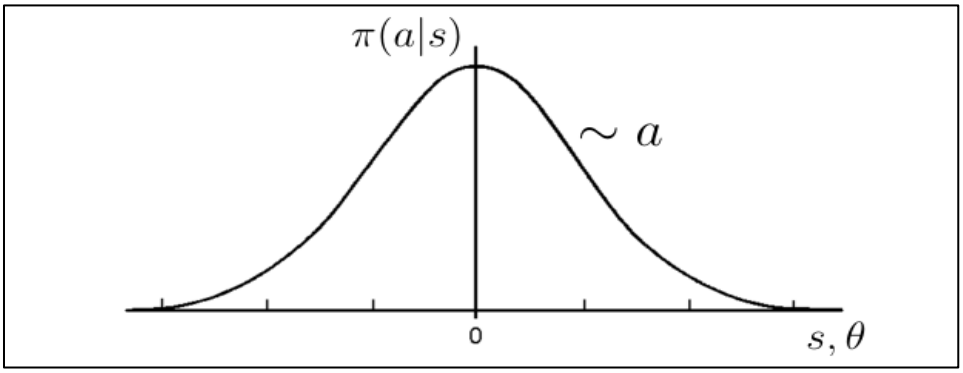
Toks taisyklės apibrėžimas tinka tik determinuotoje aplinkoje ir vadinasi determinuota taisyklė (angl. *deterministic policy*). Pavyzdžiui, turime aplinką – labirintą ir agentą – robotą. Robotas gali vaikščioti į kairę, dešinę, pirmyn ir atgal. Agento tikslas rasti išėjimą. Tada galimos taisyklės yra:

- Visada judėti į priekį
- Pabandyti apeiti sieną, jei nepavyko atlikti paskutinį veiksmą
- Jeigu reikia pasirinkti naują kelią, sukti į dešinę

Iš kitos pusės, egzistuoja taip vadinama stochastinė taisyklė ir užrašoma kaip:

$$\pi(a|s) = \Pr[a|s, \theta]$$

form.3 Taisyklės apibrėžimas naudojant tikimybę [Sb16]



grafikas 1: Veiksmai iš taisyklės, kuri šiuo atveju yra normalusis skirstinys [Opp18]

Pagal skirstinį atsitiktinai pasirenkamas veiksmas a būsenoje s pagal parametro vektorių θ (form. 3). Pavyzdys taisyklės, kaip tikimybinis skirstinys, būtų normalusis skirstinys, kur atsitiktinai pasirenkamas veiksmas a yra pavaizduotas 1 grafike.

Vertės funkcija (angl. *value function*) pasako būsenos reikšmę agentui, kai agentas atlieka veiksmus pagal taisyklę π. Kitaip sakant tai yra skaičius, kuris agentui būsenoje s apibrėžia tos būsenos naudingumą, kai agentas atlieka veiksmus pagal taisyklę π. Vertės funkcija yra apibrėžima kaip [Sb16]

$$V^{\pi}(s) = E_{\pi}[\, G_t | S_t = s] = \; E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] \; \forall \, s \; \in S$$

form.3 Atsitiktinio atpildo vidurkis apibrėžia vertės funkcija [Sb16]

Formulė 3 pasako, kad vertės funkcija yra atsitiktinio atpildo vidurkis (angl. *the expected cumulative discounted reward*) būsenoje s pagal taisyklę π. Kuo didesnė vertės būsenos reikšmė, tuo būsena agentui yra geresnė. Pagrindinis tikslas pastiprinto mokymo rasti optimaliausią taisyklę π^* (angl. *optimal policy*), pagal kurią agentas gaus didžiausią atsitiktinio atpildo vidurkį iš visų būsenų (form. 4).

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \;\; E_{\pi}[G_t|\pi] \;\; \forall s \in S$$

form.4 Optimaliausios taisyklės apibrėžimas [Adb17]

Išspręsti pastiprinto mokymo užduotį reiškia rasti tokią taisyklę, kuri duoda agentui didžiausią sukauptą atpildą ilguoju laikotarpiu. Optimaliausios taisyklės apibrėžimas Markovo

sprendimo procesui bus toks: taisyklė π yra geresnė už arba lygu taisyklei π' jeigu jos atsitiktinio atpildo vidurkis didesnis arba lygus vidurkiui taisyklės π' visose būsenose. Kitaip tariant, $\pi \geq \pi'$ tada ir tik tada jeigu $V^{\pi}(s) \geq V^{\pi'}(s)$ visose būsenose $s \in S$. Visada egzistuoja nors vieną taisyklę, kuri yra geresnė už visų kitų. Optimaliausią taisyklę galima išreikšti su optimaliausia vertės funkcija (form. 5).

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \;\; \forall s \in S$$

form.5 Optimaliausios vertės funkcijos apibrėžimas [Sb16]

3.3 Q funkcija ir Bellmano lygtis

Q funkcija arba būsenos ir veiksmo funkcija (angl. *state-value function*) pasako atlikto veiksmo a reikšmę būsenoje s pagal taisyklę π. Q-funkcija yra užrašoma kaip $Q(s,a)$ arba

$$Q^{\pi}(s,a) = E_{\pi}[\, G_t | S_t = s, A_t = a] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

form.6 Q funkcijos apibrėžimas [Sb16]

Skirtumas tarp vertės funkcijos ir Q funkcijos yra tame, kad vertės funkcija pasako būsenos s vertę agentui, o Q funkcija pasako veiksmo a vertę būsenoje s.

$$V^{\pi}(s) = \max_{a} Q^{\pi}(s,a) \; \forall s \in S$$

form.7 V funkcijos apibrėžimas naudojant Q funkciją [Adb17]

Vertės funkcija galima išreikšti naudojant Q funkciją (form. 7). Q funkcija leidžia apibrėžti ryšį tarp veiksmo ir būsenos, bet kaip keičiasi kitų būsenų reikšmės, kai agentas sąveikauja su aplinka? Čia atsiranda Bellmano lygtis. Richardas Bellmanas buvo amerikiečių matematikas, kuris išvedė šias lygtis, leidžiančias pradėti dinamiškai spręsti pastiprinto mokymo Markovo sprendimo procesus. Kaip jau minėjome:

$$P^{a}_{ss'} = Pr(s_{t+1} = s' | s_t = s, a_t = a) \quad R^{a}_{ss'} = E[r_{t+1} | s_t = s, s_{t+1} = s', a_t = a]$$

form.8 ir 9. Atpildo ir tikimybės apibrėžimas pastiprintame mokyme [Gre17]

Jeigu agentas bus pradinėje būsenoje s ir atliks veiksmą a tai jis pereis į būseną s' su tikimybe $P^{a}_{ss'}$ (form. 8). Atsitiktinio atpildo vidurkis $R^{a}_{ss'}$, kuris yra gaunamas jeigu agentas pradės būsenoje s, atliks veiksmą a ir pereis į kitą būseną s' (form. 9). Pagal atpildo ir vertės funkcijos apibrėžimą (form. 3) mes galime išreikšti vertės funkciją kaip

$$V^{\pi}(s) = E_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right]$$

Sekantis žingsnis bus gauti iš formulės pirmą atpildą ir išreikšti kitus kaip sumą [Gre17]

$$V^{\pi}(s) = E_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s\right]$$

form.10. Vertės funkcija užrašoma kaip suma dviejų narių [Gre17]

Atsitiktinio atpildo vidurkį galima apibendrinti, kaip sumą visų galimų veiksmų ir visų galimų atpildų (form. 11). Iš esmės bandome išreikšti du sumos narius naudojant galutinio atpildo formulę. Antrą narį užrašome rekursyviai (4) [Gre17].

$$E_{\pi}[\, r_{t+1} | S_t = s] = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} R^{a}_{ss'} \; (3)$$

form.11. Atsitiktinio atpildo apibendrinimas [Gre17]

$$E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2} \mid S_t = s\right] = \sum_a \pi(s,a)\sum_{s\prime} P_{ss\prime}^a\, \gamma E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2} \mid S_t = s\right]$$

form.12. Apibrėžia 10 formulės antrąjį narį [Gre17]

Įstatome narius į formulę ir pernešame bendrus narius į kairę pusę ir gauname 13 formulę [Gre17].

$$V^\pi(s) = \sum_a \pi(s,a)\sum_{s\prime} P_{ss\prime}^a\left[R_{ss\prime}^a + \gamma E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2} \mid S_{t+1} = s'\right]\right]$$

form.13. Formulė gaunamą kai į 10 įstatome 11 ir 12 [Gre17]

Nesunku pastebėti, kad atsitiktinio atpildo vidurkį galima užrašyti pagal pirmą lygtį, kaip vertės funkcija kitoje būsenoje s' ir rezultate gauname Bellmano lygtį vertės funkcijai (form. 14).

$$V^\pi(s) = \sum_a \pi(s,a)\sum_{s\prime} P_{ss\prime}^a\,[R_{ss\prime}^a + \gamma V^\pi(s')]$$

form.14. Bellmano lygtis vertės funkcijai [Gre17]

Analogiškai yra išvedama ir Q funkcijos Bellmano lygtis (form. 15).

$$Q^\pi(s,a) = \sum_{s\prime} P_{ss\prime}^a\left[R_{ss\prime}^a + \gamma\sum_{a\prime}\pi(s',a')Q^\pi(s',a')\right]$$

form.15. Bellmano lygtis Q funkcijai [Gre17]

Bellmano lygtis leidžia išreikšti vertės funkcijas arba Q-funkcijas būsenoje s rekursyviai naudojant kitas būsenas s_{t+1} ir tai leidžia naudoti pastiprintame mokyme iteracinius algoritmus. Tai reškia, kad įmanoma paskaičiuoti vertę visų agento būsenų, nes jeigu mes žinome reikšmę sekančios būsenos galima tiksliai pasakyti reikšmę einamosios būsenos [Gre17].

4 Pastiprinto mokymo algoritmai

4.1 Q funkcijos iteracija ir taisyklių iteracija

Sprendžiant Bellmano lygtį galima surasti optimaliausią vertę $V^*(s)$ ir taisyklę π^*. Vienas iš Bellmano lygties sprendimo būdų yra dinaminis programavimas [Pm10].

Dinaminis programavimas – tai uždavinio sprendimo metodas „iš apačios į viršų“. Pirmiausia mes išsprendžiame visus paprasčiausius duoto uždavinio atvejus, t.y., mažiausius dalinius uždavinius ir įsimename gautus rezultatus. Remdamiesi gautais sprendiniais, randame didesnių dalinių uždavinių sprendinius ir t.t. Šis metodas yra efektyvus tada, kai pačių mažiausių dalinių uždavinių nėra labai daug (t.y., kai jų skaičius polinomiškai priklauso nuo pradinio uždavinio dydžio) ir kai pradinio uždavinio sprendiniui rasti mums nereikia visų anksčiau gautų rezultatų, o pakanka tik tam tikros jų dalies [Dič05].

Tegul V_k bus vertės funkcija ir Q_k yra Q funkcija darant prielaidą, kad egzistuoja k epizodų. Q funkcijos iteracija arba reikšmių iteracija prasideda nuo atsitiktinės reikšmės V_0 ir naudoja Bellmano lygtį $k+1$ reikšmėms gauti [Pm10]

$$Q^\pi(s,a) = \sum_{s\prime} P_{ss\prime}^a\,[R_{ss\prime}^a + \gamma V_k(s')]\ kai\ k \geq 0 \qquad V_k(s) = \max_a Q_k(s,a)\ kai\ k > 0$$

Atsitiktinė vertės funkcijos reikšmė konverguoja iki optimaliausios reikšmės iteraciniu būdu ir iš jos yra gaunama optimaliausią taisyklę. Egzistuoja dvi algoritmo versijos – paprasta reikšmių iteracija ir asinchroninė reikmių iteracija arba Q funkcijos iteracija [Pm10]. Jų pagrindinis skirtumas yra tame, kad reikšmių iteracijoje yra naudojama reikmių funkcija vietoj Q funkcijos (pav. 4 ir 5).

procedure value_iteration (S, A, P, R, θ):
$k := 0$
repeat
$k := k + 1$
for each state s do:
$$V_k(s) = \max_a \sum_{s\prime} P^a_{ss\prime} \left[R^a_{ss\prime} + \gamma V_{k-1}(s')\right]$$
until $\forall |V_k(s) - V_{k-1}(s)| < \boldsymbol{\theta}$
for each state s do
$$\pi(s) = \operatorname{argmax}_a \sum_{s\prime} P^a_{ss\prime} \left[R^a_{ss\prime} + \gamma V_k(s')\right]$$
return π, V_k

pav. 4 V funkcijos iteracinis algoritmas [Pm10]

procedure async_value_iteration(S, A, P, R, θ):
repeat
select state s
select action $\boldsymbol{a}$
$$Q(s,a) = \sum_{s\prime} P^a_{ss\prime} \left[R^a_{ss\prime} + \gamma \max_{a\prime} Q(s', a')\right]$$
until termination
for each state s do
$$\pi(s) = \operatorname{argmax}_a Q(s,a)$$
return π, Q

pav. 5 Q funkcijos iteracinis algoritmas [Pm10]

procedure policy_iteration (S, A, P, R):
set random π
repeat
noChange ← *true*
Solve
$$V[s] = \sum_{s\prime} P(s'|s, \pi(s)) \left[R^a_{ss\prime} + \gamma V(s')\right] (1)$$
for each $s \in S$ do
Let QBest = $V(s)$

for each $a \in A$ do
$$Q(s,a) = \sum_{s\prime} P^a_{ss\prime} \left[R^a_{ss\prime} + \gamma \max_{a\prime} Q(s', a')\right] (2)$$
If $Q(s,a)$ > QBest then
$\pi[s] \leftarrow a$
QBest ← $Q(s,a)$
noChange ← *false*
until noChange
return π

pav. 6 Optimalios taisyklės paieškos algoritmas [Pm10]

Iš kitos pusės, egzistuoja algoritmas, kuris operuoja taisyklėmis. Taisyklių iteracijos pagrindinė idėja yra iteracinis taisyklių pagerinimas. Algoritmas pradeda nuo atsitiktinės taisyklės

π_0, tada randame šios taisyklės vertės funkciją, jei vertės funkcija nėra optimali, tada pageriname taisyklę. Tokio algoritmo veikimo principą galima užrašyti kaip seką:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

form. 16: Iteracinis taisyklės tobulinimas [Sb16]

Formulė 16 apibrėžia iteracinį optimaliausios taisyklės išvedimą, kur $\xrightarrow{E}$ (angl. *evaluate*) reiškia gauti iš taisyklės vertės funkciją (pav. 6 vertės funkcija yra randama sprendžiant 1 lygybę), o I (angl. *improve*) pasako taisyklės tobulinimo žingsnį (pav. 6 taisyklės tobulinimą apibrėžia 2 lygtis). Visas procesas vyksta cikle, kol nebus rasta optimaliausią taisyklę (pav. 6). Spręsti 1 lygtį (pav. 6) galima Gauso ir Žordano metodu (nuoseklus nežinomųjų eliminavimas) arba iteraciniu metodu [Pm10]. Vertės funkcijos išvedimo lygtis yra kiekvienoje aplinkos būsenoje.

4.2 Q mokymas

Q mokymo algoritmas priklauso TD mokymo algoritmų šeimai. TD (angl. *temporal difference*) metodai apskaičiuoja vertės funkcijas arba jų apytiksles reikšmes [Tes95]. Šie metodai skiriasi nuo kitų metodų, nes bando sumažinti einamąją nuoseklių prognozių paklaidą, o ne visą, ilgalaikę prognozavimo paklaidą. Pagrindinis mechanizmas šiam tikslui pasiekti yra perrašyti vertės funkcijų atnaujinimą Bellmano lygtimi. TD algoritmai kombinuoja dinaminį programavimą ir Monte Carlo metodus [Tes95]. Algoritmas padidina prognozavimo tikslumą naudojant saviranką (angl. *bootstrap*). Savirankos (angl. *bootstrap*) idėją galima nusakyti taip: į imtį pažvelkime, kaip į mažą populiaciją. Daug kartų iš jos atrinkę poaibį imties, turėtume gauti objektyvesnius paklaidų įverčius [Efr03]. Savirankos metodas reikalauja daug labai intensyvių skaičiavimų, nes pats procesas yra iteracinis. Taip kiekviename atnaujinimo žingsnyje sumažinama prognozės dispersija. TD mokymo algoritmai naudoja TD atnaujinimo taisyklę:

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))$$

form. 17: TD atnaujinimo taisyklė [Sb16]

17 formulė pasako vertės funkcijos atnaujinimo taisyklę. Formulės dalis $r + \gamma V(s') - V(s)$ pasako skirtumą tarp dabartino, einamojo atpildo $r + \gamma V(s')$ ir tikėtino atpildo $V(s)$ padauginto iš mokymo žingsnio α. Kitaip sakant, formulės dalis apibrėžia klaidos funkciją ir vadinasi TD klaida. Algoritmo tikslas minimizuoti klaidos funkciją.

Vienas iš tokių algoritmų yra *Q* mokymas (angl. *Q-learning*). Jo veikimo principas yra toks:

1. Pradedam algoritmą nuo lentelės arba matricos Q kur užrašome poros (būsena, veiksmas) reikšmę
2. Sąveikaujant su aplinką gauname s, a, r, s' (dabartinė būsena, veiksmas, gautas atpildas ir naują būseną). Šiame žingsnyje reikia nuspręsti kokį veiksmą atlikti, agentas gali išnaudoti seną taisyklę arba ištirti naują (angl. *explore and exploit problem*).
3. Atnaujinam Q reikšmę naudojant formulę

$$Q_n(\mathrm{s}, a) = (1 - \alpha) * Q_s(\mathrm{s}, \mathrm{a}) + \alpha\big(r + y * max_{a'} Q(\mathrm{s}', a')\big)$$

form. 18: Q algoritmo atnaujinimo taisyklė [Wd92]

4. Tikrinam konvergavimo sąlygas ir kartojam 2 žingsnį.

Atsižvelgiant į atpildą gautą iš aplinkos, agentas sudaro naudingumo funkciją *Q*, kuri vėliau suteikia jam galimybę pasirinkti elgesio strategiją ir atsižvelgti į patirtį [Wd92]. Pirmas algoritmo žingsnis yra suteikti tam tikrą reikšmę *Q* funkcijai. Kiekvieno laiko momentu t agentas pasirenka veiksmą a_t, analizuoja gautą atpildą už veiksmą r_t, pereina į kitą būseną s_{t+1} (kuri priklauso nuo praeitos būsenos ir pasirinkto veiksmo) ir galų gale atnaujina *Q* reikšmę: Q_s yra sena reikšmė, Q_n yra nauja reikšmė, α yra mokymosi žingsnis ($0 < \alpha \leq 1$), kuris pasako kiek naujos informacijos

įsisavinti keičiant seną [Wd92]. Koeficientas *y* pasako veiksmo svarbumą. Jeigu koeficientas *y* artėja į 0 tai agentas pasirinks atlikti einamąjį veiksmą, (toks agentas yra „trumparegis“), o jei artėja į 1 tai agentas stengsis gauti ilgalaikį didelį atpildą [Rnd16].

Kai mes gauname naują informaciją iš aplinkos ir iš karto perrašome senus pastebėjimus rezultate gauname nestabilų mokymą. Q algoritmas leidžia išvengti šios problemos, nes yra naudojamas vidurkis tarp naujos ir senos reikšmės padaugintos iš mokymo žingsnio. Reikšmių iteracijoje ir Q iteracijoje kiekviename žingsnyje yra nagrinėjama kiekvieną būseną ir kiekvienai būsenai yra daromas Bellmano lygties atnaujinimas, o Q mokyme atnaujinamos ne visos būsenos iš karto, o tik gautos iš aplinkos. Svarbiausiai yra tai, kad Q mokymo procese agentas nežino būsenos perėjimo tikimybės ir atpildų. Jis tiesiog nustato visas nežinomas reikšmės, kai gauna pastebėjimus iš aplinkos.

4.3 Gilus Q mokymas

Paprastas Q mokymas yra pritaikomas aplinkoms, kur būsenų ir veiksmų skaičius yra baigtinis ir kiekvienoje būsenoje yra diskretus skaičius veiksmų. Bet, ne visos aplinkos yra diskrečiai apibrėžiamos. Pavyzdžiui, jeigu aplinkoje yra veiksmas kuris yra intervale nuo 0 iki 1, jis gali apibrėžti fizikinę jėgą arba impulso didį, tai visiems veiksmams išsaugoti lentelėje reikėtų be galo daug atminties. Taip pat, egzistuoja daug žaidimų kur sudėtingumas, tiek būsenų erdvės dydžio ir veiksmų skaičiaus požiūriu kiekviename sprendimo cikle yra labai didelis. Autorius [Sha50] nustatė, kad būtent šachmatų būsenų skaičius yra ~10^{50}, pagal [Asb08] pokerio žaidimo (heads up no-limit Texas holdem poker) apytiksliai yra ~10^{71}, ir knygoje „Dream Pool Essays“ autorius Shen Kuo (1031- 1095) nustatė, kad Go žaidimo apytiksliai ~10^{170}. Taip pat atsirado daug naujų žaidimų, kaip realaus laiko strateginiai žaidimai, kur būsenų ir veiksmų skaičius yra dar didesnis. Jeigu paimti „Starcraft“ žaidimą, tai šio žaidimo skaičius yra daug kartų didesnis už bet kurį iš tų, nes jeigu pasirinkti žemėlapį, kurio dydis yra 128x128 ir 400 vienetų, tai būsenų skaičius bus 16384^{400} , o tai ~10^{1685}, dar neįskaitant tuo, kad kiekvienas vienetas turės savo vidines būsenas [Osu13].

Sprendimą šiai problemai pasiūlė 2013 metais DeepMind kompanija su savo nauju algoritmu. Autoriai [Mks+13] tobulina paprasta Q mokymo algoritmą neuroniniu tinklu. Neuroninis tinklas yra naudojamas Q reikšmių aproksimavimui su parametru θ ir užrašomas kaip $Q(s,a;\theta) \approx Q^*(s,a)$ ir vadinasi DQN (angl. *deep Q network*) gilus Q tinklas. [Mks+13]. Algoritmas buvo panaudotas agento apmokymui „Atari“ žaidimuose, kur tikslas buvo išmokti

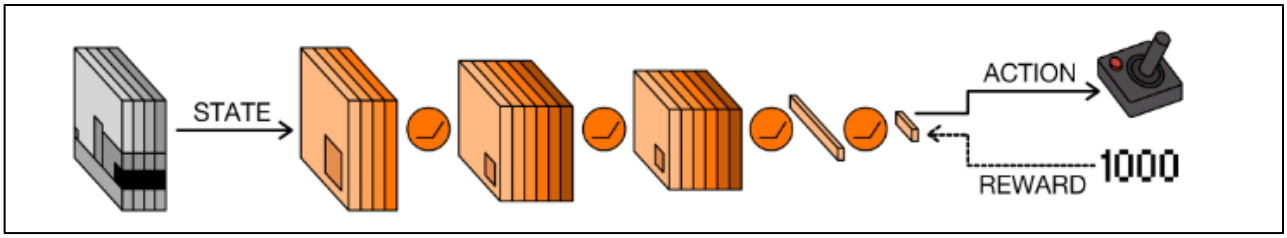
Initialize replay memory $\mathcal{D}$ **to capacity** N
Initialize action-value function Q **with random weights**
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

pav. 7 Gilaus Q mokymo algoritmo pseudokodas, kuris buvo pateiktas DeepMind 2013 [Mks+13]

agentą optimaliai žaisti žaidimą ir mokymo metu paduoti agentui žaidimo ekrano kopiją (pav.7).

Algoritmo (pav.7) pagrindinė idėja yra apmokyti agentą žaisti „Atari“ žaidimą naudojant buferį, kuris susideda iš praeitų aplinkos pastebėjimų. Iš pradžių agentas atsitiktinai pasirinka veiksmus naudojant „Epsilon“ strategiją. „Epsilon“ strategija (angl. *epsilon greedy policy*) yra metodas, kuris nustato agento tikimybę ϵ atlikti atsitiktinį veiksmą a ir su $1 - \epsilon$ tikimybe atlikti geriausią veiksmą dabartinėje būsenoje s. [Mks+13] Iš esmės, „Epsilon“ strategija leidžia agentui ištirti naują taisyklę arba atvirkščiai išnaudoti seną. Iš pradžių $\epsilon = 1$ ir po kiekvienos iteracijos yra po truputi mažinama. Tai leidžia agentui mokymo pradžioje susipažinti su aplinka, išsaugoti į buferį įgytą informaciją ir iš jos pradėti mokytis. Kai mokymo epochos skaičius pradeda didėti, ϵ mažėja ir agentas bando ieškoti optimaliausio veiksmo. Agento patirtis yra buferis, kuris saugo savyje rinkinius (s, a, r, s'). Taigi, po kiekvieno sėkmingo veiksmo agentas išsaugo į buferį dabartinę būseną, veiksmą, atpildą ir sekančią būseną. „Atari“ žaidime aplinkos stebėjimai yra žaidimo ekrano kopijos 210x160x3 ir tuo pačiu metu pasako agento būseną. Neuroninio tinklo tikslas paimti kaip įvestį ekrano kopiją ir veiksmą ir pasakyti šios kombinacijos Q reikšmę. Kitaip tariant, vietoj lentelės gilus Q mokymas naudoja neuroninį tinklą, kuris prognozuoja veiksmo ir būsenos reikšmę. Norėdami panaudoti neuroninį tinklą reikšmių aproksimavime, apskaičiuojame netiesinės funkcijos $Q(s, a)$ galutinę reikšmę (galutinė reikšmė arba agento reikšmė sekančioje būsenoje $Q'(s', a')$) (angl. *target*) naudojant Bellmano lygtį ir tada darome prielaidą, kad mes turime prižiūrimo mokymo problemą ir optimizuojame naudojant gradientinį nusileidimą. Bet šitoje vietoje atsiranda problemos su gradientiniu nusileidimu, nes optimizavimo metodas reikalauja, kad duomenys būtų nepriklausomos ir vienodai paskirstyti (angl. *independent and identivally distributed*). „Atari“ žaidimo duomenys nepatenkina tokio reikalavimo [Mks+13].

4.3.1 Gilaus Q mokymo problematika



pav. 8 Neuroninio tinklo panaudojimas pastiprintame mokyme [Adb17]

Dirbtinio neuroninio tinklo įvestis pastiprintame mokyme yra pertvarkytas žaidimo vaizdas, praktikoje žaidimo ekrano kopija yra sumažinama. Pats tinklas susideda iš konvoliucinių ir pilnai sujungtų sluoksnių, kur tarp kiekvieno sluoksnio „ReLU“ aktyvacijos funkcija [Mks+13]. Galutiniame sluoksnyje tinklas išveda veiksmą, kuris atitinka vieną iš galimų žaidimo valdymo įėjimų (pav. 8). Atsižvelgiant į dabartinę būseną ir pasirinktą veiksmą, žaidimas grąžina naują įvertinimą - naujus žaidimo taškus [Adb17]. Gilus Q mokymas naudoja skirtumą tarp naujo atpildo ir ankstesnio ir skaičiuoja paklaidą. Metodas kombinuoja neuroninį tinklą su dinamišku duomenų srautu, kuris susideda iš aplinkos stebėjimų arba būsenų. Šioje vietoje atsiranda kelios problemos.

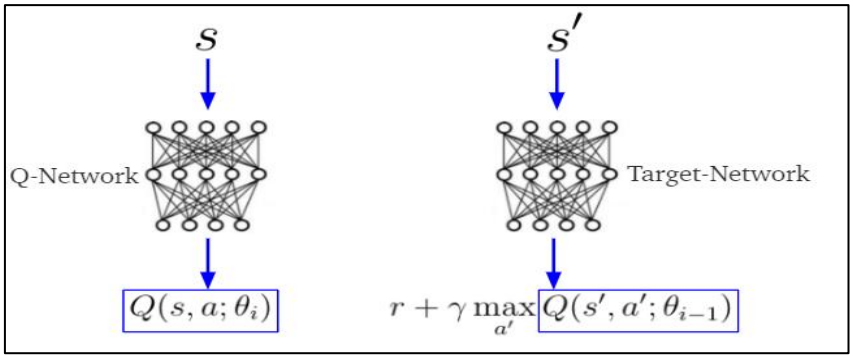
Kai agentas sąveikauja su aplinka ir kaupia įgytą informaciją kiekviename epizode, tai duomenys yra priklausomos, nes jie yra visi arti vienas kito ir visi priklauso vienam epizodui. Taip pat informacijos pasiskirstymas gautas mokymo metu nesutaps su optimaliausios strategijos pasiskirstymu. Informacija bus gauta naudojant atsitiktinę strategiją, o mokymo tikslas yra apmokyti agentą žaisti optimaliausiu būdu. Šios problemos sprendimas yra naudoti didelį buferį ir kaupti informaciją, o mokymo tikslams ištraukti iš buferio atsitiktinį poaibį ir įvesti į neuroninį tinklą. Jeigu buferis pilnas, tai kiekvienas naujas buferio įrašas stumia seną informaciją iš buferio ir tai leidžia agentui turėti naujausią aplinkos informaciją. Tokia agento apmokymo strategija vadinasi patirties buferiu (angl. *replay buffer strategy*) [Mks+13]. Taigi, tai beveik sprendžia duomenų priklausomybės problemą.

Nesunku pastebėti prieštaravimą su Markovo savybe. Priminsiu, kad Markovo savybė teigia, kad agentas atrastų optimaliausią strategiją užtenka analizuoti tik einamąjį pastebėjimą iš aplinkos arba dabartinę būseną. Pavyzdžiui, agentas žaidžia „Atari“ žaidimą „Pong“ (pav. 9) ir viena ekrano kopija pilnai neapibrėžia viso žaidimo būseną, nes agentas gaudamas įvesti (pav. 9) negali tiksliai pasakyti kamuolio ir priešininko greitį ir kryptį. Taigi, tai prieštarauja Markovo savybei ir perkelia MDP į POMDP (angl. *partially observable MDPs*). POMDP tai yra MDP, kur Markovo savybė yra nepatenkinta [Hau20]. Pavyzdys, tokios aplinkos bus kortų žaidimai, kur agento stebėjimai (kortai jo rankoje ir ant stalo) sudaro POMDP, nes agentas nemato visų priešininko kortų. Iš tikrųjų, „Atari“ žaidimuose galima laikyti keletą žaidimo ekrano kopijų ir toks masyvas iš k elementų sudaro vieną pastebėjimą iš aplinkos. Tai leidžia agentui suprasti būsenos dinamiką ir atrasti reikalingą informaciją mokymui [Mks+13].

pav. 9 Žaidimo „Pong“ ekrano kopija

Iš kitos pusės, egzistuoja problema su Bellmano lygties atnaujinimu. Bellmano lygtis atnaujina reikšmę $Q(s,a)$, naudojant sekančią reikšmę $Q'(s',a')$, bet būsenos s ir s' turi tik vieno žingsnio skirtumą. Toks duomenų skirtumas sunkina neuroniniam tinklui aproksimavimo uždavinį ir galutiniame rezultate yra gaunamas nestabilus mokymas. Atlikdami tinklo parametrų atnaujinimą, kad $Q(s,a)$ reikšmė būtų arčiau norimo rezultato, netiesiogiai galima pakeisti $Q'(s',a')$ reikšmę ir kitų šalia esančių būsenų. Kai atnaujinama reikšmė Q būsenoje s, tai sekančiuose būsenose $Q'(s',a')$ reikšmės gali blogėti ir bet koks bandymas reguliuoti reikšmę gali sugadinti $Q(s,a)$ reikšmės aproksimaciją [Mks+13].

Šios problemos sprendimas (pav. 10) remiasi tuo, kad naujas pastiprinto mokymo modelis naudoja du neuroninius tinklus. Vienas iš jų vadinasi galutinės Q reikšmės neuroniniu tinklu (angl. *target-network*). Po nustatytų žingsnių N pagrindinis neuroninis tinklas yra sinchronizuojamas su papildomu neuroniniu tinklu - daroma pagrindinio neuroninio tinklo kopiją. Papildomas tinklas aproksimuoja $Q(s',a')$ reikšmę būsenoje s' ir taip yra stabilizuojamas mokymo procesas.



pav. 10 Dviejų skirtingų neuroninių tinklų panaudojimas Q reikšmės ir galutinės Q reikšmės skaičiavimui [Opp18]

Apibendrinant visus metodus, galima užrašyti gilų Q mokymo algoritmą [Mks+13].

1. Pagrindinio $Q(s,a)$ ir papildomo $Q'(s',a')$ neuroninio tinklo koeficientams yra priskiriamos atsitiktinės reikšmės. $\epsilon \leftarrow 1$. Buferis yra tuščias.

2. Su tikimybe ϵ agentas pasirenka veiksmą a, kitaip $a = \underset{a}{\operatorname{argmax}}\ Q(s,a)$
3. Agentas atlieka veiksmą a, gauna atpildą r ir sekančią būseną s'
4. Išsaugoti rinkinį (s,a,r,s') į buferį D
5. Gauti atsitiktinį poaibį P iš buferio D
6. Kiekvienam elementui poaibyje P paskaičiuoti tikslą y
   6.1. Jei terminalinė būsena $y = r$
   6.2. Kitaip $y = r + \gamma \max_{a' \in A} Q(s',a')$
7. Apskaičiuoti nuostolio funkciją $\mathcal{L} = (Q(s,a) - y)^2$
8. Naudojant gradientinį nusileidimą atnaujinti $Q(s,a)$
9. Kiekviename n žingsnyje perkelti koeficientus iš $Q(s,a)$ į $Q'(s',a')$
10. Kol nepasiekta optimaliausia strategija kartoti nuo 2 žingsnio

4.4 Dvigubas Q mokymas

Vienas iš gilaus Q mokymo patobulinimų buvo pasiūlytas [Hgd15] autoriais. Darbe [Hgd15] autoriai teigia, kad išvardytas (pav.7) gilaus Q mokymo algoritmas turi tikimybę per daug įvertinti Q reikšmes ir tai sukelia nestabilumą mokyme ir rezultate agentas gali nesurasti optimaliausios strategijos. [Hgd15] teigia, kad problema yra dėl to, kad Bellmano lygtis naudoja max operaciją. Autoriai pasiūlė šios problemos sprendimą, kuris modifikuoja Bellmano atnaujinimą. Gilaus Q mokymo atnaujinimas vyksta pagal 18 formulę.

$$Q(s_t,a_t) = r_t + \gamma \max_a Q'(s'_t, a_t')$$

form. 18: Gilaus Q mokymo atnaujinimo taisyklė [Mks+13]

Q' yra papildomas neuroninis tinklas, kuris apskaičiuoja sekančios būsenos s' Q reikšmę ir yra sinchronizuojamas su pagrindiniu neuroniniu tinklu kiekvieną N žingsnį. [Hgd15] autoriai pasiūlė pasirinkti sekanti veiksmą a', naudojant pagrindinį tinklą Q, bet atnaujinti reikšmės per papildomą tinklą Q' ir rezultate gauname naują atnaujinimo taisyklę:

$$Q(s_t,a_t) = r_t + \gamma \max_a Q'(s', \underset{a}{\operatorname{argmax}}\ Q(s',a))$$

form. 19: Dvigubo gilaus Q mokymo atnaujinimo taisyklė [Hgd15]

[Hgd15] įrodė, kad toks naujas atnaujinimo būdas (form. 19) pilnai sprendžia reikšmių pervertinimą ir pavadino tokią architektūrą dvigubu Q mokymu.

4.5 SARSA algoritmas

(angl. *State-Action-Reward-State-Action*) SARSA yra TD algoritmas, kuris skiriasi nuo Q algoritmo tuo, kad priklauso nuo pasirinktos strategijos (angl. *on-policy*). Iš tikrųjų, tai yra dar viena Q algoritmo versija. SARSA algoritmas naudoja naują atnaujinimo taisyklę:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

form. 20: SARSA atnaujinimo taisyklė [Sb16]

Agentas, kuris naudoja SARSA algoritmą mokosi remdamasis savo dabartiniu veiksmu, tai ir atspindi 20 formulė, nes palyginant su Q algoritmu, SARSA nenaudoja maksimumo, todėl jis priklauso nuo pasirinktos strategijos ir vadinasi (angl. *on-policy*). Q algoritmo atveju agentas bando surasti geriausią veiksmą, kuris gali būti kitos strategijos, arba taisyklės dalis, todėl ir vadinasi (angl.*off-policy*). SARSA skiriasi nuo Q algoritmo tuo, kad algoritmas ieško optimaliausios strategijos, kai agentas tyrinėja aplinką, todėl SARSA algoritmas reikalauja gerą „Epsilon“ strategijos konfigūraciją. Iš kitos pusės, Q algoritmas iš karto bando konverguoti iki optimaliausios strategijos, kai ima kitos būsenos maksimalią reikšmę, todėl praktikoje yra dažniau naudojamas [Sb16].

4.6 Pastiprinto mokymo iššūkiai

Ištirti naujas taisykles arba išnaudoti sena (angl. *explore and exploit problem*). Agento sprendimų priėmimas aplinkoje susiveda į esminį pasirinkimą: tyrinėti ir surinkti daugiau informacijos, kuri gali paskatinti mus priimti geresnius sprendimus ateityje arba išnaudoti jau žinomą strategiją, kur mes priimame geriausią sprendimą, atsižvelgiant į dabartinę informaciją. Tai atsitinka todėl, kad pastiprintas mokymas vyksta dinamiškai, kai agentas interaktyviai sąveikauja su aplinka. Pastiprintame mokyme iš anksto nieks nesuteikia tam tikrų duomenų, todėl duomenys yra kaupiami mokymo metu ir veiksmai, kurių imamės, daro įtaką matomiems duomenims. Taigi, kartais verta imtis įvairių veiksmų, kad agentas gautų kuo įvairius ir naujus duomenis apie aplinką [Adb17].

Agento stebėjimai priklauso nuo jos veiksmų ir gali turėti stiprią koreliaciją su laiku. Agentai turi spręsti ilgalaikes priklausomybes nuo laiko: dažnai veiksmo pasekmės įvyksta tik po to, kai agentas pilnai pereina aplinką arba atlieka veiksmus. Tai vadinama (laikina) kredito priskyrimo problema [Adb17].

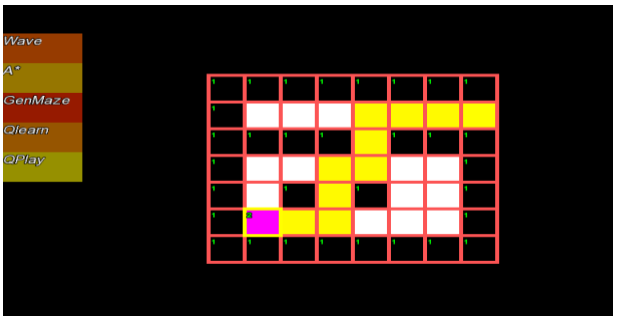
5 Q algoritmų praktinis pritaikymas

Pastiprinto mokymo algoritmai buvo praktiškai realizuoti naudojant „OpenAI gym“ karkasą. „OpenAI gym“ yra karkasas ir įrankių rinkinys, skirtas kurti ir palyginti pastiprinto mokymo algoritmus. Jis duoda galimybę laisvai struktūrizuoti agentą, bei yra laisvai suderinamas su bet kokia skaičiavimo biblioteka, pvz., praktiniame pritaikyme buvo panaudotas „OpenAI gym“ su „Tensorflow“ ir „Keras“ bibliotekomis. „OpenAI gym“ turi savyje rinkinį aplinkų (angl. environments), kur galima palesti pastiprinto mokymo algoritmus testavimo tikslams.

5.1 Praktinės aplinkos

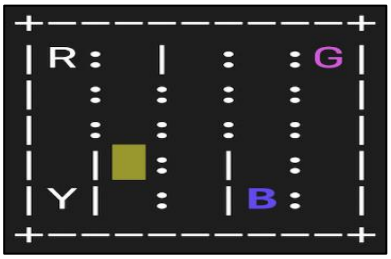
Eksperimentuose buvo panaudotos kelios aplinkos rezultatams gauti.

1. **Labirintas.** Aplinką buvo suprogramuota naudojant C++ programavimo kalbą ir naudojau atvaizdavimui SFML biblioteką. Agentas turi rasti trumpiausią kelią labirinte naudojant pastiprinto mokymo algoritmus (pav. 11).



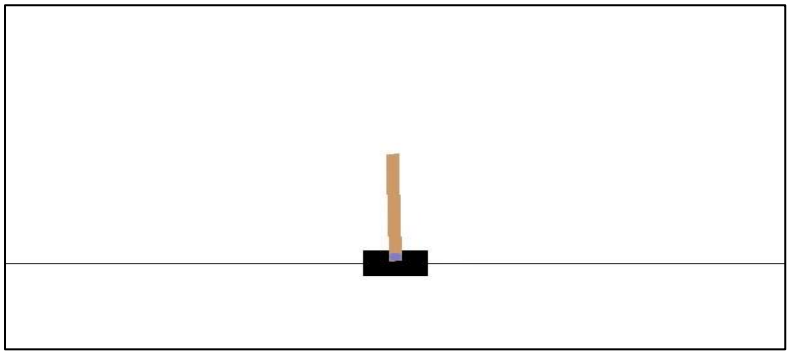
pav. 11 Labirintas, kur geltonai yra pažymėtas trumpiausias kelias

2. **Taxi-v2.** Dirbtinis agentas šioje aplinkoje sprendžia taksisto uždavinį. Yra 4 vietos (pažymėtos skirtingomis raidėmis) ir agento tikslas paimti keleivį vienoje vietoje ir išleisti jį paskirties vietoje. Agentas gauna teigiamą atpildą už sėkmingus veiksmus, bet praranda taškus kai atlieka neleistinus veiksmus kiekvienoje būsenoje. Taip pat agentas turi pervežti keleivį kuo greičiau, todėl aplinka baudžia agentą kiekvieną iteraciją (pav. 12).



pav. 12 Taxi-v2 aplinkos pavyzdys kur taške B yra keleivis ir G yra jo tikslas

3. **Cartpole-v0 ir v1.** Stulpas pritvirtinamas nepilnai prisukta jungtimi į vežimėlį, kuris juda į kairę ir į dešinę. Sistema kontroliuojama, kai vežimėliui taikoma +1 arba -1 jėga. Stulpo pradinė pozicija prasideda vertikaliai, o tikslas yra neleisti jam nukristi. Kiekvieną kartą, kai stulpas išlieka vertikalus, agentas gauna teigiama atpildą. Epizodas baigiasi, kai stulpas yra nuleistas daugiau nei 15 laipsnių, arba vežimėlis išeina iš ekrano ribų (pav 13).



pav. 13 Cartpole aplinkos pavyzdys

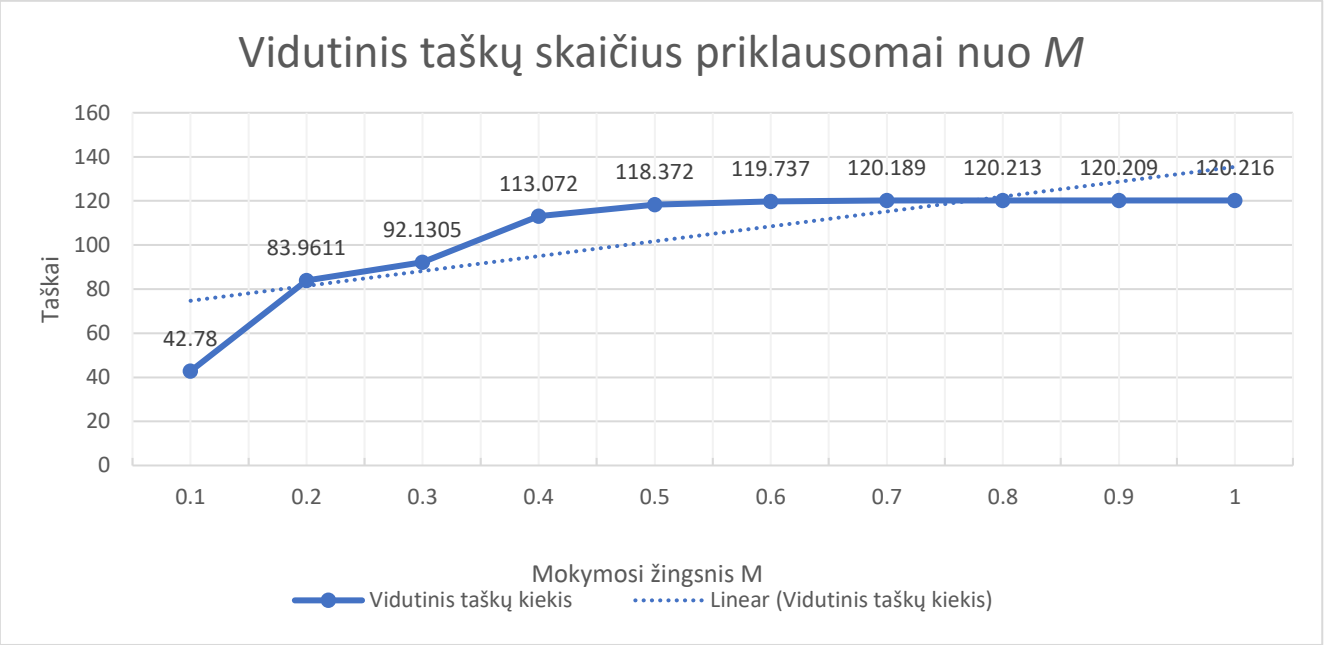
4. **MsPacman-v0.** Dirbtinis agentas valdo pagrindinį veikėją „Pac-Man“, vesdamas jį labirintu, kur „Pac-Man“ turi suryti visus taškelius, nepaliesdamas keturių jį gaudančių vaiduoklių. Kai suryjami visi taškeliai, pereinama į kitą lygį. Jei „Pac-Man“ paliečia vaiduoklis, prarandama viena gyvybė. Labirinte taip pat išdėstyti keli dideli energetiniai objektai, kuriuos surijus „Pac-Man“ keletą sekundžių gali valgyti vaiduoklius – kol energetinis užtaisas veikia, vaiduokliai tampa mėlyni ir pradeda tolti nuo veikėjo. Už vaiduoklio sunaikinimą skiriami taškai. Agento tikslas gauti kuo daugiau taškų (pav. 14).

pav. 14 Pac-Man aplinkos pavyzdys

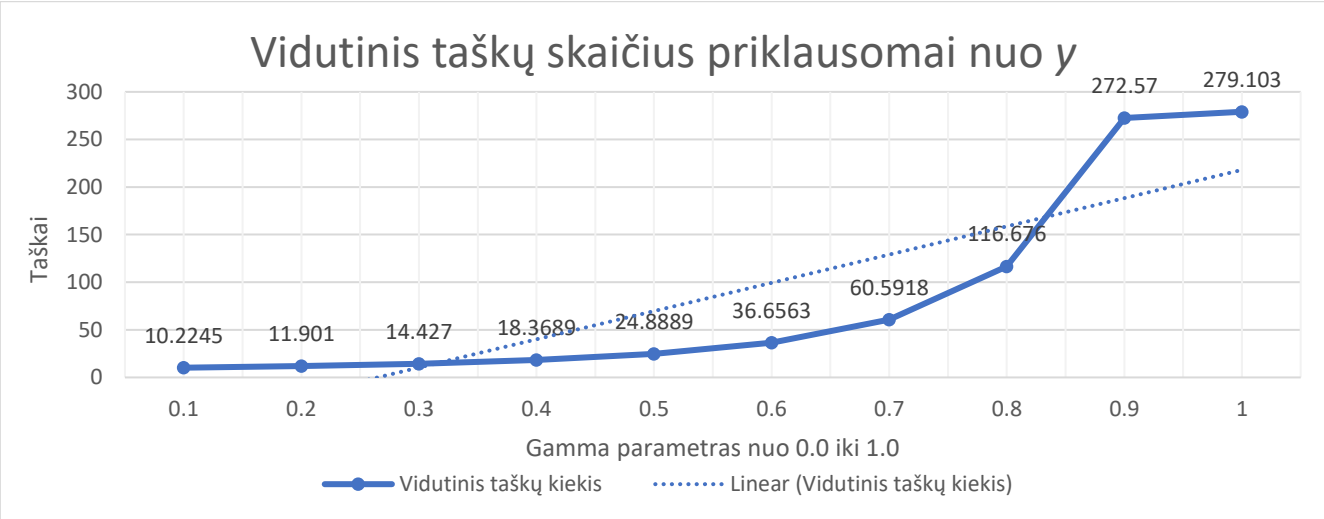
5.2 Rezultatai

5.2.1 Labirinto rezultatai

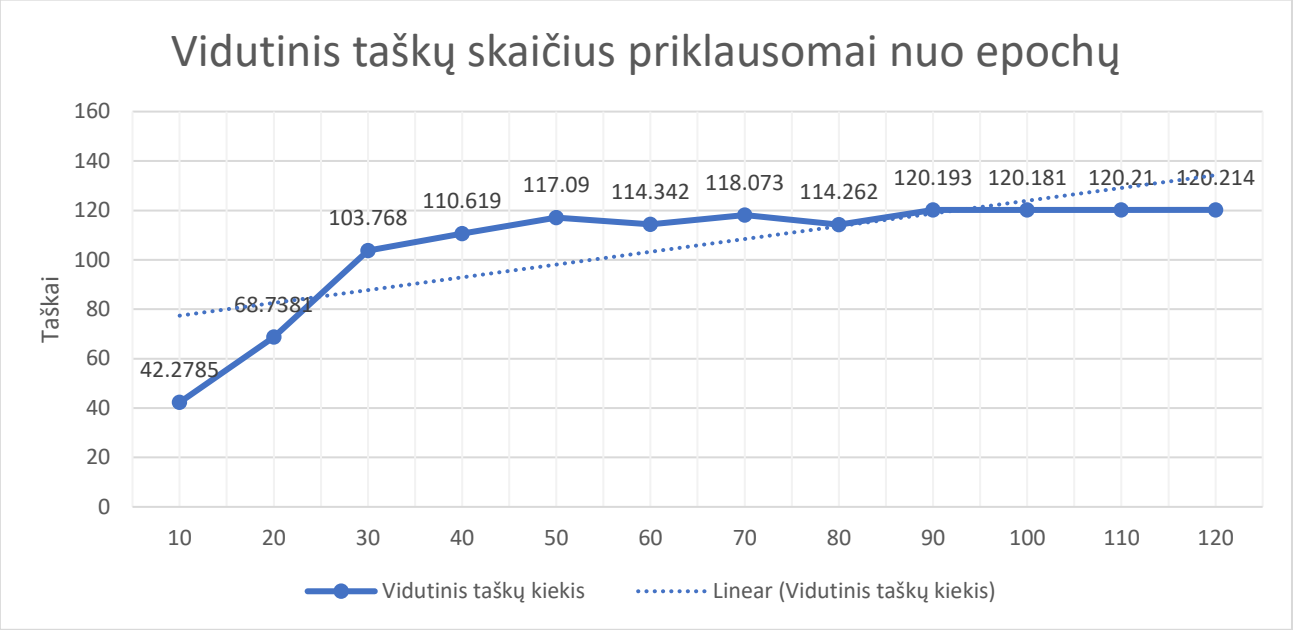
Tyrimo metų buvo suprogramuota aplinką „Labirintas“ (pav. 11), kur dirbtinis intelektas naudojant Q lentelę ir Q algoritmą buvo apmokytas rasti trumpiausią kelią. Labirintas susideda iš sienų ir laisvų langelių. Agentas gali atlikti 4 diskretus veiksmus (eiti į kairę, dešinę, viršų ir į apačią).Ant sienos negalima stovėti, bet jeigu agentas atlieka veiksmą, kuris rezultate veda į sieną, (*angl. Wall*) tai algoritmas baudžia agentą ir duoda jam -100 taškų. Taip pat kiekvienas agento žingsnis iš laisvo langelio į laisvą yra baudžiamas -1 tam, kad skatinti agentą rasti trumpiausią kelią. Jeigu agentas iš laisvo lango pereina į tikslą, tai agentas gauna maksimalius taškus. Nesunku pastebėti, kad lengvai galima pridėti papildomus (angl. bonus tiles) langelius kurie duos papildomai taškus agentui ir tada uždavinys rasti kelią gali būti pasunkintas taip, kad agentas rastu optimaliausią kelią su maksimaliu atpildu (angl. *visiting all points of interests*).



grafikas 2: Q taškų skaičius priklausomai nuo M (mokymosi žingsnis). Epoch = 48; Gamma = 0.8



grafikas 3: (Taškai ir y) Q taškų skaičius priklausomai nuo y (gamma). Epoch = 48, M = 0.4



grafikas 4: (Taškai ir iteracijos) Q taškų skaičius priklausomai nuo epochų (algoritmo iteracijų) skaičiaus. M = 0.4,Gamma = 0.8.

Vienas iš parametrų yra vidutinis taškų skaičius, jis pasako kiek vidutiniškai gavo atpildą agentas kai jis pasiekė tikslą. Agentas buvo apmokytas vaikščioti labirinte (pav. 11). Kuo didesnis vidutinis taškų skaičius tuo daugiau tikimybės yra, kad agentas pasieks tikslą. Iš 3 grafiko nesunku pastebėti, kad γ parametras pasako, kiek agentas yra trumparegis. Kai γ yra nuo 0 iki 0,5 agentas nenaudoja naujos patirties, todėl atpildas mažėja. Kai γ yra nuo 0,7 iki 1,0 agentas pradeda daugiau įsisavinti naujos informacijos apie labirintą. Iš 2 grafiko galima pamatyti, kad mokymo žingsnis kai artėja prie 1, didina vidutinį taškų skaičių, bet vidutinis taškų skaičius nuo 0.5 reikšmės sustoja sparčiai didėti. Epochų skaičius nuo 90 sustabdo taškų augimą (grafikas 4). Rezultate agentas išmoko rasti trumpiausią kelią labirinte.

5.2.2 Taxi-v2

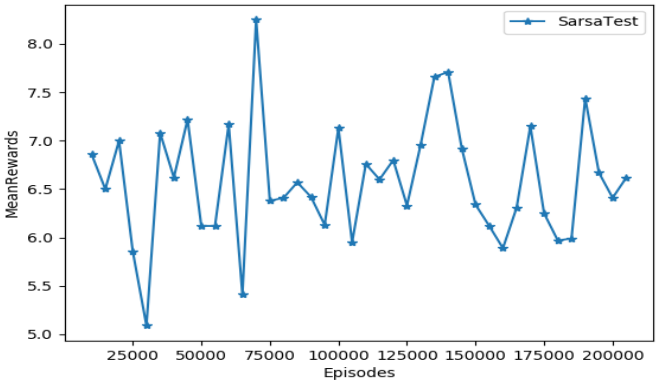
Agento tikslas šioje aplinkoje (pav. 12) išmokti kuo greičiau pervežti keleivį iš pradinio taško (pradinė keleivio padėtis kiekvienoje epochoje atsitiktinai keičiasi) iki galutinio tikslo. Keleivis gali laukti vienoje iš *R, Y, G, B* taškų. Iš viso yra 25 įmanomos agento pozicijos, 5 įmanomos keleivio pozicijos (keleivis gali ne tik laukti bet ir būti pačioje taksi), ir 4 galutiniai taškai. Rezultate agentas turi 500 diskrečių būsenų (25*5*4). Epocha baigiasi, kai agentas sėkmingai įvykdo transportavimo uždavinį. Aplinkoje agentas gali pasirinkti atlikti 6 veiksmus:

Galimi veiksmai aplinkoje „Taxi-v2“	
0	į pietus
1	į šiaurę
2	į rytus
3	į vakarus
4	Paimti keleivį
5	Išleisti keleivį

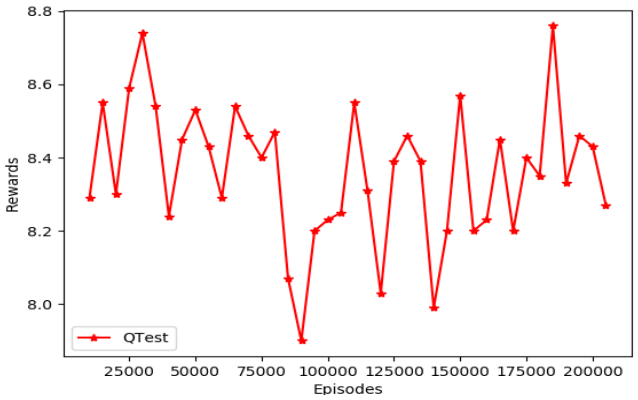
Atpildai aplinkoje „Taxi-v2“	
+20	Sėkmingai išleisti keleivį
-1	Kiekvieno žaidimo kadro kaina
-10	Jeigu agentas bando išleisti keleivį neteisingoje vietoje arba eiti į sieną

„Taxi-v2“ agentas buvo apmokytas kuo greičiau išleisti keleivį dviem būdais, naudojant SARSA ir Q algoritmus. SARSA algoritmo parametrai: $\alpha = 0.85$, $\gamma = 0.9, \epsilon = 1.0$ ir ϵ parametro mažinimo žingsnis buvo 0,99. SARSA ir Q algoritmai rado sprendimą po 8000

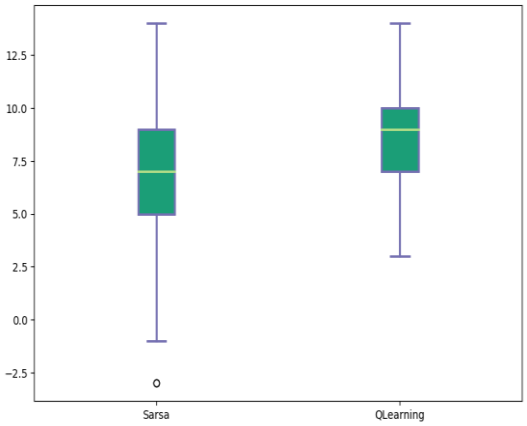
iteracijų. Mokymas truko 5 min. „Taxi-v2“ aplinka yra išspręsta, kai agentas sėkmingai perveža keleivį ir nepadaro nė vieno neleistino ėjimo. Todėl, buvo sukurtas testas, kuris skaičiavo kiekvieną neteisingą agento ėjimą. Q ir SARSA metodams praktiškai reikėjo 8000 iteracijų pilnai praeiti testą be klaidingų ėjimų. Q algoritmo parametrai: $\alpha = 0.4,\ \gamma = 0.9, \epsilon = 1.0$. SARSA algoritmas priklauso nuo strategijos, todėl jam reikia didelio mokymo žingsnio, nes galima įsivaizduoti, kad daroma paieška į plotį ir agentui reikia daugiau ištirti pasirinktą strategiją. Q algoritmo atveju yra daroma paieška į gylį, todėl mokymo žingsnis neviršija 0,5. γ buvo nustatytas 0,9, nes agentui reikia stengtis gauti ilgalaikį didelį atpildą, bet nepilnai. Q ir SARSA mokymo žingsniai buvo dauginami iš 0,999 kiekvieną iteraciją ir taip mažinami. Rezultate agentas išmoko žaisti žaidimą naudojant du algoritmus. SARSA algoritmo testavimo vidutinis taškų skaičius ~7,4, o Q algoritmo rezultatas ~8,9 (grafikas 7). Q algoritmas vidutiniškai gavo daugiau taškų aplinkoje (7 grafikas) per 8000 iteracijų. Grafikas 5, 6 parodo, kaip keičiasi agento gauti taškai, kai buvo didinamas iteracijų skaičius. Grafikas 8, 9 parodo, kad Q algoritmas ir SARSA beveik panašiai apmokė agentą pervežti keleivį, bet SARSA vietomis pasiekė 20 ėjimus, o Q algoritmas 15-16. Rezultate agentas išmoko optimaliai spręsti „Taxi-v2“ (be neleistinų ėjimų ir su greičiausiu maršrutu) ir Q algoritmas pasirodė efektyviau (grafikas 7 ir 6).



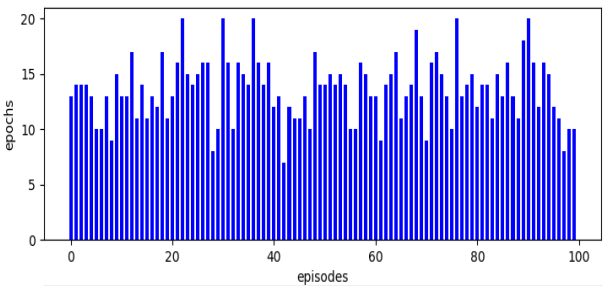
grafikas 5 Sarsa algoritmo rezultatai priklausomai nuo mokymo iteracijos



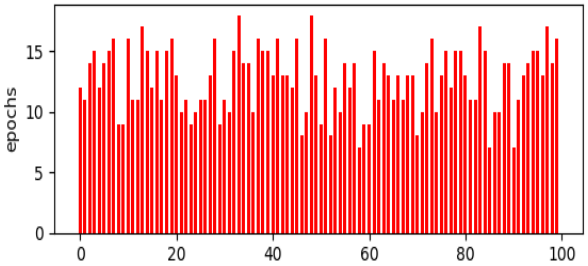
grafikas 6 Q algoritmo testavimo rezultatai priklausomai nuo iteracijos



grafikas 7 SARSA ir Q algoritmo testavimo taškų palyginimas po 8000 iteracijų



grafikas 8 Sarsa algoritmo ėjimų skaičius testavimo metu



grafikas 9 Q algoritmo ėjimų skaičius testavimo metu

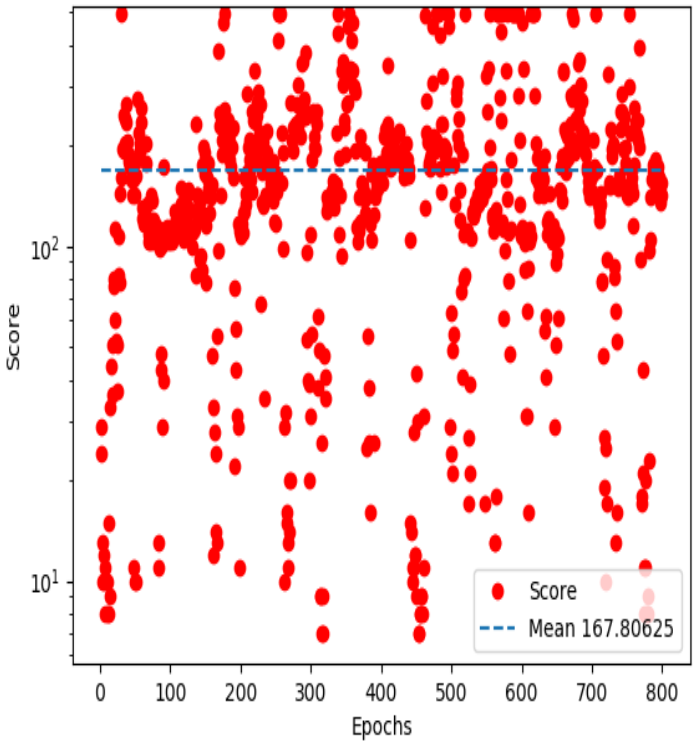
5.2.3 Cartpole-v0 ir v1 rezultatai

Agento tikslas „Cartpole“ aplinkoje (pav. 13) kuo ilgiau laikyti stulpą vertikalioje pozicijoje. Agento būseną apibrėžia 4 skaičiai ir aplinkoje yra galimi tik 2 veiksmai:

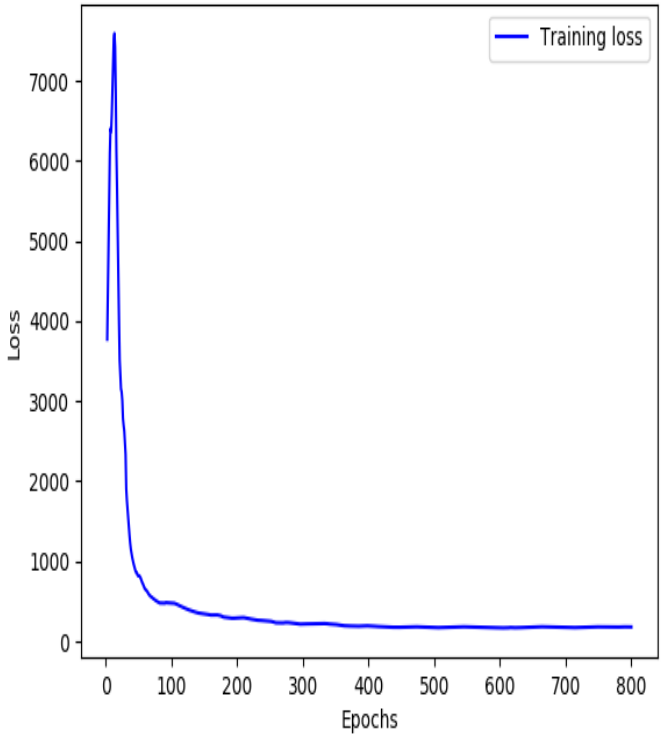
„Cartpole“ agento būsena			
0	Vežimėlio pozicija	-2.4	2.4
1	Vežimėlio greitis	-∞	+∞
2	Stulpo ir vežimėlio kampas	~ -41.8°	~ 41.8°
3	Stulpo viršutinis kampas	-∞	+∞

„Cartpole“ agento veiksmai	
0	Stumti į kairę
1	Stumti į dešinę

Iš esmės, agentas bando išgyventi aplinkoje ir pasiekti aukščiausią taškų kiekį, todėl pagal nutylėjimą kiekvienas žaidimo kadras duoda +1 atpildą. „Cartpole-v0“ aplinkoje maksimalus taškų kiekis yra 200, o „Cartpole-v1“ yra 500. Buvo realizuotas gilus Q mokymo algoritmas (pav. 7) „OpenAI“ aplinkoje, kuris apmoka agentą spręsti „Cartpole“ uždavinį. Numatyta atpildo schema buvo modifikuota taip, kad už kiekvieną nesėkmingą bandymą agentas gauna 10 taškų baudą. Grafikai 10, 11, 12 parodo gilaus Q mokymo algoritmo (pav. 13) rezultatus Cartpole-v1 aplinkoje, kur buvo panaudoti du neuroniniai tinklai mokymo stabilizavimui (pav. 10). Iš 11 grafiko galima pamatyti, kad klaidos funkcija iš pradžių didėja. Tai yra dėl to, kad agentas mokymo metu gauna naują informaciją apie aplinką ir neuroninis tinklas dar negali tiksliai apskaičiuoti koeficientų reikšmę, nes agentas pastoviai keičia ir pildo buferį. Klaidos funkcijos reikšmės taip nešokinėja, kaip vieno tinklo atveju. Rezultate gavau dirbtinį agentą, kuris žaidžia (pav. 13) žaidimą ir bando laikyti stulpą vertikaliai. Mokymo metu vidutinis atpildas buvo 167.8 (grafikas 10), o testavimo metu reikšmės buvo tarp 130 – 155 (grafikas 12). Neuroninio tinko

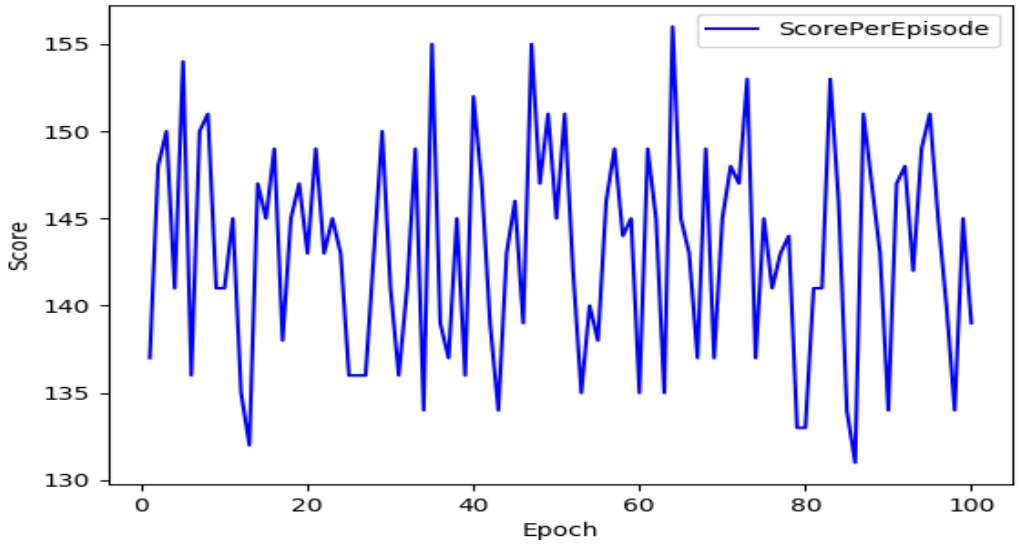


grafikas 10 Mokymo taškų skaičius priklausomai nuo epochos



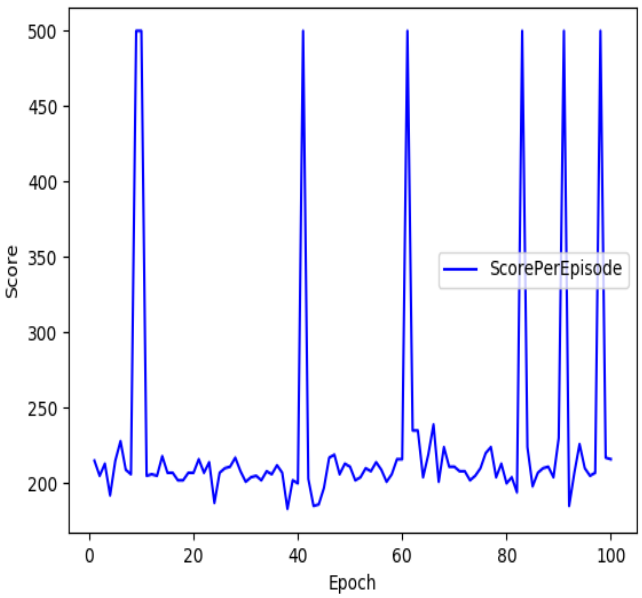
grafikas 11 Pagrindinio tinklo „mse“ klaidos funkcija

konfigūracija: 4 skaičiai kaip įvestis, 3 sluoksniai po 12 neuronų su „relu“ aktyvacijos funkcija, „mse“ klaidos funkcija (grafikas 11) ir „Adam“ optimizavimo algoritmu. Paskutinis sluoksnis apibrėžia veiksmus todėl susideda iš dviejų neuronų. Visas mokymo procesas truko 9 val. ir mokymo proceso pakartojimas duoda beveik panašius rezultatus, bet jeigu sumažinti papildomo neuroninio tinklo sinchronizavimo žingsnį iki 2 epizodų, tai mokymo rezultate agentas gali iš viso nesurasti sprendimo.

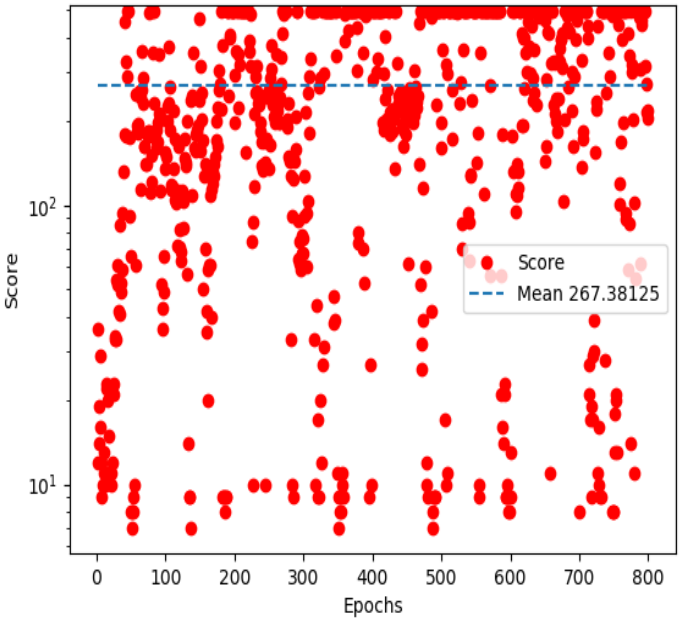


grafikas 12: Apmokyto modelio testavimo rezultatas

Rezultate gavau agentą, kuris žaidžia žaidimą, bet ne su optimaliausia strategija. Pastiprinto mokymo parametrai „CartPole“ aplinkoje: 800 epochų, $\alpha = 0.01$, $\gamma = 0.95$, buferio didis 2000, 32 atsitiktiniai elementai yra ištraukiami iš buferio. Eksperimento metu suformulavau hipotezę, kad agentui neužtenka naudoti tokią atpildo schemą optimaliausiai strategijai rasti, nes agentas tiesiog gauna +1 už sėkmingą stulpo balansavimą ir -10 kai pralaimi žaidimą, bet nėra schemoje papildomos informacijos apie ilgalaikį balansavimą. Todėl nusprendžiau patobulinti atpildo

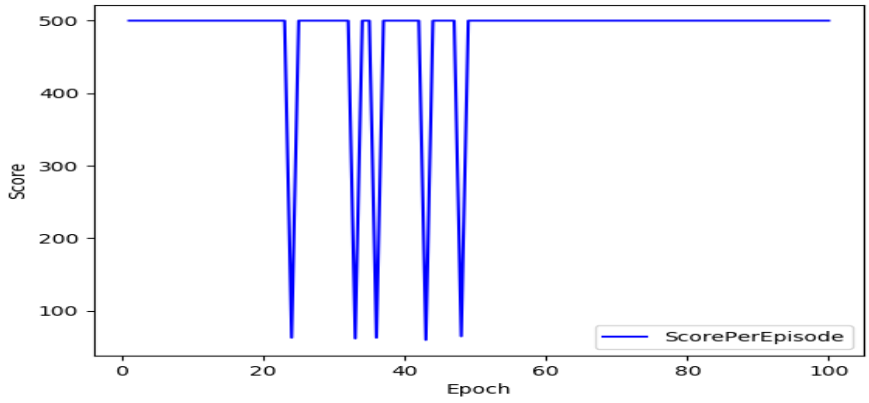


grafikas 13 Apmokyto modelio testavimo rezultatas naudojant naują atpildo schemą

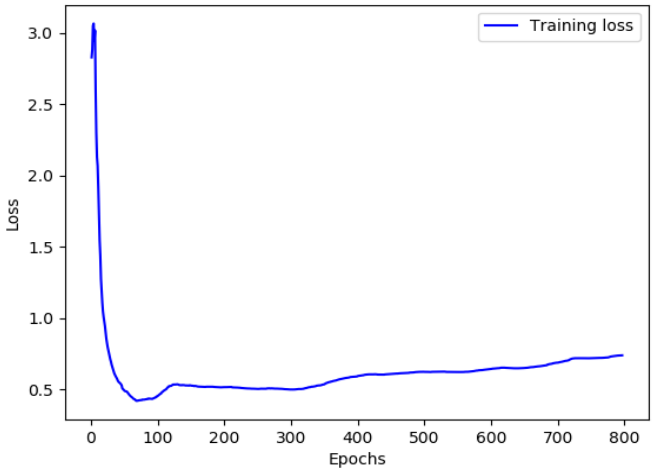


grafikas 14 Mokymo metu gauti taškai

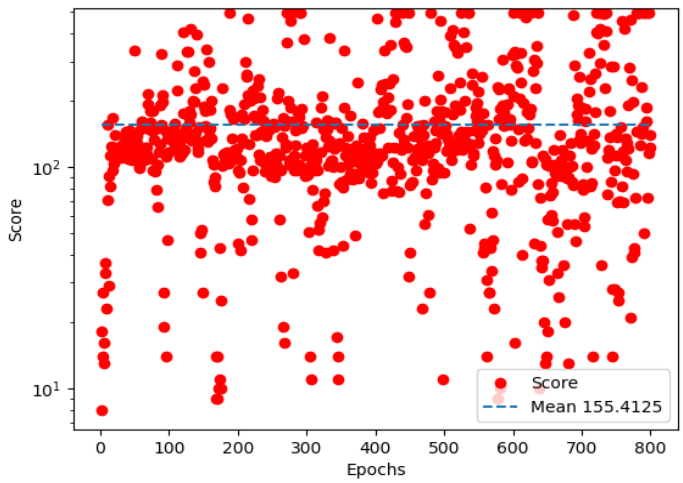
schemą ir patikrinti rezultatą. Jeigu agentas perėjo į terminalinę būseną, tai $r = time - maxTime$, kur $time$ pasako žaidimo kadro numerį ir $maxTime = 500$ yra maksimalus kadrų skaičius ir agento tikslas. O jei agentas yra ne terminalinėje būsenoje, tai $r = r * time$. Rezultate gauname atpildo schemą, kur agentas gauna neigiamą atpildą, kai pralaimi žaidimą $[0..-500]$ ir gauna teigiamą rezultatą kiekvieno kadro metu $[0..+500]$. 13 grafike yra parodytas tokio modelio testavimo rezultatas. Išeina, kad agentas patobulino žaidimo strategiją ir gauna taškus virš 200 netgi mokymo metu (grafikas 14) ir tai reiškia, kad rastas sprendimas aplinkoje „Cartpole-v0“, kur maksimalus skaičius yra 200. Taip pat, agentas sugebėjo iš 100 testavimo epochų penkis kartus pasiekti aukščiausią rezultatą. Toks sukurto modelio modifikavimas pagal hipotezę patobulino rezultatus ir išsprendė pirmą versiją aplinkos, bet tai dar nėra optimali strategija aplinkoje „Cartpole-v1“. Rezultatai parodo, kad agentas išmoko žaisti, bet neatrado optimaliausios strategijos, todėl nusprendžiau padidinti buferį iki 10000 elementų ir analizuojamą poaibį iki 64. Kai agentas gauna daug skirtingų atpildų, mokymas sudėtingėja ir reikalauja daugiau epochų, todėl supaprastinau atpildo schemą. Nusprendžiau panaudoti tik vieną papildomą atpildą, kai agentas pasiekia 499 taškus tai jis papildomai gauna +200 taškus. Toks schemos supaprastinimas leidžia stimuliuoti agentą žaisti daugiau iteracijų ir palengvinti mokymo procesą. Be to, pastebėjau, kad buferis susideda iš panašių duomenų. Iš tikrųjų, kai agentas ima atsitiktinį poaibį iš buferio duomenys iš kažkokio tai epizodo yra šalia vienas kito, todėl pridėjau papildomą atsitiktinį ištrauktų elementų rūšiavimą. Visų išvardintų metodų rezultatas yra pavaizduotas 15 grafike. Agentas išmoko beveik optimaliai žaisti žaidimą ir 95% atvejų agentas sugebėjo pasiekti maksimalus taškus aplinkoje „Cartpole-v1“. Eksperimento metu gavau 155 vidutinį taškų kiekį (grafikas 17), bet testavimo rezultatas yra geriausias iš visų kitų. Tai atsitiko dėl to, kad naudojau papildomą atsitiktinį rūšiavimą, kai ištraukiau duomenis iš buferio. Galų gale, agentas išmoko žaisti beveik optimaliai „Cartpole v0 ir v1“ žaidimą.



grafikas 15 Modifikuoto modelio testavimo rezultatas



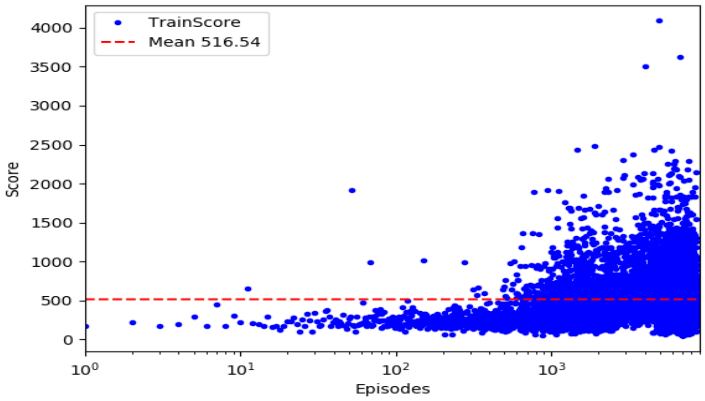
grafikas 16 Modifikuoto modelio klaidos funkcija



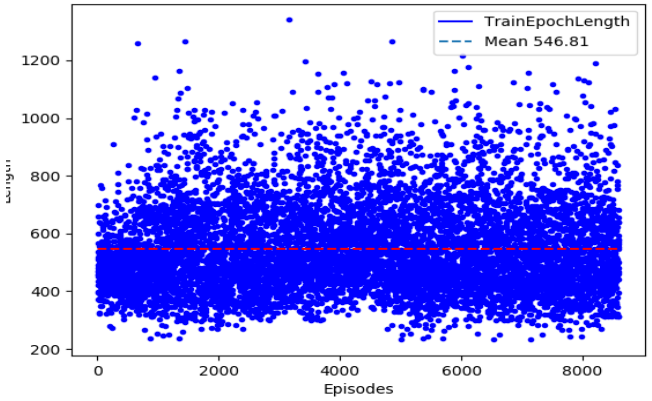
grafikas 17 Modifikuoto modelio mokymo metu gauti taškai

5.2.4 MsPacman-v0

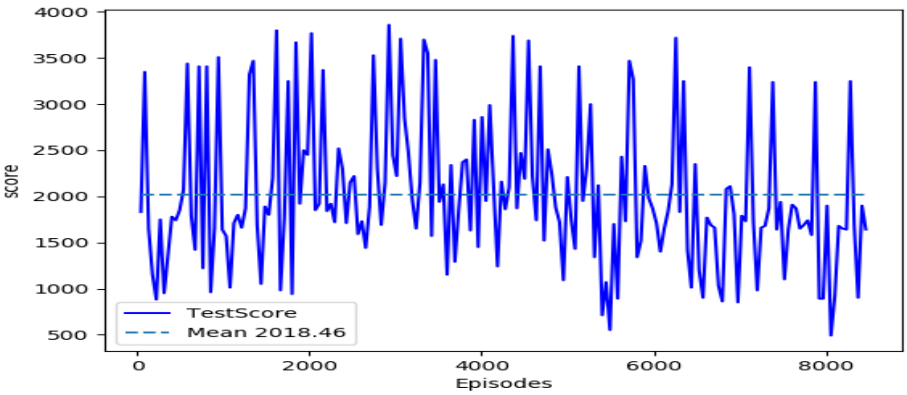
Šios aplinkos sunkumas yra tame, kad agento aplinkos informacija sudaro žaidimo ekrano kopija 210x160x3 (pav. 14). Tokiam uždaviniui spręsti buvo realizuotas dvigubas gilus Q algoritmas (form. 19) ir panaudotas „Google Cloud“ kompiuteris su Tesla P100 GPU. Agentas buvo apmokytas žaisti šioje aplinkoje per 8000 epizodų (grafikas 18 ir 19). Žaidimo ekrano kopijos buvo sumažintos iki 110x80x1 ir paduotos neuroniniam tinklui. Neuroninio tinklo konfigūracija: 3 konvoliuciniai sluoksniai (32x64x64) su ReLu aktyvacijos funkcija, „mse“ klaidos funkcija ir „Adam“ optimizavimo algoritmu. Neuroninio tinklo konfigūracija remiasi [Mks+13]. Buferio dydis 500000 įrašų. Agentas pradeda mokymą po 10000 įrašų buferyje (grafikas 18), nes agento mokymui reikia gauti pakankamai duomenų apie aplinką. Visas mokymas truko 56 val. Vienas aplinkos epizodas yra viena agento gyvybė. Testavimas vyko mokymo metu, po 45 mokymo epizodų agentas buvo testuojamas (grafikas 20). Rezultate gavau agentą, kurio testo taškų vidurkis sudaro 2018 (grafikas 20 ir 21). Tai nėra geriausias įmanomas rezultatas, bet pagal [Hgd15] šioje aplinkoje žmogus dar žaidžia geriau negu agentas apmokytas Q algoritmu, bet [Sfr+17] teigia, kad pavyko rasti optimaliausią sprendimą. Vienas iš tobulinimų būtų padidinti mokymo epizodus ir padidinti patirties buferį, bei perdaryti buferį taip, kad fragmentai būtų rūšiuojami pagal prioritetą, arba panaudoti hibridinę pastiprinto mokymo sistemą [Sfr+17].



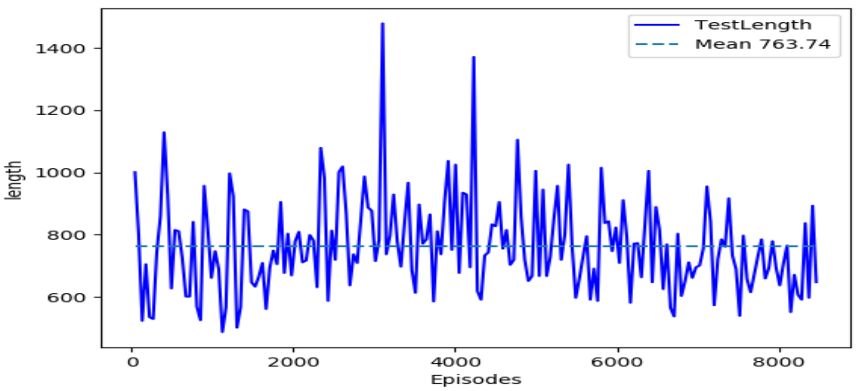
grafikas 18 Taškų skaičius gautas mokymo metu kiekviename epizode



grafikas 19 Žaidimo ilgis kiekviename mokymo epizode



grafikas 20 Agento gauti taškai testavimo metu



grafikas 21 Agento žaidimo ilgis testavimo metu

6 Išvados ir pasiūlymai

Šiame darbe buvo siekiama praktiškai įgyvendinti pastiprinto mokymo Q algoritmus ir rezultate pavyko apmokyti dirbtinį agentą žaisti skirtingose aplinkose beveik optimaliausiu būdu naudojant Q algoritmus. Šiam tikslui pasiekti buvo įvykdytos šios užduotys:

- Darbe atlikta pastiprinto mokymo pagrindų apžvalga. Buvo išvardintos visos pagrindinės sąvokos, apibrėžimai, bei matematinės formulės, nes jie leidžia tiksliai aprašyti pastiprinto mokymo problematiką. Bakalaurinio darbo metu pastiprintas mokymas buvo išnagrinėtas iš teorinės ir praktinės pusės.
- Šiame darbe atlikta pastiprinto mokymo algoritmų analizė. Darbe buvo išanalizuotos pagrindinės pastiprinto mokymo idėjos, metodikos ir algoritmų pritaikymo specifikos. Tai pat, buvo pateikti algoritmų tobulinimo būdai ir naujausi metodai.
- Tikslui pasiekti buvo praktiškai realizuoti pastiprinto mokymo algoritmai skirtingose aplinkose. Bakalaurinio darbo metu Q algoritmai, bei jų modifikacijos buvo realizuoti sukurtoje aplinkoje „Labirintas“, bei naudojant „OpenAI gym“ ir beveik visuose aplinkose agentas išmoko optimaliai sąveikauti su aplinka. Buvo realizuotas gilus Q mokymas „Cartpole-v0-v1“, bei „MsPacman-v0“ aplinkoje.
- Eksperimento metu atlikti pastebėjimai ir suformuluotos hipotezės dėl optimalios pastiprinto mokymo sistemos kiekvienoje aplinkoje.

Bakalaurinio darbo metu buvo suformuluoti pasiūlymai ir pastebėjimai:

- Atpildo schema turi konkrečiai skatinti agentą siekti tikslo, pvz., jeigu tikslas yra pasiekti tašką A, tai agentas būtinai turi papildomai gauti teigiamą atpildą, dėl pasiekto tikslo. Teisingai ir tiksliai sudaryta atpildo schema leidžia patobulinti agento mokymo procesą, bei pagreitinti optimalios taisyklės paiešką.
- SARSA algoritmas konverguos greičiau, jei bus panaudota „Epsilon“ mažinimo strategija.
- Gilaus Q mokyme papildomas duomenų rūšiavimas praktiškai padidina apmokymo rezultatą.
- Gilaus Q mokymo papildomo tinklo sinchronizavimo žingsnis turi būti ne mažiau 4 epizodų.
- Atpildų atvaizdavimas į vieną intervalą nuo [-1..1] leidžia pagreitinti mokymą.
- Agentui reikia leisti ištirti aplinką, todėl mokymas turi prasidėti ne iš karto, o po to, kai patirties buferis bus pakankamai užpildytas. Pasiūlymas būtų įvesti parametrą N ir paleisti mokymą, kai buferio įrašų skaičius viršys nustatytą parametrą. Tai pagreitina geriausios strategijos paiešką.
- Dviguba Q atnaujinimo taisyklė pagreitina konvergavimo procesą.
- Atpildo schema turi būti subalansuota. Per daug neigiamo arba tik teigiamas atpildas neskatina agento ieškoti optimaliausios strategijos.
- Optimaliausias γ parametro intervalas yra $0{,}9 \leq \gamma \leq 0{,}99$.

Apibendrinant reikia pabrėžti, kad pastiprinto mokymo Q algoritmai buvo sėkmingai pritaikyti aplinkoms su Markovo sąlyga. Tolimesnis darbas būtų pritaikyti hibridinius pastiprinto mokymo algoritmus aplinkoms, kur Markovo savybė yra nepatenkinta.

7 Reziume

Reinforcement learning is a very rapidly growing field of study, where a lot of new innovations, state-of-the-art methods and technics are immerging because of never ending challenges for the artificial intelligence. Reinforcement learning opens a new way of looking at the general artificial intelligence and helps to solve problems that other machine learning methods failed to solve. Because of this reasons reinforcement learning is a target subject for this paper.

This paper analyzes essential parts of reinforcement learning methods and practically applies Q learning algorithms to solve different tasks in game like environments. As a result, different Q algorithms where used to successfully train agents in contrasting environments.

8 Šaltiniai

[Adb17] K. Arulkumaran, M.P Deisenroth, M. Brundage and A. A. Bharath. *A Brief Survey of Deep Reinforcement Learning*. IEEE Signal Processing Magazine, Special Issue on Deep Learning for Image Understanding, 2017, pp. 26-38

[Asm03] S. Asmussen. *Applied Probability and Queues*. Springer Science & Business Media, New York, 2003

[Asb08] G. Andrew, T. Sandholm and T. B. Sorensen. A heads-up no-limit Texas Hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, 2008, pp 911-918.

[Bur03] M. Buro. Real-Time Strategy Gaines: A New AI Research Challenge. *ICJAI*, 2003, pp. 1534-1535.

[Dič05] V. Dičiūnas. Algoritmų analizės pagrindai : mokymo priemonė. - URL: http://www.mif.vu.lt/katedros/cs/Asmen/algoritmu_analize.pdf . 2005, 106 p.

[Efr03] B. Efron. Second Thoughts on the Bootstrap. *Statistical Science-Volume 18*, 2003, pp 135 -140

[Gre17] J. Greaves. *Understanding RL: The Bellman Equations*. – URL: https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/. 2017-11-03

[Hau20] M. Hauskrecht. Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research 13*, 2000, pp. 33-94

[Hgd15] H. Hasselt , A. Guez, and D. Silver. *Deep Reinforcement Learning* with *Double* Q-Learning.. Google DeepMind. 2015

[Klm96] L. Kaelbling, M. Littman and A. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research 4*, 1996, pp. 237-285.

[Lap18] M. Lapan. *Deep Reinforcement Learning Hands-On*. Packt Publishing,

Birminghan, 2018, 546 pages

[Mks+13] V.Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wiersta ir M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. Deepmind Technologies. 2013

[Opp18] A. Oppermann. *Self Learning AI-Agents Part II: Deep Q-Learning*. – URL: https://towardsdatascience.com/self-learning-ai-agents-part-ii-deep-q-learning-b5ac60c3f47. 2018-10-28

[Osu13] S. Ontanon, G. Synnaeve and A. Uriarte. A Survey of Real-Time Strategy Game AI Research and Competition in StarCraft, *IEEE Transactions on Computational Intelligence and AI in games, IEEE Computational Intelligence Society*, 2013, pp. 1-19.

[Rnd16] S. Russell, P. Norvig and E. Davis. *Artificial Intelligence: A Modern Approach. 3rd ed*, Prentice Hall, Upper Saddle River, NJ, 2016, pp. 649.

[Pm10] L.D. Poole, K.A. Mackworth. Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press New York, 2010, 653 pages.

[Roz82] Y.A. Rozanov. *Markov Random Fields*. . Springer Science & Business Media, New York, 1982, 196 pages.

[Sb16] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press. Cambridge, Massachusetts, 2016

[Sfr+17] H. Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes and J. Tsang. *Hybrid Reward Architecture*

for Reinforcement Learning. Technical report, Montreal, 2017.

[Sha50] C. E. Shannon. Programming a Computer for Playing Chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1950, pp. 256-275.

[Tes95] G. Tesauro. Temporal diffrence learning and td-gammon. *Communications of the ACM*, 1995, pp 58-68.

[Wd92] C. Watkins, P. Dayan, Q-Learning, *Machine learning*, 1992, pp. 279-292.