Kauno technologijos universitetas

Informatikos fakultetas

Gilusis skatinamasis mokymasis kompiuterinių žaidimų agentų valdymui

Magistro baigiamasis projektas

Žygimantas Šlikas

Projekto autorius

Prof. Dr. Rytis Maskeliūnas

Vadovas

Kaunas, 2025

Kauno technologijos universitetas

Informatikos fakultetas

Gilusis skatinamais mokymasis kompiuterinių žaidimų agentų valdymui

Baigiamasis magistro studijų projektas

Dirbtinio intelekto informatika (6211BX007)

Žygimantas Šlikas

Projekto autorius

Prof. Dr. Rytis Maskeliūnas

Vadovas

Prof. Dr. Tomas Blažauskas

Recenzentas

Kaunas, 2025

Kauno technologijos universitetas

Informatikos fakultetas

Žygimantas Šlikas

Gilusis skatinamasis mokymasis kompiuterinių žaidimų agentų valdymui

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs;

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Žygimantas Šlikas

Patvirtinta elektroniniu būdu

Šlikas, Žygimantas. Gilusis skatinamais mokymasis kompiuterinių žaidimų agentų valdymui. Magistro baigiamasis projektas / vadovas prof. dr. Rytis Maskeliūnas; Kauno technologijos universitetas, informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): informatikos mokslai, Informatika (B01)

Reikšminiai žodžiai: Skatinamasis mokymas, neuroniniai tinklai, VAE, kompiuteriniai žaidimai.

Kaunas, 2025. 62 p.

Santrauka

Norint dirbtinį intelektą pritaikyti kuo įvairesniam spektrui problemų, reikia, kad jis galėtų: pats tyrinėti aplinką, išmokti optimaliausius veiksmus konkrečiose situacijose ir gebėtų planuoti kuo daugiau veiksmų į priekį. Šio tipo užduotims spręsti naudojami skatinamojo mokymosi tipo algoritmai, o ypač paskutiniu metu, ir neuroniniais tinklais grįsti jų tipai. Viena pagrindinių jų problemų yra didelis treniravimo duomenų kiekio poreikis. Šiame tyrime buvo atlikti aksperimentai su populiaraus algoritmo PPO modifikacijomis ir siekėme sumažinti reikiamų duomenų kiekį arba pasiekti geresnius rezultatus su tuo pačiu kiekiu. Bandymai buvo atlikti su 3, skirtingų savybių aplinkomis – kompiuteriniais žaidimais „Ms Pacman“, „VizDoom Health gathering“ ir „VizDoom Deffend the center“. Buvo pasiūlyti 2 atskiras modifikacijas: paskirstytos strategijos funkcija, ir VAE grįstas vaizdo suspaudimas. Paskirstyta strategijos funkcija, kuri remiasi idėjomis iš ansamblinių mašininio mokymosi metodų, tokių kaip atsitiktinis miškas. Paskirstyta strategijos funkcijos atšakos treniravimo metu įgyja skirtingus atsitiktinius svorius, tokiu būdu agentas išmoksta įvairesnę bendrą strategiją. Viename iš žaidimų geriausia variacija per epizodą vidutiniškai surinko po 1518 taškus, o nemodifikuotas algoritmas surinko 1377 taškus, kas rodo ~10% pranašumą. Visos algoritmo variacijos turėjo mokymosi nestabilumų ir konverguodavo į geresnius arba blogesnius rezultatus, tačiau modifikuoto algoritmo rezultatų dispersija pasirodė daug didesnė. Pasiūlytas algoritmas geriausiu bandymu surinko 2046 taškus, o nemodifikuotas 1666, kas atitinka ~23% geresnius rezultatus. Kita modifikacija tyrinėjo kaip galima panaudoti VAE kaip potencialiai greitesnį, neuroninio tinklo treniravimo metodą, kai jis panaudojamas vaizdo suspaudimui. Tai turėjo padėti efektyviau panaudoti duomenis, treniruojant su jais pakartotinai, tačiau atlikus įvairius integravimo bandymus pasirodė, kad VAE ir PPO treniravimas vienu metu yra gana nestabilus ir nedavė geresnių rezultatų nei pradinis algoritmas.

Šlikas, Žygimantas. Deep reinforcement learning for computer games agents' control. Master's Final Degree Project / supervisor prof. dr. Rytis Maskeliūnas; Faculty of Informatics, Kaunas University of Technology.

Study field and area (study field group): Computer science, Informatics (B01).

Keywords: Reinforcement learning, neural networks, VAE, computer games.

Kaunas, 2025. 62.

Summary

To apply artificial intelligence to a wider range of problems, it must be able to: explore its environment independently, learn the most optimal actions in specific situations, and be capable of planning several steps ahead. To solve such tasks, reinforcement learning algorithms are used and recently, neural networks–based ones. High demand for training data is one of the main issues. In this study, we conducted experiments with modifications of the popular PPO algorithm and aimed either to reduce the amount of data required or to achieve better results with the same amount of data. The experiments were carried out in three environments with different characteristics — computer games: "Ms. Pacman", "VizDoom Health Gathering", and "VizDoom Defend the Center". We proposed two separate modifications: a distributed policy function and VAE-based image compression. The distributed policy function is based on ideas from ensemble machine learning methods such as random forests. During training, the branches of the distributed policy function acquire different random weights, allowing the agent to learn a more diverse overall strategy. In one of the games, the best variation achieved an average of 1,518 points per episode, while the unmodified algorithm scored 1,377, indicating ~10% improvement. All algorithm variations had training instabilities and converged to either better or worse results, but the modified algorithm showed a much higher variance in results. In the best trial, our modified algorithm scored 2,046 points, while the unmodified one scored 1,666, which corresponds to ~23% better performance. In the other modification, we explored how a VAE could be used as a potentially faster neural network training method when applied to image compression. This was intended to help us use the data more efficiently by enabling repeated training on it. However, after various integration experiments, it turned out that training the VAE and PPO simultaneously was quite unstable and did not yield better results than the original algorithm.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

RL – (ang. Reinforcement learning) skatinamasis mokymasis.

MDP – (ang. Markov decision process) Markovo sprendimo procesas.

CNN – Konvoliucinis neuroninis tinklas.

LSTM – (ang. Long short term memory) rekurentinis neuroninio tinklo sluoksnis leidžiantis prisiminti informaciją iš senesnių tinklo įvesčių.

GRU – (ang. Gated recurrent unit) naujesnė rekurentinė neuroninio tinklo sluoksnio versija leidžianti prisiminti informaciją iš senesnių tinklo įvesčių.

PPO – (ang. Proximal policy optimization) Strategijos funkcijos konservatyviu optimizavimu grįstas RL algoritmas.

CPU – kompiuterio centrinis procesorius.

GPU – Grafikos apdorojimo įrenginys.

RAM – Kompiuterio darbinė atmintis.

CUDA – GPU programavimo biblioteka.

ReLU – (ang. Rectified linear unit) Netiesiškumo sutiekianti, neuroninio tinklo aktyvacijos funkcija.

SELU – (ang. Scaled exponential linear unit) Netiesiškumo sutiekianti, neuroninio tinklo aktyvacijos funkcija.

VAE – (ang. Variational autoencoder) Neuroninio tinklo architektūros tipas

API – (ang. Application programing interface) sąsaja skirta komunikavimui su tam tikra programa.

Terminai:

Q vertės funkcija – Skatinamojo mokymosi metodas kai algoritmas periodiškai patikslina būsenos–veiksmo vertės funkciją. Sprendimai yra priimami palyginant skirtingų veiksmų įverčius.

Skatinamasis mokymasis – Mašininio mokymosi poskyris, kuriame kompiuterio valdomas agentas sąveikauja su tam tikra aplinka, iš kurios gauna duomenis (būsenas) ir įverčius.

Autoenkoderis – neuroninio tinklo architektūros tipas, skirtas informacijos suspaudimui.

Hiperparametrai – mašininio mokymosi algoritmo parametrai kuriuos koreguoja algoritmo naudotojas.

Įvadas

Pastaraisiais metais itin sparčiai vystosi ir populiarėja dirbtinio intelekto ir mašininio mokymosi sritys. Ši nuo seno populiari kompiuterių mokslo šaka siekia kurti vis pažangesnes sistemas, kurios kuo labiau priartėtų ar pranoktų žmogaus galimybes priimti sprendimus ir atlikti užduotis realiame pasaulyje. Kaip galutinį tikslą galima įvardinti galimybę automatizuoti visas žmonių atliekamas kasdienes ir pramonines veiklas bei gebėjimą algoritmams patiems vystytis ir tobulėti.

Kaip geriausia ir plačiausiai naudojama strategija pripažįstama mašininiu mokymusi grįstas problemos sprendimas, nes jis geba nuolat keistis ir prisitaikyti prie kintančių sąlygų, bei išmokti apdoroti gaunamus duomenis iš įvairių jutiklių, kurie renka informaciją apie aplinkinį pasaulį ir gali turėti itin didelį srautą triukšmingos ir sunkiai interpretuojamos informacijos.

Mašininio mokymosi sritis, pagal tai kokio tipo problemas siekia spręsti, dažnai išskiriama į tris kategorijas: prižiūrimasis mokymasis, neprižiūrimasis mokymasis ir skatinamasis mokymasis. Pirmieji du tipai sprendžia problemas kurios yra statinio tipo, t. y. gavus įvesties reikšmes, algoritmas grąžina atitinkamą rezultatą. Šios kryptys yra sparčiai plėtojamos, nes yra plačiai taikomos srityse kur turimi duomenų rinkiniai ir norima susieti tam tikrus duomenų atributus, bet nežinoma juos jungianti funkcija. Tačiau, nors ir ypač naudingi duomenims apdoroti, šie algoritmai kompiuteriams nepadeda atlikti valdymo užduočių, kur reikia generuoti ilgas instrukcijų sekas, atlikti planavimą ir mokintis neturint išankstinio duomenų rinkinio. Pavyzdžiui robotų valdymas arba sudėtingų sistemų optimizavimas. Šio tipo uždaviniams spręsti naudojami skatinamojo mokymosi algoritmai.

Esminiai skiriamieji skatinamojo mokymosi algoritmų bruožai yra mokymasis iš realiu laiku generuojamų rezultatų, kurie gaunami algoritmo kontroliuojamam agentui atliekant veiksmus realioje arba virtualioje aplinkoje. Taip pat, algoritmas, norėdamas pasiekti tikslą, dažniausiai turi grąžinti ne vieną teisingą atsakymą, o seką veiksmų, atsižvelgiant į gaunamą informaciją apie aplinką. Taip pat dažnai pasitaikantys iššūkiai gali būti: ne visa aplinkos informacija (agentas žino tik tai, ką rodo jo jutikliai), stochastiniai aplinkos pokyčiai (aplinka gali keistis nepriklausomai nuo algoritmo veiksmų), kiti agentai aplinkoje.

Tokių algoritmų treniravimas užtrunka daug laiko ir ribojančiu veiksniu dažnai tampa agento atliekamų instrukcijų ir gauto grįžtamojo ryšio, aplinkos pokyčio informacijos kiekis. Taip pat, kuriant algoritmus, norima, kad jie vėliau galėtų būti panaudoti kuo platesniam problemų spektrui. Dėl šių priežasčių skatinamojo mokymosi algoritmai apmokomi virtualiose aplinkose arba kompiuteriniuose žaidimuose. Žaidimų ir simuliuotų aplinkų naudojimas turi daug naudingų savybių: algoritmų treniravimą galima paspartinti atliekant treniravimą keliose lygiagrečiai veikiančiose aplinkose, žaidimai gali turėti įvairias užduotis ir būti skirtingo sudėtingumo, reikalauti skirtingų įgūdžių. Tai naudinga siekiant, kad algoritmas būtų kuo universalesnis. Dar vienas naudingas žaidimų aspektas yra tai, kad dauguma turi žaidėjo surinktų taškų sistemą, kuri gali būti panaudota agento treniravimui, bei siekiant tarpusavyje palyginti skirtingų algoritmų efektyvumą (pvz. literatūroje dažnai naudojamas 50 *Atari* žaidimų rinkinys – pagal juose surenkamus taškus galima palyginti, kuris algoritmas pasiekia geresnių rezultatų).

Tikslas
patobulinti jau egzistuojančio giliojo skatinamojo mokymosi algoritmo efektyvumą duomenų kiekio atžvilgiu. Modifikuota algoritmo versija turėtų pasieki geresnių rezultatų su tokiu pačiu kiekiu žaidimo imitacijos, arba pasiekti tuos pačius rezultatus imitavus mažesnį kiekį žaidimo kadrų.

Uždaviniai:

1. pasirinkti tinkamus įrankius neuroniniams tinklams realizuoti ir kompiuteriniams agentams mokyti; pademonstruoti egzistuojančio algoritmo veikimą bei sukurti įrankius veikiančiam algoritmui demonstruoti;

2. remiantis inovacijomis mašininio mokymosi srityje, pasirinkti modifikacijas kurios gali suamžinti treniravimo duomenų poreikį;

3. eksperimentiškai išmatuoti modifikacijų poveikį algoritmo duomenų efektyvumui, ir surasti optimalius hiperparametrus.

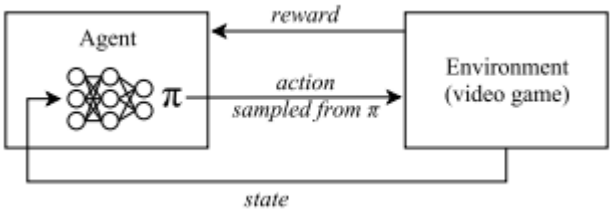
Mokslinis naujumas.

Šiame tyrime buvo pasiūlyta ir išbandėme 2 naujus giliojo skatinamojo mokymosi metodus, kuriais siekiame sumažinti treniravimui reikiamų duomenų kiekį. Pirmoji yra paskirstyta strategijos funkcija, kuri remiasi idėjomis iš ansamblinių mašininio mokymosi metodų, tokių kaip atsitiktinis miškas. Paskirstytos strategijos funkcijos atšakos treniravimo metu įgyja skirtingus atsitiktinius svorius, tokiu būdu agentas išmoksta įvairesnę bendrą strategiją ir kaip rodo atlikti eksperimentai pasiekia geresnių rezultatų. Kiek žinoma, ši modifikacija anksčiau nebuvo tyrinėta. Antroji tyrinėta modifikacija remiasi variaciniu autoenkoderiu, skirtu vaizdui suspausti. Panašūs metodai jau yra aprašyti literatūroje, bet juose VAE yra naudojamas kaip pasaulio modelis ir skirtas nuspėti ateities aplinkos būsenas. Šiuo atveju jis panaudojamas kaip potencialiai greitesnis, neuroninio tinklo treniravimo metodas, nes jam netaikomi ribojantys duomenų reikalavimai. Buvo atlikti įvairius bandymus kaip būtų galima treniruoti skatinamojo mokymosi algoritmą kai dalį jo sudaro VAE ir ar tai paspartintų treniravimą.

1. Skatinamojo mokymosi literatūros analizė

1.1. Skatinamojo mokymosi problemų ir algoritmų aprašymo būdai

Skatinamasis mokymasis gali būti apibūdinamas kaip dinaminė sistema kurioje agentas (algoritmo ar žaidėjo valdomas objektas) sąveikauja su aplinka. Sąveika gali būti apibūdinta kaip ciklinis apsikeitimas informacija, kuriame agentas atlieka veiksmą, aplinka sureaguoja į agento veiksmus ir pasikeičia pagal ją valdančius veiksnius. Galiausiai agentas stebi, kokia yra nauja aplinkos būsena ir kokį atlygį jis gavo, pavyzdinė diagrama matoma **1.1 pav**. Atlygis gali būti ir konkretūs veiksmai arba daiktai surinkti virtualioje aplinkoje, arba tai gali būti dirbtinai sudaryta išvestinė funkcija kuria skatina algoritmą išmokti norimų savybių. Atlygio funkcija reikalinga kad algoritmas žinotų kokie rezultatai yra geidžiami ir ko reikia vengti. Dažniau duodamas grįžtamasi ryšys padeda agentams greičiau mokintis ir lengviau susieti veiksmus su teigiamais rezultatais, tačiau galimos situacijos kai atlygio reikšmė gaunama tik paskutiniame etape, pavyzdžiui šachmatų žaidimas – rezultatas žinomas tik užbaigus partiją.



1.1 pav. Skatinamojo mokymosi agento ir aplinkos sąveikos ciklas [1]

Sistemos, kurios turi daug skirtingų būsenų ir jų pasikeitimų sąlygų, su skirtingomis tikimybėmis, teorijoje dažnai apibūdinamos kaip Markovo grandinės. Su šiuo modeliu taip pat susijęs Markovo sprendimo procesas (ang. MDP). Tai yra matematinis modelis kuris aprašo kaip kiekvienas iš galimų atsitiktinių įvykių veikia sistemos perėjimus į skirtingas būsenas. Markovo procesu grįsti problemos sprendimai daro prielaidą kad visa reikalinga informacija yra saugoma būsenos apraše, ir agentu nereikia prisiminti ankstesnių būsenų istorijos. Yra įrodyta, [2] kad jai problemą galima aprašyti MDP, tuomet klasikiniai skatinamojo mokymosi algoritmai tokie kaip *Q–learning* gali surasti optimalų sprendinį.

Dėl to dauguma pirmųjų giliuoju mokymusi grįstų skatinamojo mokymosi sprendimų bandė atkartoti MDP savybes [3], [4]. Buvo stengiamasi kad įvestys kiekviename žingsnyje turėtų visą reikiamą informaciją sprendimui priimti, o neuroninio tinklo architektūra turėjo imituoti *Q–learning* metodo naudingumo matricą. Taip buvo siekiama užtikrinti kad tinklas treniravimo metu turėtų nuolat mažėjančią nuostolio (ang. *loss*) reikšmę, ir galiausiai konverguotų prie optimalios strategijos [1].

Skatinamojo mokymosi agentų veikimo cikle veikia keletas svarbių funkcijų kurios nusprendžia kokius veiksmus pasirinks atlikti algoritmas, svarbiausios iš jų yra būsenos ir atlygio, bei veiksmo pasirinkimo funkcijos. Visi giliuoju mokymusi pagrįsti algoritmai savo neuroniniuose tinkluose turi atskiras atšakas kurios aproksimuoja vieną ar kelias iš toliau pateiktų funkcijų.

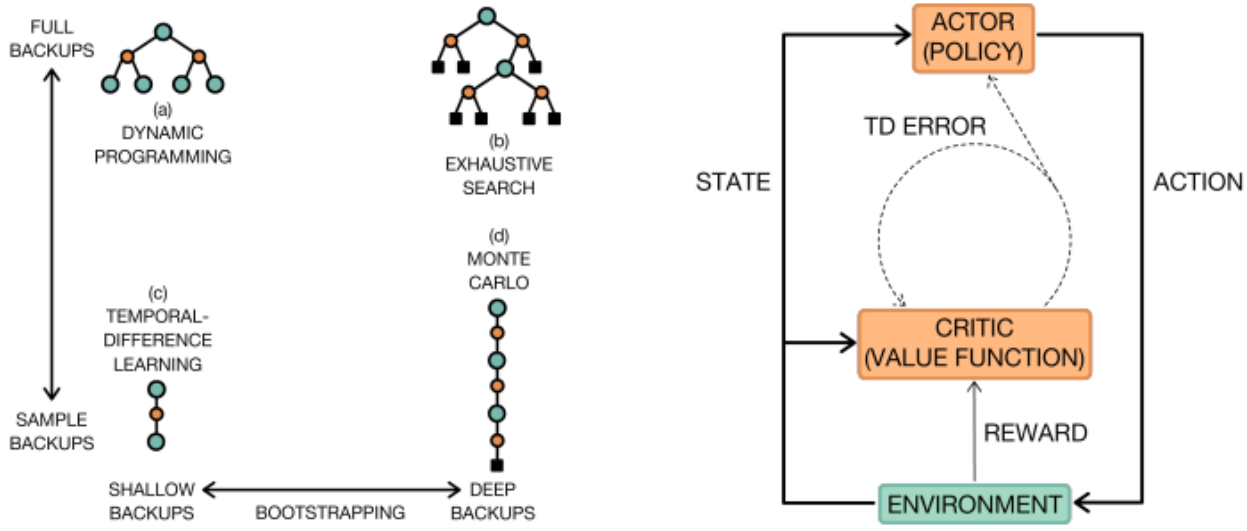
1. Strategija (ang. *Policy*) – susieja agento būseną su būsimu agento veiksmu. $\pi(a, s) = \Pr(a_x = a | s_x = \mathrm{s})$, čia a yra veiksmas, o s būsena. Strategijos funkcija žymima raide π ir grąžina tikimybę kuri nurodo kiek šio veiksmo pasirinkimas yra geresnis, už kitus.

2. Būsenos vertės funkcija – $V_\pi(s) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t \,|s_0 = s]$, čia γ yra nuvertėjimo reikšmė, ir s – dabartinė būsena, r žymi atlygį gautą kiekvieno žingsnio metu. Nevertėjimo parametras nurodo kiek algoritmas teikia prioritetą artimiems atlygiams palyginus su esančiais tolimoje ateityje. V funkcija parodo, kaip gerai yra agentui būti pozicijoje *s*, atsižvelgiant į tai kad sekantys veiksmai yra pasirenkami pagal strategiją π.

3. Būsenos–veiksmo vertės funkcija (Q matrica) – $Q_\pi(s,a) = \mathrm{E}[R|s,a,\pi]$ ši funkcija grąžina naudingumo reikšmes visoms būsenos ir veiksmo poroms. Dažnai naudojamas šios funkcijos variantas yra *Q–learning* algoritmas kuris neturi kintančios strategijos ir visada pasirenka maksimalaus atlygio veiksmą. Tai leidžia surasti optimalų agento valgymo planą, tik jei žinomas visas MDP. Taip pat šios funkcijos sąryšis su auksčiau aprašytomis gali būti išreikštas $\mathrm{V} = \pi * Q$. Tačiau keletas iš šios funkcijos trūkumų yra tai, kad būsenų ir veiksmų kombinacijų skaičius auga kvadratiškai ir nėra tinkamas didesnėms problemoms. Taip pat atlygio reikšmės yra išmokstamos kai veikiama pagal nustatytą strategiją, algoritmas negali pats pakeisti strategijos funkcijos.

1.2. Svarbiausi algoritmų architektūrų tipai

Žinant, kad yra keletas būdų, kaip aprašyti skatinamojo mokymosi problemas ir sprendinių optimalumą, nenuostabu, kad praktikoje egzistuoja keletas skirtingais principais veikiančių algoritmų. Skatinamojo mokymosi algoritmai renkantis veiksmus gali naudoti: būsenos vertės funkciją, dinaminio programavimo arba laikino skirtumo, istorinių duomenų vidurkio, tiesioginio strategijos tobulinimo, ir aktoriaus–kritiko metodus. Algoritmų tipų skirtumai pavaizduoti **1.2 pav**.



1.2 pav. Dešinėje – vertės funkcijų aproksimavimo būdai; kairėje – aktoriaus–kritiko algoritmo struktūra [5]

1.2.1. Vertės funkcijos

Vertės funkcijomis pagrįstiems algoritmams galima priskirti tuos metodus, kurie naudoja būsenos naudingumo, būsenos veiksmo naudingumo (Q–funkcija, SARSA) ir pranašumo (ang. *advantage*) funkcijas. Šiuo metu sėkmingiausiai iš vertės funkcijų yra taikomas pranašumo metodas kuris susieja būsenos–veiksmo ir būsenos vertės funkcijas pagal formulę $A^\pi = Q^\pi - V$, čia A nurodo

kiekvieno veiksmo naudingumą lyginat su kitais veiksmais, galimais konkrečioje būsenoje. Šios funkcijos idėja yra sumažinti reikšmių diapazoną (ang. variance) ir taip palengvinti mokymosi užduotį. Treniravimo metu vertės funkcijos ir ypač Q–funkcijos reikšmės gali būti apskaičiuojamos naudojant dinaminį programavimą. Numatoma, kad būsenų perėjimai atitinka Markovo savybę ir priklauso tik nuo dabartinės būsenos, o ne nuo praeities. Tai leidžia Q reikšmes išreikšti pagal Bellmano lygtį [4] ir atnaujinti iteraciniu vidurkio patikslinimu (ang. *bootstrapping*), kai reikšmės patikslinamos pagal formulę $Q^{\pi}(s_t, a_t) = Q^{\pi}(s_{t+1}, a_{t+1}) +$, čia α yra mokymosi greitis < 1, o $\delta = Y - Q^{\pi}(s_t, a_t)$ laikinasis skirtumas (ang. Temporal difference), Y reiškia toje iteracijoje gautą atlygį. TD metodu besiremiantys algoritmai gali atnaujinti savo vertės funkcijas kiekviename žingsnyje, kai tik gauna naują atlygio reikšmę [5].

Kitas funkcijų aproksimavimo būdas yra naudojant Monte Carlo metodus [5], kurie remiasi keleto atsitiktinai sužaistų partijų taškų vidurkiu, kaip būsenos naudingumo įverčiu. Pagrindinis skirtumas nuo laikinojo skirtumo ir dinaminio programavimo metodų yra tas, kad sužaidžiama keletas ėjimų į priekį kol pasiekiama pabaiga ar gaunamas koks nors atlygis. Tai yra ir vienas iš šių strategijų trūkumų, nes ne visos aplinkos turi aiškią ar greitai pasiekiamą pabaigą dėl to gali būti sunku gauti rezultatus net vienam pavyzdžiui. Pranašumas yra tas, kad metodas nesiremia ankstesnių būsenų įverčiais ir gali būti naudojamas užduotims kur aplinka neatitinka Markovo taisyklių, pavyzdžiui, aplinkos, kuriose yra daug triukšmo ar nuo išorės veiksnių priklausančių pokyčių [5].

1.2.2. Strategijos funkcijos

Skatinamojo mokymosi algoritmai taip pat gali priimti sprendimus tiesiogiai iš gautų įvesties duomenų, nenaudodami būsenos ar veiksmo įverčio funkcijų. Tam pasitelkiant strategijos funkcijas kurios kaip rezultatus grąžina tikimybių pasiskirstymus kaip naudinga yra pasirinkti kiekvieną iš galimų veiksmų. Šios funkcijos gali būti aproksimuotos naudojant neuroninius tinklus su daug parametrų, ir dažniausiai yra mokomos naudojant gradientu grįstus optimizavimo metodus. Algoritmo tikslas yra pasiekti kuo geresnį galutinį rezultatą, pasirenkant optimalią veiksmų seką ir turint duomenis apie dabartinę būklę. Šis problemos apibrėžimas nėra labai apribojantis, todėl strategijos optimizavimo algoritmai gali būti taikomi plačiam užduočių spektrui. Strategijos funkcija taip pat gali būti taikoma ir užduotims su tolydžiais rezultatais, nes veiksmai gali būti grąžinami kaip Gauso pasiskirstymo vidurkis ir standartinis nuokrypis. Treniruojant strategijos funkcijos neuroninius tinklus reikalingas skaitinis įvertis kuris leistų optimizuoti tinklą ir pasirinkti geresnius veiksmus. Veiksmų naudingumo įverčiai gali būti gauti deterministiniu arba stochastiniu metodu. Deterministiniam metodui reikalinga turėti aplinkos modelį, kuris žino, kokios yra aplinkos būsenų pasikeitimo taisyklės ir tikimybės. Naudojant aplinkos modelį ir strategijos funkciją galima apskaičiuoti visus įmanomus rezultatus ir jų tikėtinumus kai pradedama iš tam tikros padėties, tai leidžia tiksliai apskaičiuoti visų būsenų ir veiksmų naudingumą. Dauguma skatinamojo mokymosi metodų nenaudoja pasaulio medelio, todėl dažniau naudojamas stochastinis, veiksmo arba būsenos vertės aproksimavimas. Tačiau rezultatai gauti skaičiuojant vidutines reikšmes gautas naudojant Monte Carlo metodą, nėra tinkami gradientui skaičiuoti, nes atsitiktiniai pasirinkti veiksmai neturi gradiento. Ši problema sprendžiama naudojant REINFORCE taisyklę $\theta_{(t+1)} = \theta_t + \alpha \nabla_{\theta} log\, \pi_{\theta}(a_t|s_t) \Sigma_{k=t}^{T} \gamma^{k-t} R_t$, kuri leidžia apskaičiuoti strategijos gradientą kai turima keletas stochastinių veiksmų sekų su rezultatais. Gradientai yra apskaičiuojami iš empiriškai surinktų rezultatų, todėl dažnai yra nepatikimi, skiriasi tarp atskirų simuliacijų. Problemą galima sumažinti

vietoje tiesioginio galutinės būsenos įverčio naudojant pranašumo funkciją, kuri atima bendrą kiekvienos būsenos įvertį ir palieka tik skirtumus tarp skirtingų veiksmų [5].

1.2.3. Aktoriaus–Kritiko metodai

Trečias, ir šiuo metu vienas populiariausių yra vadinamas aktoriaus–kritiko skatinamojo mokymosi sprendimas. Jis naudoja 2 funkcijas: strategijos (aktoriaus) ir kritiko (būsenos vertės), kaip galima matyti **1.2 pav**. Metodas naudoja 2 atskiras funkcijas kad treniravimo metu funkcijos galėtų patobulinti viena kitą. Kadangi naudojami veikimo metodai tiek iš vertės, tiek iš strategijos iteravimo funkcijų, šio tipo sprendimai gali panaudoti inovacijas iš abiejų krypčių. Kaip treniravimo duomenis galima panaudoti tiek pilnas Monte–Carlo veiksmų sekas, tiek laikinojo skirtumo metodus.

1.2.4. Pasaulio modeliu paremti metodai

Turint aplinkos modelį ir žinant būsenų pasikeitimo sąlygas ir tikimybes, galima naudoti dinaminį programavimą, nuoseklią paiešką, arba simuliuoti veiksmų sekas euristinės paieškos algoritmams. Tokios aplinkos simuliacijos agento „galvoje“ leidžia atlikti planavimą į priekį ir taip patobulinti agento strategiją, ir sutrumpinti treniravimąsi realioje aplinkoje. Pasaulio modelis ir galimos būsenos ne visada yra žinomos, todėl norint naudoti modeliu grystą strategiją, pirmiausiai reikia išmokti kaip veikia aplinkinis pasaulis naudojant algoritmo įvesties duomenis. Papildoma pasaulio modelio funkcija prideda papildomo sudėtingumo, reikia atlikti daugiau skaičiavimų treniruojant, modelio algoritmas gali turėti klaidų kurios persiduos ir galutiniams agento sprendimams, taip pat treniruojant atsiranda keleto ciklišKai susietų funkcijų sąveika kuri gali trukdyti siekiant pastovaus nuostolio funkcijos konvergavimo.

1.3. Gilusis skatinamasis mokymasis

Paskutiniu metu daug pažangos buvo pasiekta pritaikant giliojo mokymosi metodus kurie gali išmokti aproksimuoti sudėtingas funkcijas, ir efektyviai susidoroti su duomenimis turinčiais daug dimensijų. Pavyzdžiui konvoliuciniai neuroniniai tinklai yra dažnai naudojami pradiniams įvesties duomenims apdoroti [4], [5], nes jie gali, daug pasikartojančios informacijos turinčius, paveikslėlius suspausti į keletą skaitinių parametrų, kurie toliau apdorojami kituose neuroninių tinkų sluoksniuose. Vienas pirmųjų sprendimų kuris panaudojo gilųjį skatinamąjį mokymąsi buvo „Deep Q network“ (DQN) tačiau dabar egzistuoja daug pažangesnių sprendimų [5], [4].

1.3.1. Gilusis Q tinklas

Kuriant DQN algoritmą buvo siekiama išmokinti kompiuterį žaisti *Atari* kompiuterinius žaidimus. Pagrindinė jo idėja yra imituoti klasikinę Q funkciją ir gautus būsenos duomenis paversti į galimų veiksmų naudingumo įverčius. Buvo siekiama kad algoritmas gebėtų išmokti optimalaus valdymo tiesiogiai iš vaizdinių įvesties duomenų, todėl pirmieji sluoksniai yra konvoliuciniai. Dėl skaičiavimo efektyvumo, ekrano kadrai iš pradžių buvo sumažinti ir paversti į vienspalvius. Galiausiai naudota architektūra priėmė 4 84x84 paveikslėlius, pirma konvoliucija turi 16 8x8 dydžio filtrus, antra 32 4x4 su 2 pikselių žingsniu, tuomet naudojamas pilnai sujungtas sluoksnis su 256 neuronais. Tarp visų sluoksnių naudojamos ReLU aktyvacijos funkcijos. Galutinis sluoksnis taip pat yra sujungtas ir turi skirtingą neuronų skaičių nuo 4 iki 18 priklausomai nuo žaidime galimų veiksmų skaičiaus. Kad neuroninį tinklą būtų galima apmokinti efektyviau išnaudojant turimus

skaičiavimo resursus, autoriai pritaikė patirties pakartojimo (ang. Experience repaly) metodą, kai N paskutinių agento sužaistų epizodų yra išsaugomi atmintyje kaip būsenos , veiksmo ir atlygio poros. Atmintis leidžia treniravimą atlikti paketais (ang. batch), kai naudojamas keletas žaidimo istorijos duomenų pavyzdžių kad apskaičiuoti tikslesnį vertės funkcijos gradientą, tai taip pat padidina surinktų duomenų panaudojimo efektyvumą, nes atskiri pavyzdžiai gali būti pasirinkti apmokymui keletą kartų, į atskirus paketus. Naudojant šią strategiją ir renkant pavyzdžius tinklo apmokymui kiekvienoje iteracijoje svarbu kad pavyzdžiai būtų pasirinkti iš atsitiktinų pozicijų, nes gretimi įrašai dažniausiai yra labai statistiškai susiję – apibūdina beveik tą pačią, mažai pasikeitusią būseną, o tai iškreiptų būsenos funkcijos gradiento reikšmę. Neuroninio tinklo svorių atnaujinimui iš pradžių reikia apskaičiuoti gautų rezultatų nuostolio (ang. loss) reikšmę. DQN algoritme ji apibrėžiama kaip – $L_i(\theta_i) = E_{s,a}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]$, čia, $y_i = E_s[r + \gamma max'_a Q(s',a';\theta_{i-1})|s,a]$ yra tikslo funkcija iteracijoje i ir θ_i – žymi neuroninio tinklo parametrus. Verta pastebėti kad tinklo nuostolio reikšmė priklauso nuo tinklo parametrų, tai yra vienas pagrindinių skirtumų nuo paprasto prižiūrimojo mokymosi algoritmų kurių tikslo reikšmės y yra nustatytos prieš pradedant treniravimą ir nekinta. Giliajame skatinamajame mokyme kinta ne tik tinklo spėjimai, bet ir siekiami išmokti tikslai. DQN algoritmas nenaudoja pasaulio modelio, ir neturi išmokstamos strategijos, vietoje to jis išmoksta maksimizuoti atlygį planuojant vieną žingsnį į priekį (godi strategija) [4].

Giliojo Q tinklo architektūra buvo vienas sėkmingiausių žaidimų valdymo sprendimų publikavimo metu, todėl jo pagrindu yra pasiūlyta daug modifikacijų kurios patobulina tam tikrus algoritmo aspektus. Keletas jų :

- Kadangi Q funkcija pasirenka maksimalaus įverčio veiksmą kiekviename žingsnyje, ji dažnai gali pervertinti realų būsenos naudingumo įvertį. Todėl yra pasiūlyta naudoti dvigubo Q tinklo algoritmą, kuris išmoksta 2 atskirus Q įverčius ir naudoja tarpusavio reikšmes skaičiuojant kiekvieno tinko nuostolio funkcijas.

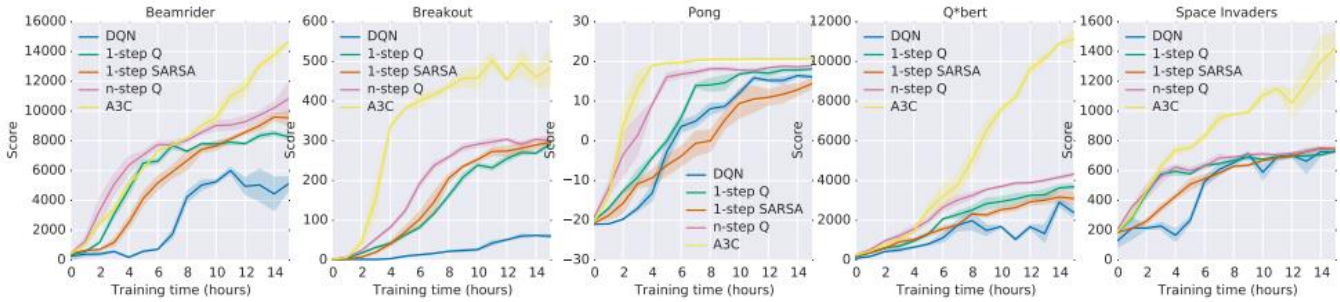
- Dar vienas mokslinis straipsnis siūlo vietoje tikėtino veiksmo naudingumo įverčio, apmokinti tinklą grąžinti tikėtinų reikšmių pasiskirstymą.

- Kitas patobulinimas yra pasiūlymas išskaidyti vieną Q vertės funkciją į atskirus sluoksnius kurie skaičiuotų būsenos vertės V ir pranašumo A funkcijas. Tai padeda tinklui lengviau išmokti būsenos bazinę vertę ir santykinius veiksmų įverčius, vietoje to kad skaičiuoti vertes visiems veiksmams atskirai [5].

1.3.2. Asinchroninis pranašumo aktoriaus–kritiko metodas (A3C)

DQN algoritmo naudojamas patirties pakartojimas leido išspręsti duomenų pavydžių koreliacijos problemą, ir efektyviau panaudojo iš agento patirties gautus duomenis. Tačiau šis sprendimas apriboja algoritmus nuo strategijos funkcijos naudojimo, nes prieš keletą iteracijų surinkti išsaugoti duomenys būtų sugeneruoti naudojant kitokią strategiją, todėl nebūtų tinkami treniruoti naujausią versiją. Tai yra viena iš priežasčių kodėl autoriau pasirinko neuroniniu tinklu aproksimuoti būtent Q funkciją, nes tai yra algoritmas nenaudojantis treniruojamos strategijos, o tik vertės reikšmes. Kad išvengtų šio trūkumo A3C algoritmo autoriai pasiūlė naudoti asinchroninį treniravimą [6], kai tas pats žaidimas žaidžiamas keleto agento kopijų vienu metu. Dėl to kiekviename žingsnyje sugeneruojamas keletas naujų duomenų pavyzdžių, kur visi agento sprendimai sugeneruoti naudojant tuos pačius parametrus. Tai leidžia apmokinti algoritmus kurie naudoja kintančią

strategijos funkciją pavyzdžiui: SARSA, n–žingsnių metodas, arba aktoriaus kritiko metodai. Lygiagretus asinchroninis duomenų rinkimas taip pat turi ir praktinę naudą, nes šiuolaikiniai procesoriai turi keletą branduolių ir galia vykdyti daug simuliacijų vienu metu [5], [6].



1.3 pav. Skirtingų algoritmų surinktų taškų ir mokymosi greičio palyginimas. DQN treniruotas naudojant Nvidia K40, asinchroniniai metodai treniruoti naudojant procesorių su 16 branduolių [6].

Asinchroninis pranašumo aktoriaus kritiko metodas (A3C) naudoja neuroninį tinklą kaip strategijos $\pi(a_t|s_t;\theta)$, ir būsenos naudingumo funkcijos $V(s_t;\theta_v)$ aproksimatorius. Būsenos vertės reikšmė atimama iš visų strategijos įverčių kaip bazinė būsenos vertė, kad būtų sumažinta funkcijos reikšmių variacija. Kaip matoma **1.3 pav**., A3C metodas mokosi greičiau nei ankstesni sprendimai ir pasiekia žymiai geresnių rezultatų. Tinklo apmokymui naudojama modifikuota REINFORCE nuostolio funkcijos reikšmė, kur vietoje atlygių sumos naudojamas pranašumo įvertis. Agentui sąveikaujant su aplinka išsaugoma iki t_{max} ilgio epizodai arba kol pasiekiama galutinė žaidimo būsena, tada kiekviena epizodo būsena yra įvertinama atsižvelgiant į tai kokie atlygiai bus gauti ateities būsenose. Pirmos epizodo būsenos įvertis yra apskaičiuojamas naudojant visą seką, paskutinei būsenai naudojamas tik jos pačios gauta atlygio reikšmė. Parametrus atnaujinantis gradientas gali būti aprašomas formulę $\nabla_\theta \log \pi(a_t|s_t;\theta) A(s_t,a_t,\theta,\theta_v)$, čia $A(s_t,a_t,\theta,\theta_v) = \sum_{i=0}^{k-1}\gamma^i r_{t+1} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$ – pranašumo funkcijos apskaičiavimas, kurk gali variuoti bet neviršinti t_{max}. Algoritmas naudoja lygiagrečius aktorius kad surinktų pakankamai treniravimo pavyzdžių kiekvienoje iteracijoje ir algoritmo apsimokymas išliktų stabilus [6]. Verta pastebėti kad nors ir strategijos ir vertės parametrai žymimi skirtingais simboliais $\theta ir \theta_v$, didžioji dalis parametrų yra tas pats neuroninis tinklas, ir tik paskutinis sluoksnis turi atskiras softmax išvestis strategijai ir vertės funkcijai. Dar vienas autorių pastebėjimas yra kad pridedant papildomos entropijos prie strategijos gradiento, jis paskatinamas pradžioje daugiau laiko skirti eksperimentavimui ir tyrinėjimui, taip išvengiant išankstinio konvergavimo į suboptimalią strategiją. Galutines strategijos gradientas išreiškiamas – $\nabla_\theta \log \pi(a_t|s_t;\theta)\big(R_t - V(s_t;\theta_v)\big) + \beta\nabla_\theta \mathrm{H}\big(\pi(s_t;\theta)\big)$, čia H yra entropija, ir β entropijos sureguliavimo (ang. regularization) stiprumas [6]. Parametrams optimizuoti buvo išbandyti 3 metodai: Momentum SGD, RMSProp ir RMSProp su vidinių parametrų dalinimusi. RMSProp algoritmas tusi slenkančio vidurkio parametrus kurios saugo viduje, kai duomenys yra generuojami keleto simuliacijų vienu metu, šiuos parametrus galima saugoti kiekviename procese atskirai arba turėti bendrą vidurkį. Straipsnyje pateikiami rezultatai rodo geriausias yra RMSProp su bendrais parametrais optimizatorius, o po jo eina RMSProp ir SGD algoritmai. Dar vienas svarbus autorių buvo apie tai kad naudojant lygiagrečiai žaidžiančius agentus ir jų duomenis, buvo pastebimas didesnis nei 1:1 tiesinis algoritmo apsimokymo pagreitėjimas lyginant su algoritmu naudojančiu vieną agentą. Tai grindžiama tuo kad asinchroniniai aktoriai naudingi ne tik dėl didesnio duomenų kiekio, bet ir dėl didesnės jų įvairovės kuri pagerina apskaičiuotų gradientų tikslumą [6].

Autoriai A3C algoritmą testavo diskrečios ir tolygios kontrolės užduotyse, todėl yra aprašytos 2 šiek tiek skirtingos algoritmų architektūros. Diskretiems žaidimams naudojamas neuroninis tinklas kurio pirmas sluoksnis yra 16 8x8 dydžio konvoliucinių filtrų su poslinkiu 4, sekantis sluoksnis taip pat yra 32 4x4 konvoliucinis su poslinkiu 2, kuriam seka pilnai sujungtas sluoksnis su 256 neuronais. Tarp visų sluoksnių naudojamas ReLU aktyvacijos sluoksnis. Aktoriaus–kritimo medeliai turėjo 2 atskirus išvedimus, vieną tiesinį neuroną būsenos vertės reikšmei, ir Softmax aktyvuotą sluoksnį su tiek neuronų kiek yra galimų veiksmų, kurk kiekviena reikšmė nusako veiksmo pasirinkimo tikimybę. Tuo tarpu tolygios kontrolės užduotims tokioms kaip MuJoCO fizikos stimuliatoriui, neuroninis tinklas buvo modifikuotas, kad po pilnai sujungto sluoksnio yra 128 blokų LSTM sluoksnis. Dar svarbesnis pakeitimas yra strategijos funkcijos rezultatuose, vietoje vienos reikšmės tinklas grąžina 2 reikšmes kiekvienam galimam veiksmui kurios rodo vidurkį μ ir variaciją σ^2. Veiksmai pasirenkami pagal tinklo grąžinta pasiskirstymą. Tolydžių užduočių algoritmai turi visiškai atskirus tinklus strategijos ir vertės funkcijos, vietoje diskrečių užduočių sujungto tinklo, nors autoriai teigia, kad tai neturėtų būti esminis skirtumas. Kadangi tolydžių užduočių simuliacijos epizodai užtrunka kelis šimtus žingsnių, nėra reikalo apjunginėti kelių epizodų duomenis, todėl tinklo parametrų atnaujinimai atliekami po kiekvieno simuliacijos epizodo [6]

1.3.3. Pagalbinės užduotys skatinamajam mokymuisi, UNREAL algoritmas

Skatinamojo mokymosi algoritmai tokie kaip A3C naudoja agento surinktą patirtį kokios žaidimo būsenos duoda taškų ir kokios ne bei stengiasi optimizuoti neuroninio tinklo parametrus taip kad būtų maksimizuotas gautų taškų kiekis. Tačiau treniravimo pradžioje neuroniniai tinklai yra atsitiktinai inicializuoti ir agento veiksmai nėra nuoseklūs, jis klaidžioje negaudamas teigiamų įverčių. Tai neleidžia efektyviai optimizuoti tinklo nes duomenys neturi teigiamų pavyzdžių kaip agentas turėtų elgtis. Šiai problemai spręsti straipsnio autoriai pristato keletą algoritmo patobulinimų originaliam A3C [6] sprendimui kurie padeda iki 10 kartų pagreitinti tinklo treniravimą, bei pagerinti galutinius rezultatus patobulinant tinklo vidinę būsenos reprezentaciją. Rezultatų pagerėjimas labiausiai matomas tolygios kontrolės užduotyse tokiose kaip DeepMind Labyrinth [7].

Kadangi kompiuteriniai žaidimai ir ypač realios aplinkos turi daug daugiau nesiekiamų būsenų nei siekiamų, mažai tikėtina kad agentas galėtų greitai suprasti kurie veiksmai duoda teigiamus rezultatus. Dėl to siūloma pridėti fiktyvias atlygio funkcijas kurios skatina agentą tyrinėti, bei nuspėti kaip jo veiksmai keičia aplinkinį pasaulį. Tam panaudotos 2 papildomos funkcijos. Pirmoji naudoja aplinkos reprezentacijas gautas iš konvoliucinių sluoksnių ir bando nuspėti koks bus kito būsenos kadro įvertis, kai turima prieš tai buvusių būsenų seka. Šios dalies treniravimui duomenys surenkami iš 2 agento istorijos saugyklų kurių viena saugo duomenis su teigiamais atlygiais o kita be teigiamų rezultatų. Kiekviena treniravimo duomenų grupė surenkama iš lygaus skaičiaus teigiamų ir neigiamų pavyzdžių, kas leidžia greičiau išmokti kurios aplinkos vietos duoda geriausią atlygį. Kita papildoma funkcija stengiasi pagerinti tinklo tyrinėjimą ir informacijos rinkimą. Tam pridėta nauja tinklo atšaka papildomai strategijai, kuri naudoja tinklo duomenis iš LSTM sluoksnio ir išskleidžia juos naudojant dekonvoliucijos operaciją. Tuomet tinklas yra treniruojamas pasirinkti tokius veiksmus kurie maksimaliai keistų pikselių reikšmes, nes tai reikštų kad agentas tyrinėja naują aplinką. Dar viena modifikacija kuria siekiama pagreitinti treniravimą yra patirties pakartojimas, toks kokį naudojo DQN algoritmas. Patirties pakartojimas naudojamas naujai atlygio spėjimo funkcijai, ir taip pat naudojamas atskira saugykla naujausioms agento patirties sekoms

kurios naudojamos papildomai treniruoti A3C algoritmo – būsenos vertės funkciją. Pilna UNREAL algoritmo nuostolio funkcija aprašom formule $L_{UNREAL}(\theta) = L_{A3C} + \lambda_{VR}L_{VR} + \lambda_{PC\sum_c L_Q^{(c)}} + \lambda_{RP}L_{RP}$, čia L_Q žymi papildomos, pikselių kontrolės, užduoties nuostolio funkciją, L_{RP}– atlygio prognozės, ir L_{VR} atlygio patirties pakartojimo. Skirtingos nuostolio funkcijos ir optimizatoriai veikia per skirtingas tinklo atšakas, bet dalis parametrų tokių kaip konvoliucinei bei LSTM sluoksniai yra bendri. Dėl to pagerėja jų kokybė ir informacijos reprezentacijos universalumas [7]

1.3.4. Aktoriaus kritiko patirties pakartojimas (ACER)

Didelė dalis giliojo skatinamojo mokymosi algoritmų patobulėjimų stipriai remiasi dideliu kiekiu duomenų gautų iš ilgai trunkančių simuliacijų. Siekis sumažinti treniravimosi, ir agento sąveikos su aplinka laiką ypač svarbus agentams veikiantiems realioje aplinkoje. Universaliausi ir geriausius rezultatus pasiekia – strategijos funkciją naudojantys metodai, todėl ACER [8] algoritmo autoriai papildė A3C realizaciją taip kad jis galėtų efektyviai išsaugoti ir pakartotinai panaudoti duomenis treniravimui. Taip pat siekiama kad algoritmas būtų efektyvus tiek diskrečiose tiek tolydžiose užduotyse [8].

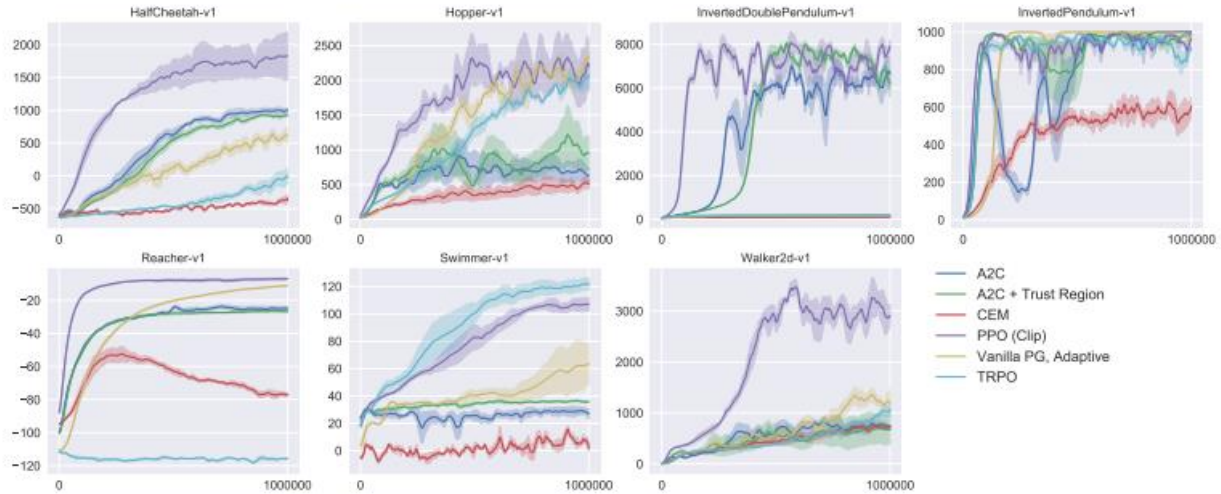
Skirtingose iteracijose atlikti agento veiksmai buvo pasirinkti naudojant kitokias strategijos versijas, dėl to pagal juos negalima patikimai aproksimuoti dabartinės strategijos gradiento. Kad agento strategiją būtų galima pakartotinai apmokyti panaudojant išsaugotus ankstesnius, agento veiksmų epizodus, straipsnio autoriai pasiūlė naudoti strategijos gradientą su svarbumo svoriais, kurie yra apskaičiuojami pagal ankstesnės ir dabartinės strategijos panašumus. Panašumo koeficientas žymimas ρ_t ir apskaičiuojamas pagal formulę $\rho_t = \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}$, čia π žymi dabartinę strategiją, o μ ankstesnę strategiją pagal kurią buvo sugeneruoti agento veiksmai, a_t – pasirinktas veiksmas, x_t – aplinką apibūdinantis vektorius. A3C algoritme naudojama „kritiko“ funkcija kuri buvo apskaičiuojama pagal būsenos vertės funkciją ir gautas atlygio reikšmes. Kadangi ACER naudoja duomenis sugeneruotus ir su senesnėmis strategijomis, jame naudojama Q funkcija nes ji gali būti apmokinama naudojant veiksmus sugeneruotus kitos strategijos (ang off–policy). Taip pat kadangi Q funkcija yra aproksimuojama iš keleto žingsnių, jos įvertis yra tikslesnis nei naudojant būsenos vertės funkciją su gautis atlygiais. Straipsnio autoriai naudoja Retrace algoritmą Q funkcijos aproksimavimui, kuris gali būti išreiškiamas rekursine formule $Q^{ret}(x_t, a_t) = r_t + \gamma\overline{\rho}_{t+1}[Q_{ret}(x_{t+1}, a_{t+1}) - Q(x_{t+1}, a_{t+1})] + \gamma V(x_{t+1})$, čia $\overline{\rho}_t = \min\{c, \rho_t\}$ apribotas strategijų panašumo koeficientas, $V(x) = E_{a\pi}Q(x, a)$. Yra įrodyta, kad Retrace algoritmas konverguoja į tikslinės strategijos vertės funkciją. Galiausiai algoritme naudojamas strategijos gradientas aprašomas formule $g^{marg} = E_{x_t\beta, a_t\mu}\left[\rho_{t\nabla_\theta} log\pi_\theta(a_t|x_t)Q^\pi(x_t, a_t)\right]$, čia $\beta, ir\mu$ atitinkamai aprašo būsenų ir veiksmų pasiskirstymus [8].

Treniravimo metu, aktoriaus kritiko metodo strategijos funkcijos tikslinės reikšmės dažnai turi didelį kintamumą, todėl norint užtikrinti tolydų ir stabilų tinklo treniravimą, reikia apriboti kiek reikšmės gali būti pakeistos kiekvienoje iteracijoje. Atmetus metodus kurie reikalauja per daug resursų, ACER autoriai siūlo naudoti slenkančios vidutinės strategijos tinklą. Vidutinės strategijos tinklo parametrai θ_a atnaujinami kiekvienoje iteracijoje pagal formulę $\theta_a = \alpha\theta_a + (1 - \alpha)$. Einamosios strategijos funkcijos gradientas perskai2iuojamas taip kad neviršijant limito naudojama anksčiau minėta funkcija, jei pokytis per didelis – gradientas nukreipiamas labiau vidutinės strategijos kryptimi [8].

1.3.5. Proksimaliniai strategijos optimizavimo algoritmai (PPO)

Skatinamojo mokymosi literatūroje dažnai minimas PPO (ang. Proximal policy optimization) [10] algoritmas. Jis gali būti apibūdintas kaip labai paprasta nuostolio funkcijos modifikacija, kuri stengiasi išvengti didelių strategijos funkcijos pokyčių. Dėl paprasto suformulavimo, tinkamas naudoti įvairiose užduotyse. PPO nuostolio funkcija apibrėžiama $L^{CLIP+VF+S}(\theta) = \mathrm{E}[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S(s_t|\pi_\theta)]$, čia S yra strategijos entropijos matas, skirtas paskatinti tyrinėjimą, $L^{VF}(\theta) = (V_\theta(s_t) - V^{targ})^2$, parametrai $c_1 \mathrm{ir} c_2$ naudojami kaip algoritmo hiperparametrai, ir $L^{CLIP}(\theta) = \mathrm{E}\left[min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t, clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)A_t\right)\right]$, kurioje A žymi pranašumo funkciją, o ϵ yra hiperparametras su autorių siūloma reikšme 0.2 [10]. Šis funkcijos apibrėžimas gali būti naudojamas kaip tiesioginis nuostolio funkcijos pakeitimas klasikiniam strategijos gradiento metodui. Clip ir min funkcijos apriboja algoritmą nuo didelių strategijos pokyčių, jei per daug skiriasi nuo ankstesnės versijos. Pranašumas skaičiuojamas pagal metodą GAE [9]. Šis metodas yra tikslesnis už paprastą agento atlygių sudėjimą su gamma nuvertėjimu, nes jis taip pat atsižvelgia į trajektorijos patikimumą ir kokią dalį rezultato sudaro suapvalintos reikšmės (vertės funkcijos spėjimas). Taip pat šiame straipsnyje paminėta kad Adam tinklo optimizavimo algoritmas tinkamas naudoti su PPO kaip greitesnė gradientinio nusileidimo versija. Duomenų surinkimas vykdomas keliuose lygiagrečių agentų, per N ilgio epizodus. Tuomet vykdoma keletas epochų gradientinio optimizavimo duomenų paketais (ang. batches) [10].

Iš pateiktų rezultatų matyti kad PPO algoritmas demonstruoja greitą ir kokybišką apsimokymą tolydžios kontrolės užduotyse, tačiau Arcade žaidimuose surinko mažiau taškų nei ACER algoritmas. Verta pastebėti, kad šiam metodui pavyko pasiekti gerus rezultatus ir keletoje Arcade žaidimų kurių ACER ir A3C nesugebėjo išmokti [10].

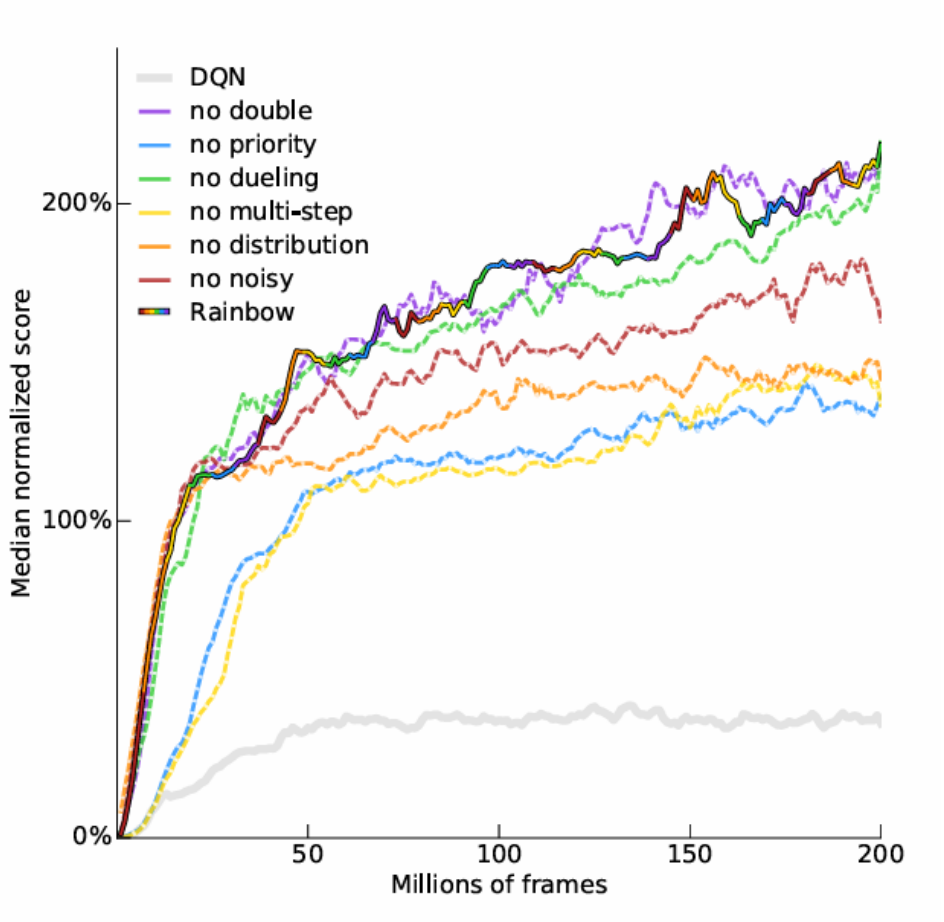


1.4 pav. Keleto algoritmų apsimokymo palyginimas skirtingose tolydžios kontrolės užduotyse [10]

1.3.6. Rainbow

Ieškant pavyzdžių kurie galėtų padėti patobulinti algoritmo efektyvumą itin tinkamas yra Rainbow pavyzdys [11]. Tai yra giliojo Q tinklo algoritmo [4] įvairių patobulinimų junginys. Straipsnyje rašoma kad modifikacijų junginys leidžia agentu surinki ~3 kartus daugiau taškų jei originali DQN

versija. Kai tai pasiektų Rainbow panaudoja 7 modifikacijas pasiūlytas įvairiuose kituose straipsniuose. Jų įtaką rezultatui galima matyti **1.5 pav**.



1.5 pav. Skirtingų modifikacijų įtaka Rainbow algoritmui. DQN žymi bazinę versiją be modifikacijų. Kiekviena kreivė rodo rezultatus pašalinus vieną iš modifikacijų. Kuo didesnis atotrūkis nuo Rainbow – tuo svarbesnė modifikacija [11]

1.4. Kitos mašininio mokymosi inovacijos

Realizuojamas RL algoritmas bus paremtas giliaisiais neuroniniais tinklais todėl naudinga pasiremti mašininio mokymosi inovacijomis kurios padeda stabilizuoti mokymąsi ir kitose neuroninių tinklų panaudojimo srityse. Dauguma šiuolaikinių neuroninių tinklų modelių naudoją vienokią ar kitokią normalizacijos variaciją, kad įgalintų gilesnių tinklų treniravimą. Su sekomis ir laiko eilutėmis susiję modeliai naudoja rekurentinių tinklų architektūras kad suteiktų tinklui gebėjimą atsiminti kitus nei tik įvestyse esantys duomenys. Čia pateikiami keli populiariausi naudojami sprendimai ir jų veikimo principai.

1.4.1. Rekurentiniai tinklai

Skatinamojo mokymosi algoritmai paprastai priima sąlygą kad sprendžiama problema gali būti aprašyta MDP (Markovo sprendimų procesu) kuris teigia kad aplinkos būsena agentui yra pilnai matoma. Tačiau nevisi uždaviniai tenkina šią sąlygą. Pavyzdžiui 3D tipo kompiuteriniai žaidimai vienu metu ekrane parodo tik daly scenos, o kita dalis turi būti įsiminta kai agentas vaikšto arba sukiojasi aplik. Neuroninių tinklų architektūrų klasė kuri geba atsiminti ankstesnes įvestis yra vadinama rekurentiniais tinklais. Jie pasižymi tuo kad turi vidinius būsenos vektorius kuriuos

perduoda sekančiai iteracijai ir taip įgalina tinklą atsiminti. Plačiausiai naudojami LSTM ir GRU variantai kurie turi papildomos išmokstamos logikos kuri valdo kokia informacija yra atsimenama, pamirštama, ir kokia dalis parodoma kaip išvestys. Nuo pirmosios publikacijos 1995 LSTM architektūrai buvo pasiūlyta daugelis modifikacijų, tačiau dažniausiai naudojama yra išreiškiama per formules:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$

$$f_t = \sigma\big(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\big)$$

$$g_t = \tanh\big(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\big)$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot t\tanh(c_t)$$

Čia *t* žymi iteracijos numerį, *x* įvestis, *W* ir *b* svorius neuronų sluoksniuose, *h* yra LSTM vidinė būsena, tanh ir σ – aktyvacijos funkcijos, ⊙ – Hadamardo daugyba (atitinkamų matricos narių sandauga). Toliau *i, f, g, o* žymi įvesties, pamiršimo, išvesties ir būsenos vartai, *c* yra išvestis. Nuo pasirinktų matricų dydžio priklauso kiek elementų turės paslėptosios būsenos ir išvesties vektoriai. Visi literatūroje narinėti algoritmai naudojo LSTM sluoksnį, kai buvo sprendžiamos 3D pasaulio užduotys [12] [13].

1.4.2. Normalizacijos

Kai treniruojamas gilius neuroninis tinklas su daug sluoksnių, kiekviena įvesčių daugyba su sekančio sluoksnio svoriais iškreipia aktyvacijų pasiskirstymą. Po kiekvienos daugybos galimų reikšmių intervalas didėja ir atsiranda vis labiau nuo 1 ir -1 nutolusių reikšmių. Tai kenkia mokymuisi nes atgalinio žingsnio metu vieni neuronai gauna didesnius arba mažesnius atnaujinimus nei kiti, taip pat išsikreipia skirtingų sluoksnių mokymosi greitis. Šiai problemai spręsti buvo pasiūlytas plačiai naudojamas sprendimas rinkinio normalizavimas (ang. Batch normalization) [15]. Vėliau buvo pasiūlyti optimalesni svorių (ang. Weight normalization) [16] ir sluoksnių (ang. Layer normalization) [14] normalizavimo sprendimai. Visi jie siekia atstatyti normalų aktyvacijų pasiskirstymą, ir gali būti įterpiami po kiekvieno sluoksnio. Kaip rodo rezultatai literatūroje, tai leidžia patikimai ir stabiliai apmokyti gilesnius neuroninius tinklus. Rinkinio normalizavimas atlieka korekciją su visu duotu duomenų rinkiniu, o tai nėra teigiamas efektas, nes duomenų pavyzdžiai neturėtų būti tarpusavyje susiję. Dėl to geriausiu laikomas sluoksnio normalizavimo metodas. Kaip rodo autorių straipsnyje pateikiami duomenys, tinklai su normalizavimu pasiekia geresnių galutinių rezultatų, o panaudojus sluoksnio normalizavimą LSTM viduje, jo konvergavimas pagreitėja iki 8 kartų tam tikrose užduotyse [14] [15] [16].

1.5. Išvados

Atlikus literatūros analizę nuspręsta, kad šiai užduočiai bus naudojamas metodas kaip ir dauguma nenaudojantis pasaulio modelio. Sprendimų priėmimui geriausia naudoti aktoriaus–kritiko modelį, dėl stabilesnio mokymosi ir geresnių rezultatų. Q funkcija grįsti metodai pasiekia geresnį duomenų panaudojimo efektyvumą ir garantuoja kad algoritmas galiausiai pasieks optimalią strategiją, tačiau

jiems reikalingas tikslus hiperparametrų optimizavimas, mokymasis dažnai yra nestabilus ir gali nekonverguoti. Tuo tarpu strategijos gradientu grįsti metodai leidžia spręsti įvairesnių tipų užduotis, jų mokymasis yra stabilesnis su didesne tolerancija hiperparametrų reikšmėms, tačiau jiems reikalingas didelis kiekis vis naujai sugeneruotų duomenų. Skatinamajame mokymesi svarbus simuliacijų vykdymo ir duomenų rinkimo klausimas, kaip rodo literatūra, lygiagretus keleto agento kopijų naudojimas ypač pagerina mokymosi greitį ir stabilumą, todėl svarbu turėti tinkamus įrankius žaidimų simuliacijoms vykdyti kuo lygiagrečiau. Tai taip pat pagretina eksperimentų vykdymą ir leidžia išbandyti daugiau hiperparametrų reikšmių, kurių reguliavimas yra labai svarbus RL algoritmams [17].

2. Žaidimai, reikalavimai ir duomenų apdorojimo infrastruktūra

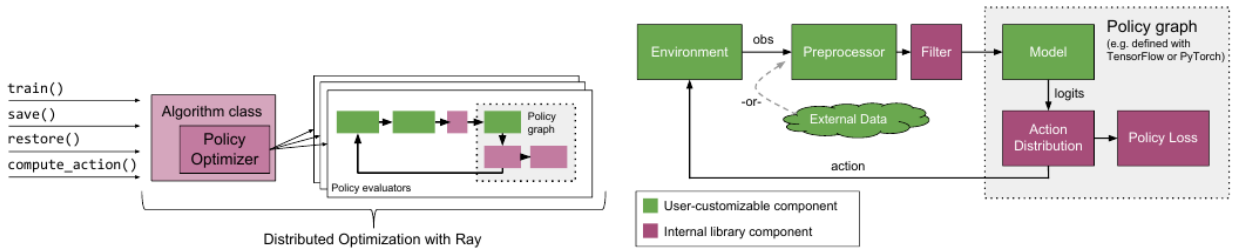
Šiame skyriuje aprašoma techninė projekto realizavimo pusė, pasirinkti įrankiai ir veikimo schema. Pirmame skyriuje aptariama, kokių programinių įrankių reikia norint efektyviai vykdyti gilioje skatinamojo mokymosi eksperimentus, bei kokie žaidimai būtų tinkamiausi lyginant algoritmų kokybę. Reikalavimų skiltyje pateikiami projektui iškelti programiniai ir aparatūriniai apribojimai, bei pagrindiniai funkciniai ir nefunkciniai reikalavimai. Pateikiama funkcionalumo ir algoritmo treniravimo sekų diagramos. Projekto plano skiltyje pateikiama kokie yra žingsniai siekiant realizuoti norimus pakeitimus algoritme ir išsiaiškinti kaip atlikti pakeitimai pakeitė programos efektyvumą. Kadangi tie patys skatinamojo mokymosi algoritmai gali būti sukonfigūruoti su skirtingais hiperparametrais, algoritmų skilty aptariama, kokia variacija buvo naudojama pradiniame straipsnyje ir kokios korekcijos buvo reikalingos šiame darbe realizuotai versijai. Galiausiai, paskutiniuose skyriuose aptariama, kokie pakeitimai bus atlikti planuojamuose eksperimentuose ir koksai jų pasirinkimo pagrindimas. Toliau seka kokybės kriterijai, pagal kuriuos matuojama projekto sėkmė.

Įrankių analizė

Kaip galima matyti iš algoritmų ACER [8], PPO [10], A3C [6], Rainbow [11] eksperimentinių dalių, dabartiniams skatinamojo mokymosi algoritmas reikalingas didelis kiekis treniravimo duomenų kad jie pasiektų priimtiną lygį. Literatūroje dažnai lyginami algoritmų rezultatai po 1 mln., 40 mln., 200 mln., ar dar daugiau žaidimo sugeneruotų kadrų. Daugiausiai treniravimo duomenų reikalauja užduotys su dideliu kiekiu įvesties duomenų dimensijų (pvz. nuotraukos/vaizdo įrašai). Taip pat sudėtinga išmokti žaidimus, kuriuose reikalinga strategija ir planavimas, nes reikia išmokti įvertinti didelį skaičių galimų būsenų, o tam reikia daugiau kadrų iš žaidimo stimuliatoriaus. Todėl, norint kurti skatinamojo mokymosi algoritmus ir juos efektyviai apmokinti, reikalinga infrastruktūra ne tik neuroniniams tinklams aprašyti, bet ir paleisti didelį kiekį lygiagrečių procesų su pasirinkto žaidimo kopija. Taip pat reikia turėti komunikacijos metodus, leidžiančius išsiųsti agento veiksmus ir surinkti rezultatus iš atskirų procesų. Tai naudinga ne vien dėl efektyvesnio modernios aparatūrinės įrangos išnaudojimo, bet ir dėl papildomo stabilizuojančio efekto, kurį suteikia didesnė surinktų duomenų įvairovė [6]. Iš vieno proceso surinkti kadrai turi didelę koreliaciją, o tai blogai atspindi tikrąjį duomenų pasiskirstymą ir neleidžia apskaičiuoti gero gradiento tinklo optimizavimui.

Atliekant įrankių paiešką buvo pastebėta, kad vyrauja 2 žaidimo aplinkų lygiagretinimo metodai. Tai yra: vektorizavimas ir lygiagretūs procesai. Vektorizavimu yra vadinamas keleto simuliacijų nuoseklus vykdymas viename procese. Tai sumažina perteklinį, su žaidimo veikimu nesusijusių duomenų kopijavimą, nes, procesų sukūrimas reikalauja papildomų resursų skaičiavimo ir RAM atminties prasme, o taip pat juose laikomos ir neuroninių tinklų kopijos. Nepaisant vieno proceso efektyvumo jis vis tiek gali atlikti tik ribotą operacijų skaičių, todėl siekiant pagreitinti apmokinimą, didesnės apimties algoritmams naudojamas lygiagretus žaidimų simuliavimas kelete procesoriaus branduolių. Kadangi šie metodai padeda spęsti skirtingas problemas, geriausiems rezultatams pasiekti naudojamas abiejų derinys. Pagal šiuos kriterijus pasirinkome naudoti Ray [18] python biblioteką kurios rllib [19] dalis skirta giliajam skatinamajam mokymuisi. Rllib galimybės leidžia lengvai panaudoti abu aplinkų lygiagretinimo metodus, taip pat yra galimybė treniravimą vykdyti dideliu mastu procesus paskirstant per kompiuterių tinklą. Bibliotekoje yra realizuota keletas

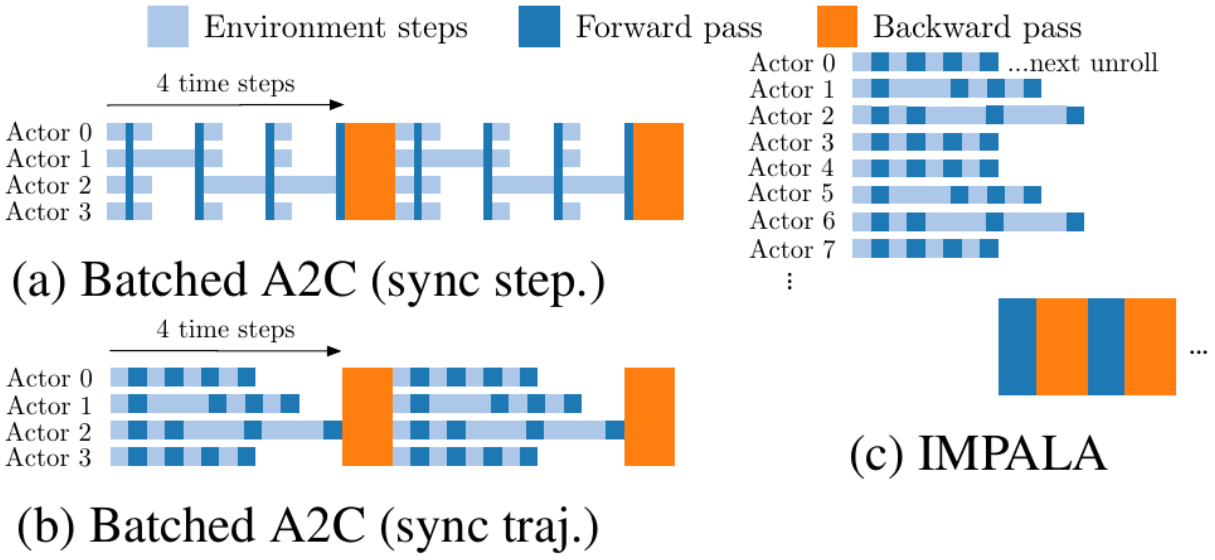
populiariausių RL algoritmų. Paveikslėlyje **2.1 pav** matoma rllib karkaso veikimo seka ir pagrindinės klasės. Norimas algoritmas aprašomas užklojant atitinkamas bazines klases.



(a) High-level RLlib algorithm API (b) Policy evaluator internal components

2.1 pav. Apibendrinta rllib bibliotekos klasių komunikavimo schema kai vykdomas agento mokymas [19]

Neuroninių tinklų treniravimas reikalauja daug lygiagrečių skaičiavimų, todėl šis darbas paprastai atliekamas ant GPU įrenginių. Prieš pradedant gradientinio optimizavimo operaciją, reikia duomenis, iš žaidimų simuliacijos, esančius RAM perkelti į GPU esančią VRAM atmintį. Tai galima atlikti sinchroniniu arba asinchroniniu metodu, o rllib bibliotekoje galima naudoti abu variantus. Sinchroniniu atveju GPU laukia kol visi CPU branduoliai baigs darbą ir surinks nustatytą kiekį kadrų, nors tai yra paprastesnis ir dažniau naudojamas metodas, jis neleidžia pilnai išnaudoti kompiuterio resursų nes vienu metu dirba tik CPU arba GPU bet ne abu. Asinchroniniu atveju GPU atlieka tinklo treniravimą tik su tų procesų duomenimis kurie jau baigė darbą, o kitiems leidžia toliau dirbti, nors tai ir nepakeičia algoritmo esmės, treniravimas atliekamas greičiau kai skaičiuojama realiu laiku. Paveikslėlyje **2.2 pav** matomas veiksmų išsidėstymas naudojant skirtingus duomenų surinkimo metodus, A2C yra sinchroninis A3C [6] algoritmo variantas.



2.2 pav. Sinchroninis ir asinchroninis duomenų surinkimas [20]

Asinchroninio RL agento treniravimo idėja buvo pristatyta IMPALA [20] algoritme. Autoriai parodė kad paskirstant darbus tokiu būdu galima padidinti per sekundę surinktų kadrų kiekį nuo ~2-

4 karus priklausomai nuo papildomų sąlygų. Taip pat apmokant šiuo būdu algoritmas pasiekia geresnį galutinį rezultatą, o treniravimas GPU vyksta tuo pat metu kaip ir duomenų rinkimo darbai. Autoriai teikia kad IMPALA gali pasiekti tokius pačius rezultatus kaip A3C algoritmas po 10 valandų vietoje 7,5 dienos [20].

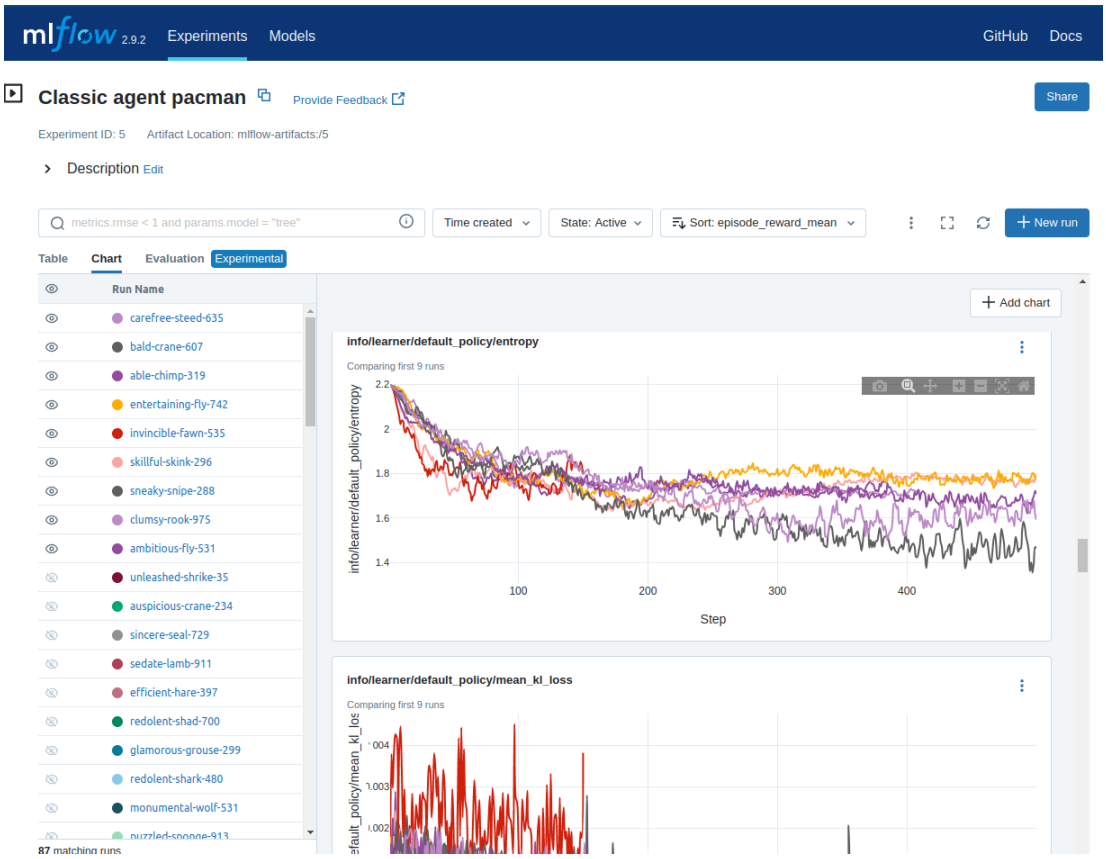
Kuriant skatinamojo mokymosi algoritmus taip pat svarbu nuspręsti kokio tipo žaidimas ar aplinka bus reikalinga. Esama keletas kategorijų kurios skirstomos pagal įvesties ir išvesties duomenų tipus, nuo jų priklauso kokie algoritmai bus tinkami tai problemai spręsti. RL agento stebima aplinka (įvestys) gali būti skaitinės reikšmės reiškiančios objektų padėtis ir judėjimą arba pikselių reikšmių matrica t. y. vaizdinė informacija. Agento veiksmai gali būti arba dvejetainės reikšmes reiškiančios mygtukų paspaudimus, arba tolydūs kintamieji reiškiantys įvairius kintamo intensyvumo veiksmus, pvz. kompiuterio pelės judėjimas. Siekiant kad kuriamas agentas būtų kuo universalesnis, buvo pasirinkta naudoti aplinką su pikselių matricos įvestimis ir veiksmų aibe kuri turi tiek dvejetainius, tiek tolydžių reikšmių veiksmus. Kad vykdomi eksperimentai vyktų greičiau, varta naudoti žaidimą kuris naudoja kuo mažiau kompiuterio resursų nes tai leidžia greičiau generuoti duomenis kaip buvo minėta anksčiau. Pagal šiuos kriterijus pasirinkome naudoti Doom žaidimo repliką VizDoom [21] kurioje yra keletas skirtingų lygių su įvairiomis užduotimis, taip pat ši biblioteka atitinka gymnasium [22] API kuris yra plačiai naudojamas skatinamojo mokymosi industrijos, atviro kodo standartas paremtas OpenAI Gym API.

2.3 pav. Ekrano vaizdas gautas
iš vizDoom bibliotekos žaidimo

Atliekant eksperimentus su skatinamojo mokymosi algoritmais, reikia surinkinėti parametrus apie algoritmo veikimą, kad vėliau būtų galima palyginti skirtingas eksperimento versijas, ir ieškoti potencialių klaidų. Testavimo tikslais taip pat patogu turėti vaizdo įrašą kaip agentas žaidžia žaidimą, skirtingose treniravimo epochose. Dėl to buvo pasirinkta naudoti atviro kodo mlFlow biblioteką, kuri veikia kaip serverinė programa, o RL algoritmas prisijungia prie jos pagal konfigūracijų faile nurodytą IP adresą. mlFlow šio projekto tikslams naudojamas parametrų rinkimo, video failų ir diagramų išsaugojimo, eksperimentų palyginimo ir organizavimo , ir neuroninių tinklų parametrų išsaugojimo (skirtingos versijos einant apmokymui) funkcionalumai. MLflow sąsajos vaizdą galima matyti 2.4 pav paveikslėlyje.

Bendrai programavimui buvo pasirinkta naudoti Python programavimo kalbą, dėl jos plataus naudojimo tiek gilioje tiek skatinamojo mokymosi srityse, plataus bibliotekų pasirinkimo, bei asmeninės patirties dirbant su ja. Taip pat Python yra vienas universaliausių įrankių siekiant apjungti skirtingų sričių funkcijas, šiuo atveju viena programa turi atlikti žaidimų simuliacijas

paskirstytoje aplinkoje, atlikti neuroninių tinklų treniravimą grafikos įrenginyje, ir sekti ir dokumentuoti gautus duomenis tolimesniam tyrinėjimui. Neuroninių tinklų realizavimui ir apmokymui pasirinkome naudoti PyTorch biblioteką dėl jos plataus naudojimo, lengvo klaidų aptikimo ir šalinimo. Kodo rašymui ir versijų valdymui naudotos „Microsoft“ VS Code ir Git programos.



2.4 pav. Ekrano vaizdas matomas MLflow serveryje. Sąsajos kairėje galima matyti skirtingų eksperimentų sąrašą, o dešinėje parametro kitimo apmokymo metu grafiką. Skirtingos spalvos žymi skirtingus eksperimentus

2.1. Kuriamo sprendimo reikalavimai

Šioje skiltyje aprašomi pagrindiniai kuriamą sistemą apibudinantys kriterijai. Iš pradžių pateikiama kokios aplinkos reikia kad kuriama programa galėtų veikti, toliau aprašoma kokio funkcionalumo siekiama.

2.1.1. Aparatūriniai ir programiniai reikalavimai

Kad realizuojama programa veiktų, kompiuteris turi atitiktį šious reikalavimus:

- Operacinė sistema: Linux kernel 5.15 64–bit
- Python kalbos interpretatorius verisja 3.10.2

Papildomos Python bibliotekos:

- Ray .2.9.0
- VizDoom 1.2.3
- gymnasium 0.28.1
- PyTorch 2.1.2
- OpenCV 4.5.5
- MlFlow 2.9.2
- Plotly 5.18.0

Eksperimentai atliekami kompiuteryje su šiais aparatūrinias parametrais:

- Prociasorius su 12/24 fiziniai/loginiais branduoliais
- 32 GB RAM atminties
- Vaizdo plokštė palaikanti Nvidia CUDA 12.4 biblioteką ir turinti 24 GB integruotos atminties.

2.1.2. Funkciniai ir nefunkciniai reikalavimai

Algoritmas turi tenkinti šiuos funkcinius reikalavimus:

- Algoritmo apmokymui skirti parametrai nurodomi per konfigūracinius failus
- Treniravimo metu gaunamos metrikos turi būti surinktos ir išsaugotos
- Su mokymo rezultatų kreivėmis turi būti saugojama ir agento žaidimo pavyzdžio video.
- Apmokinti modeliai saugomi serveryje kartu su treniravimo rezultatais.

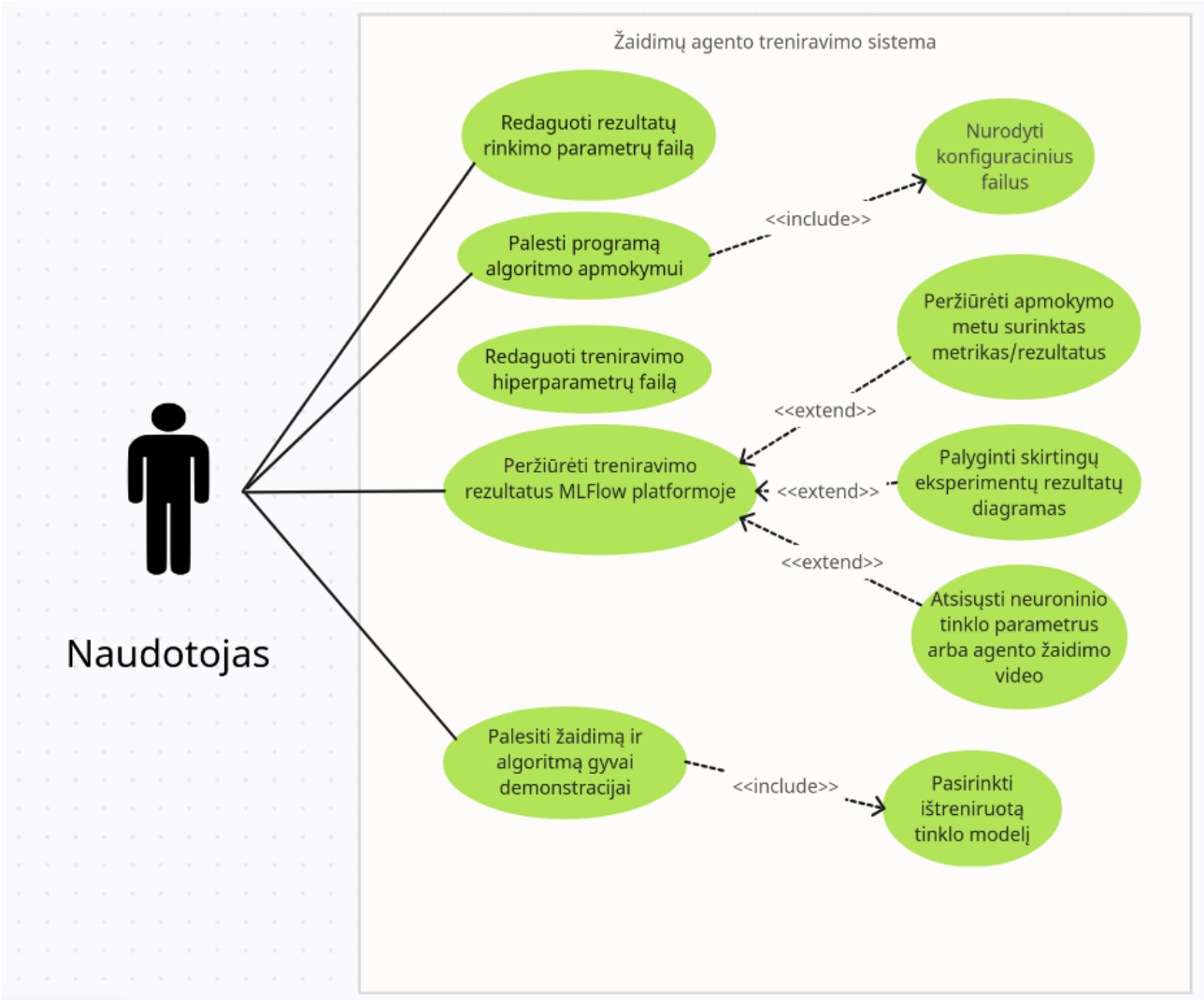
Nefunkciniai reikalavimai:

- Žaidimų simuliacijos vykdomos lygiagrečiai, pasirinktame skaičiuje CPU branduolių
- Žaidimų aplinkos duomenis perduoda per bibliotekos gymnasium [22] apibrėžtą sąsają
- Algoritmas kaip įvestis naudoja tik vaizdinius duomenis (pikselių reikšmes)
- Algoritmas geba išmokti valdyti žaidimo agentą tolydžiomis ir diskrečiomis išvestimis
- Nenaudojami euristiniai metodai konkrečiam žaidimui.

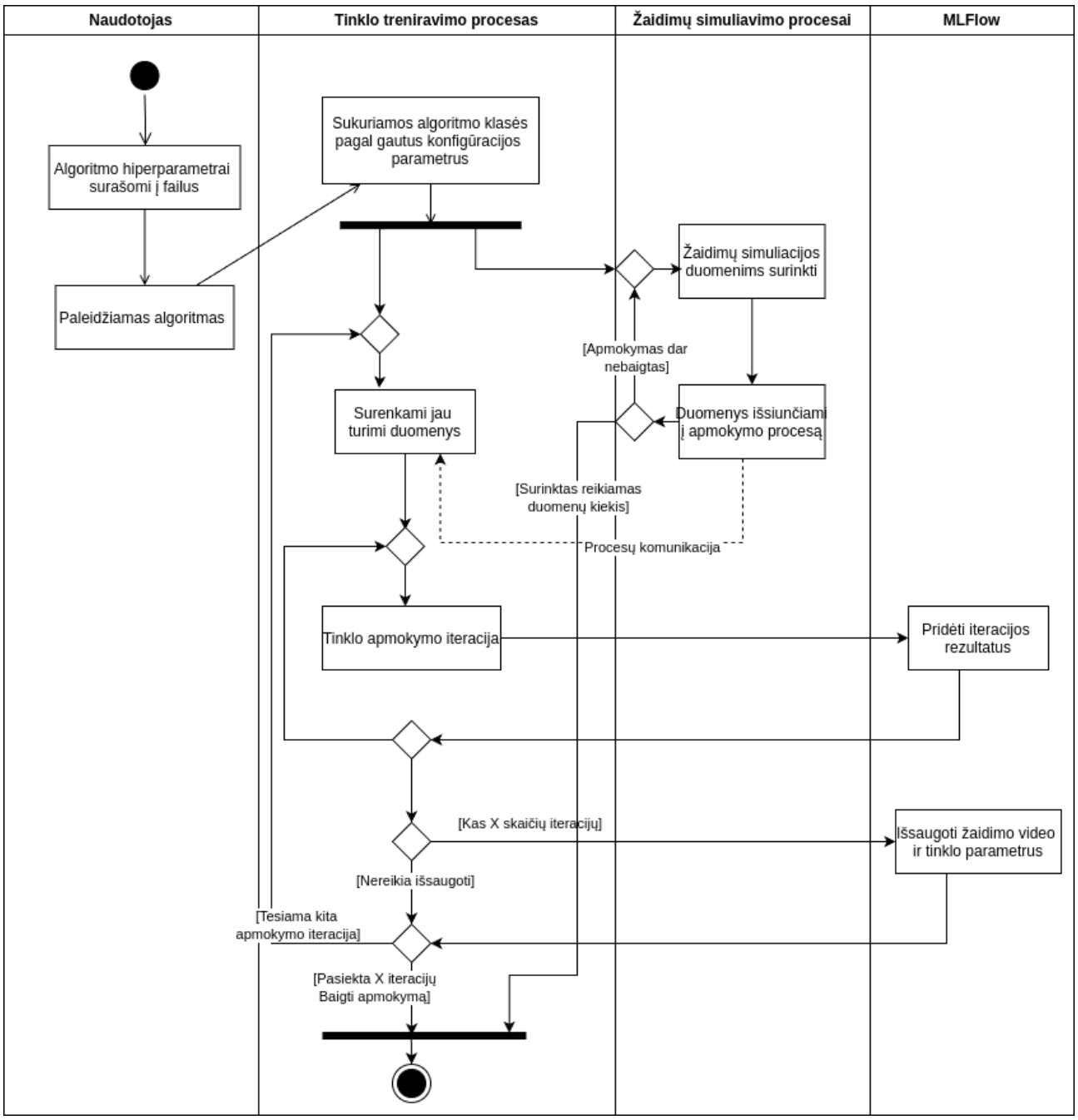
2.1.3. Sistemos funkcijos ir veikimo metodai

Kuriamos programos tikslinis naudotojas yra tas kuris nori atlikti eksperimentus, kad apmokintų pasirinktą algoritmą tam tikroje aplinkoje ir surasti jam optimalių hiperparametrų rinkinį.

Diagramoje vaizduojamos funkcijos kurias gali atlikti vartotojas.



2.5 pav. Kuriamos programos UML panaudos atvejų diagrama



2.6 pav. Sistemos veiklos diagrama vaizduojanti skirtingus žingsnius ir lygiagrečių procesų komunikaciją žaidimo agento apmokymo metu

2.2. Projekto planas

Išanalizavus literatūroje esamus giliojo mokymosi algoritmus buvo identifikuoti pagrindiniai iššūkiai, trukdantys šių algoritmų platesniam panaudojimui. Pagal tai nuspręsta tobulinti duomenų kiekio ir mokymosi efektyvumo problemą. Siekiant, kad rezultatus būtų galima taikyti platesniam užduočių spektrui, buvo pasirinkta naudoti algoritmą, kuris gerai veikia tiek su diskrečiomis išvestim, tiek su tolydžioms. Vadovaujantis literatūros analizėje surinktai duomenimis buvo pasirinkta naudoti PPO (ang. Proximal policy optimization) [10] kaip bazinį variantą, su kuriuo bus atliekami modifikacijų bandymai. Norint įvertinti ar atliktos modifikacijos naudingos, iš pradžių reikia nustatyti kokie rezultatai gali būti pasiekti naudojant originalų variantą. Kad tai išsiaiškinti, bus panaudotas rlLib karkasas lygiagrečioms simuliacijoms su pasirinkta žaidimo Doom simuliavimo biblioteka VizDoom. PPO algoritmo hiperparametrai ir tinklo architektūra sukonfigūruoti taip, kaip aprašyta jo straipsnyje. Palyginimą galima matyti 2.3 lentelė. Kad įsitikintume teisingu algoritmo veikimu, algoritmą apmokėme su vienu iš straipsnyje tyrinėtų „*Atari*“ žaidimų „Ms PacMan“. Jame po 5 mln. simuliacijos kadrų algoritmas pasiekė panašius rezultatus į rodomus PPO straipsnyje. Tačiau pasirinktas žaidimas turi esminį skirtumą nuo aprašytų literatūroje tame, kad jis yra 3d o ne 2d tipo, kas neleidžia algoritmui vienu metu matyti visos žaidimo būsenos. Tai neleidžia PPO išmokti VizDoom žaidimo ir reikalauja papildomų architektūrinių pakeitimų. Remiantis kituose literatūros straipsniuose naudojamais sprendimais, pasirinkta įterpti papildomą LSTM [13] sluoksny tarp konvoliucinių ir likusių strategijos ir vertės funkcijų sluoksnių. Tai tinklui padeda išsaugoti papildomos informacijos vidinėje būsenoje, remiantis anksčiau matytais vaizdais. Toliau naudojantis algoritmu su šia tinklo architektūra bus siekiama surasti optimalius hiperparametrus, su kuriais tinklui pavyksta išmokti žaisti žaidimą. Hiperparametrų optimizavimas yra svarbus žingsnis giliajame skatinamajame mokymesi, nes daugelis algoritmų yra nestabilūs ir dažnai nesimoko, jei nustatyti parametrai yra netinkami pasirinktai aplinkai. Nustačius kokių rezultatų galima pasiekti su originaliu algoritmu, bus išbandomos įvairios modifikacijos su neuroninio tinklo architektūra ir treniravimo procesu. Jomis bus siekiama surinkti daugiau taškų, nei su ankstesnėmis versijomis, atlikus treniravimą su 10 mln. žaidimo kadrų.

2.3. Nemodifikuoto algoritmo struktūra ir rezultatai

PPO straipsnyje aprašomas neuroninis tinklas „*Atari*“ žaidimams naudoja tokią pat architektūrą taip ir A3C algoritmas. Prieš perduodant duomenis neuroniniam tinklui atliekama keletas išankstinio apdorojimo žingsnių: pirma agentas bendrauja su žaidimu tik kas 4 kadrą, nes nėra reikalo keisti veiksmus kiekviename žingsnyje, tuomet gauti RGB kadrai konvertuojami į vieno kanalo pilkus paveikslėlius, kad sumažėtų užimama vieta, taip pat visi vaizdai normalizuojami į 84x84 pikselių dydžio kadrus, galiausiai neuroniniam tinklui kiekviename naujame žingsnyje, paduodami paskutiniai 4 gauti kadrai (slenkančio lango principu) tai leidžia nuspėti objektų judėjimą scenoje, kas nėra matoma turint tik viena kardą. Toliau įvestims apdoroti naudojamas neuroninis tinklas su lentelėse **2.1 lentelė** ir **2.2 lentelė** nurodytomis architektūromis ir **2.3 lentelė** aprašytais hiperparametrais.

2.1 lentelė Nemodifikuota neuroninio tinklo struktūra naudota PPO algoritmo straipsnyje

Kodavimas		
Tipas	Aktyvacija	Parametrai
Knvoliucinis 2D	ReLU	Rinkinio dydis, 4 kadrai, 16 kanalų, 8x8 filtras, 4 filtro

				postūmis	
Konvoliucinis 2D		ReLU		Įvesties kanalai 16, išvesties kanalai 32, 4x4 filtras , 2 filtro postūmis	
Pilnai sujungtas		ReLU		256 išvesčių sluoksnis	
Strategijos funkcija			Vertės funkcija		
Tipas	Aktyvacija	Parametrai	Tipas	Aktyvacija	Parametrai
Pilnai sujungtas		256-> Galimų veiksmų kiekis	Pilnai sujungtas		256 -> 1

2.2 lentelė. Modifikuota tinklo struktūra su LSTM sluoksniu reikalingu 3D aplinkai išmokti

Kodavimas					
Tipas		Aktyvacija		Parametrai	
Knvoliucinis 2D		ReLU		Rinkinio dydis, 4 kadrai, 16 kanalų, 8x8 filtras, 4 filtro postūmis	
Konvoliucinis 2D		ReLU		Įvesties kanalai 16, išvesties kanalai 32, 4x4 filtras , 2 filtro postūmis	
Pilnai sujungtas		ReLU		256 išvesčių sluoksnis	
LSTM		Tanh, Sigmoid		256 dydžio vidinis būsenos vektorius	
Strategijos funkcija			Vertės funkcija		
Tipas	Aktyvacija	Parametrai	Tipas	Aktyvacija	Parametrai
Pilnai sujungtas		256-> Galimų veiksmų kiekis	Pilnai sujungtas		256 -> 1

2.3 lentelė. Straipsnyje aprašytų ir naudotų PPO algoritmo hiperparametų palyginimas. Pagrindinis skirtumas – didesnis lygiagrečių simuliacijų kiekis naudojamame variante. α daugiklis mažinamas nuo 1 iki 0 tiesiškai viso treniravimo metu

Hiperparametras	**PPO straipsnyje**	**Naudotos reikšmės**
Rinkinio dydis	128*8(aktoriai)	10000
Mokymosi greitis	2,5*10-4* α	2,5*10-4*α
Duomenų rinkinio epochos	3	8
Mini-rinkinio dydis	32*8	1000
Gamma	0,99	0,99
GAE lambda	0,95	0,95
Lygiagretūs aktoriai	8	12*16
Apribojimas epsilon	0,1* α	0,1* α
Vertės funkcijos koeficientas	1	1
Entropijos koeficientas	0,01	0,01

Atliekant pirminius algoritmo realizacijos testavimo eksperimentus pastebėta kad, neužtenka nustatyti tik parametrų iš algoritmo straipsniu, nes dėl skirtingo gautų taškų kiekio vertės funkcija nesugeba išmokti būsenos vertės, o tai neleidžia agentu labiau tobulėti praėjus dar tik 30 iteracijų

(300k žaidimo kadrų). Siekiant kad tinklo mokymasis veiktų visose jo atšakose, ir algoritmas galėtų pasiekti geresnių rezultatų, buvo atlikta keletas mokymąsi stabilizuojančių modifikacijų.

Gradientų vektoriaus ilgio apribojimas iki 1. Tai neleidžia tinklo svorių keisti labai didelėmis reikšmėmis per vieną iteraciją, kas gali nutikti kai agento surenkami duomenys yra nepastovūs ir kinta kiekvienoje iteracijoje. Naudojant tokias aktyvacijos funkcijas kai ReLU, Sigmoid ir Tanh kyla grėsmė kad vienas didelis tinklo svorių atnaujinimas privers tam tikrus neuronus įgyti reikšmes kur aktyvacijos funkcijos išvestinė yra labai maža arba (ReLU atveju) lygi 0, kad neleidžia svoriams toliau kisti ir mokyti sekančiose iteracijose. Tai vadinama mirusių neuronų problema, o jos žala yra sumažėjąs tinklo ekspresyvumas ir galimybė išmokti sudėtingus sąryšius.

Mokymuisi didelę reikšmę turėjo nuostolio funkcijos apribojimas. Atliekant bandymus su įvairiomis tinklo konfigūracijomis pastebėta, kad algoritmas pasiekia daug geresnių rezultatų treniravimo pabaigoje jei vertės funkcijos klaidoms pritaikomas reikšmių apribojimas (ang. *clipping*). Skaičiuojant vertės funkcijos nuostolio vertę (ang. *Loss value*) iš pradžių apskaičiuojamos kiekvieno žingsnio vertės funkcijos klaidos, šios vertės ir yra apribojamos, kad nebūtų didesnės nei nustatyta. Tai gelbsti nuo pasitaikančių labai didelių klaidos verčių kurios iškraipo visą vidurkį ir stabdo mokymąsi. Eksperimentiškai nustatyta kad gera šio parametro reikšmė yra ~1.

Atliekant algoritmo testavimą su „Ms Pacman“ žaidimu taip pat pastebėta kad algoritmas pasiekia ~2 kartus geresnių rezultatu pakoregavus PPO algoritmo iteracinio mokymo naudojant duomenų mini–rinkinius parametrus. Šiame darbe naudojama realizacija turi didesnį žaidimų simuliacijos lygiagretumą ir surenka daugiau duomenų prieš atliekant tinklo svorių atnaujinimą. Dėl to pakoregavus parametrus taip kad duomenys būtų skaidomi į mažesnius rinkinius ir panaudojami gradientinem optimizavime daugiau kartų, neuroniniam tinklui pavyko įsisavinti daugiau informacijos iš turimų duomenų. Paveikslėlyje **2.7 pav** galima matyti skirtingų parametrų poveikį galutiniam rezultatui.

Metrics			Parameters		
episode_reward_max	episode_reward_mean	episode_reward_min	training/num_sgd_iter	training/sgd_minibatch_size	training/vf_clip_param
2130	912.4	540	8	1000	1
2090	635.8	230	3	2500	1

2.7 pav. Algoritmo surinkti taškų kiekiai po apmokymo 5M kadrų. Pirmoje eilutėje matoma kad rezultatai geresni naudojant 8 iteracijas su tai pačiais duomenimis vietoje 3 ir suskaidant duomenis į mažesnius mini-rinkinius skaičiuojant gradientus

2.4. Siūlomi sprendimai

Remiantis literatūroje aprašytais RL sprendimais, naujovėmis giliojo mokymosi srityje, bei problemomis su kuriomis buvo susidurta algoritmo realizavimo metu, buvo nutarta atlikti eksperimentus su keletu potencialiai naudingų modifikacijų.

Atsižvelgus į tai kad RL algoritmai pasižymi nestabiliu mokymusi, būtų naudinga minimizuoti kuo daugiau tai sukeliančių priežasčių arba to padarinių. Kadangi naudoti algoritmai remiasi neuroniniais tinklai kaip funkcijų aproksimatoriais, galima pasinaudoti keletu patobulinimų kurie

naudojami kituose neuroninių tinklų uždaviniuose. Vienas jų yra labiau „neuronų mirčiai“ atsparios aktyvacijos funkcijos, kurios nepraranda mokymosi spartos esant dideliems svyravimams gradientuose ir nuostolio funkcijose. Kaip alternatyva dabar naudojamai ReLU funkcijai gali būti naudojamos LeakyReLU, ELU, Symlog arba panašios funkcijos. Kita giliųjų neuroninių tinklo mokymo problema yra lėtesnis gilesnių sluoksnių mokymasis susijęs su nykstančiais gradientais (ang. *Vanishing gradients*). Kaip galima matyti paveikslėlyje **2.8 pav**, gilesnių sluoksnių tokių kaip pirmas ir antras konvoliuciniai ir pirmas pilnai sujungtas sluoksnis, reikšmes atnaujinimas daug lėčiau (nuo 100 ik 1000 kartų) nei tų kurie yra arčiau tinklo išvesčių kur pradedamas skaičiuoti gradientas pagal nuostolio funkcijos reikšmę. Šiai problemai spręsti naudojami rinkinio normalizavimo (ang. *Batch normalization*) [15] ir modernesnis ir tinkamesnis sluoksnio normalizavimas (ang. *Layer normalization* [14]. Šie sprendimai įterpiami į tinklą kaip papildomi sluoksniai kurie standartizuoja neuronų aktyvacijas į standartinį pasiskirstymą kuris padeda suvienodinti gradientų dydžius skirtinguose sluoksniuose ir palengvina tinklo treniravimą [14].

2.8 pav. Neuronų svorių vertės santykis su atnaujinimo pokyčiu. X ašis rodo apmokymo iteraciją, Y ašyje rodomas santykis logaritminėje skalėje kiek pasikeis neuronų svoriai skirtinguose sluoksniuose. -6 rodo kad svoriai tame sluoksnyje pasikeis 1 milijonąja dalimi

Strategijos optimizavimu grįsti metodai tokie kaip PPO pasižymi prastu duomenų panaudojimo efektyvumu nes jiems apmokinti tinkami tink tie duomenys kurie buvo surinkti naudojant optimizuojamą strategiją. Tai neleidžia panaudoti tokių metodų kaip patirties pakartojimas kurie naudojami Q funkcija gystuose metoduose ir leidžia optimizuoti tinklą su bet kokiais turimais duomenim. Tačiau ne visas optimizuojamas tinklas naudojamas strategijos ar vertės funkcijai aproksimuoti. Bene pusė tinklo sluoksnių yra naudojami vaizdinių duomenų suspaudimui į kompaktiškesnį būseną apibūdinantį vektorių. Ši dalis neturi specialių reikalavimų ir gali būti apmokoma su visais turimais duomenimis. Kad būtų galima tai atlikti nenaudojant strategijos ir vertės funkcijų išvesčių reikalinga papildoma tinklo išvestis pagal kurią būtų galima skaičiuoti nuostolio funkciją ir gradientą. Šiam tikslui puikiai tinka palčiai naudojama autoenkoderių architektūrų klasė. Esama keletas šios architektūros modifikacijų kurios padeda sukurti tolydesnes ir lengviau interpretuojamas būsenos vektorių reprezentacijas. Labiausiai perspektyvia atrodo beta–VAE variacija [23]. Vienas jos pranašumas yra tolydi vidinės reprezentacijos erdve, t. y. panašios vidinio vektoriaus reikšmės generuoja panašius žaidimo būsenos kadrus. Kitas pranašumas yra tai

kad vidinio vektoriaus kintamieji yra dekoreliuoti kur kiekvienas vidinio vektoriaus skaliaras atsakingas už skirtingą būsenos požymį. Šiai modifikacijai atlikti tinkle reikia pridėti papildomą atšaką kuri iš suspausto vektoriaus bandytų vėl atkuri pikselių vaizdą kuris buvo duotas tinklui pradžioje. Atšakos treniravimas gali būti atliekamas apie pradinės nuostolio funkcijos pridedant papildomą sudedamąją dalį, arba naudojant atskirą optimizatorių kuris veiktų lygiagrečiai su pirmajam [24] [23].

Atliekant pirminius eksperimentus pastebėta, kad vertės funkcijos klaidos yra gerokai didesnės nei strategijos funkcijos, kas trukdo agento gebėjimui įvertinti veiksmų naudą tolimesnėje ateityje. Būsenos vertės funkciją V galima pakeisti būsenos–veiksmo vertės funkcija Q. Ji nepriklauso nuo strategijos, todėl gali būti apmokoma su bet kokiais duomenimis panaudojant patirties pakartojimą. Kaip rodo literatūros šaltiniai, tai leidžia pasiekti didžiausią duomenų panaudojimo efektyvumą. Q funkcija gali būti perskaičiuota į V pagal išraišką $V^{\pi}(s) = E_a[\mathrm{Q(s,)}^* \pi\mathrm{(s,)}]$Tai leistų pakeisti tik vertės funkciją, o kitos algoritmo dalys išliktų nepakitusios.

Remiantis pavyzdžiais iš kitų mašininio mokymosi metodų tokių kaip atsitiktinis miškas, galima pastebėti kad keleto mažų sprendimų modelių rinkinys pasiekia geresnių rezultatų nei vienas mašininio mokymosi modelis. Kelių mažesnių modelių rinkinys vadinamas ansambliniu mokymusi. Tai galima pritaikyti būsimo tinklo architektūrai kai sprendimai priimami pasitelkiant keleto išvesčių vidurkį. Vietoje dabar naudojamų vienos vertės ir vienos strategijos funkcijų atšakų galima naudoti keletą atšakų kurios vėliau naudojamos vidurkiui išvesti.

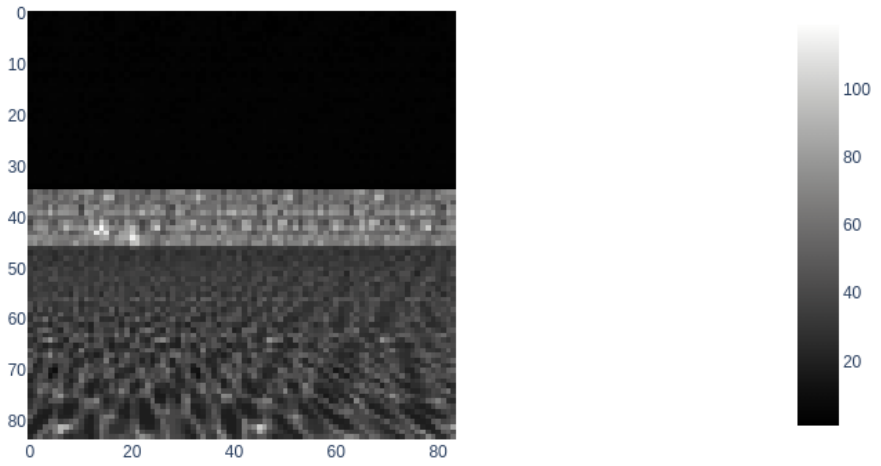
2.5. Kokybės kriterijai

Pasirinktas, giliojo skatinamojo mokymosi algoritmas, apmokomas žaisti Doom žaidimo lygį iš VizDoom bibliotekos naudojant tik vaizdiniais duomenis t. y. RGB pikselių matricą. Algoritmo kokybė vertinama pagal šiuos kriterijus:

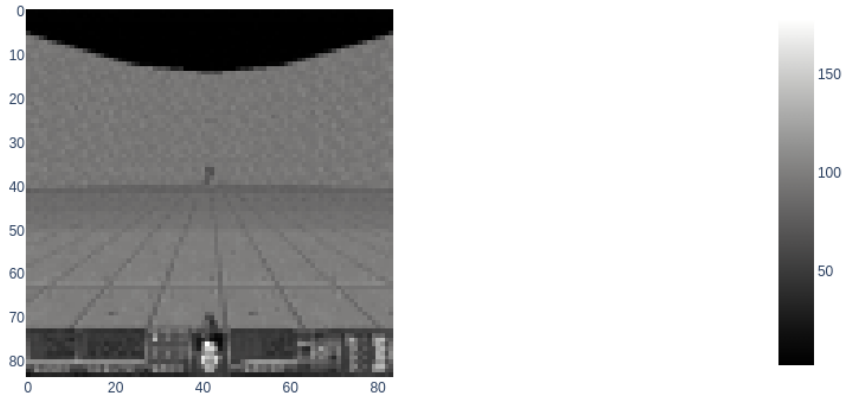
- Modifikuotas algoritmas turi surinkti nemažesnį taškų kiekį nei pasirinktas nemodifikuotas algoritmas su optimaliais hiperparametrais.
- Modifikacijos turi patobulinti algoritmo efektyvumą simuliacijos duomenų atžvilgiu t. y. agentui reikės mažiau sąveikauti su aplinka.
- Modifikacijos turi būti bendrinės, nenaudoti konkrečiam pasirinktam žaidimui pritaikytų euristinių metodų, ir potencialiai patobulinti algoritmo duomenų efektyvumą ir kitose aplinkose.

3. Paskirstytos strategijos ir VAE integravimo eksperimentai

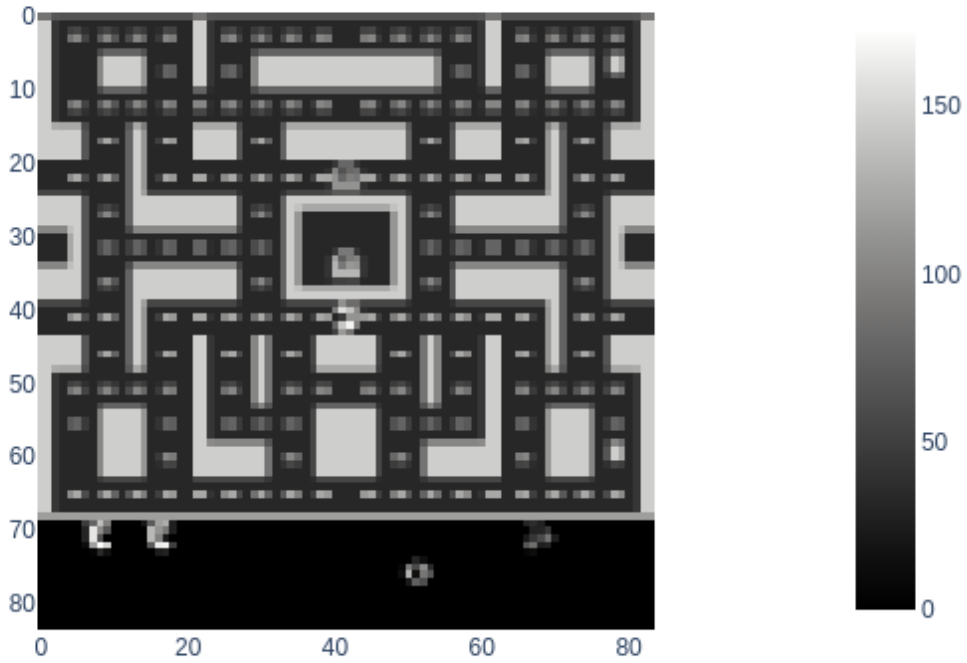
Šiame skyriuje nagrinėjamos neuroninio tinklo architektūros ir treniravimo strategijos modifikacijos kuriomis siekiama patobulinti giliojo skatinamojo mokymosi algoritmų efektyvumą duomenų atžvilgiu. Iš pradžių aprašomos atliktos modifikacijos ir jų veikimas, toliau pateikiami rezultatai gauti apmokant algoritmus skirtinguose žaidimuose su skirtingais parametrais, galiausiai aptariami gauti rezultatai. Pagrindiniai eksperimentai atlikti bet po 3 kartus su tomis pačiomis konfigūracijomis. Algoritmai apmokomi naudojant skirtingo tipo aplinkas: „VizDoom“ „Deffend the center“ (Tolydžių įvesčių kontrolė – kompiuterio pelė), „VizDoom“ „Health gathering“ (navigavimas 3D tipo aplinkoje), „Ms Pacman“ (2D tipo aplinka) **3.1 pav**, **3.2 pav**, **3.3 pav**. Eksperimentai atlikti su standartiniu PPO tinklu, padidintu standartinio tinklo variantu, 2 modifikacijomis su atminties sluoksniais LSTM ir GRU, VAE pagrįsta architektūra, bei nauja pasiūlyta struktūra su keliomis strategijos tinklo atšakomis. Atliekant pirminius testus pastebėta, kad algoritmų treniravimo procesas yra itin nestabilus ir jautrus mažiems hiperparametrų pokyčiams ir kad šią problemą galima efektyviai išspręsti naudojant sluoksnio normalizavimo sluoksnius prieš kiekvieną aktyvacijos funkciją. Su šiuo pakeitimu beveik visi treniravimo eksperimentai su skirtingais hiperparametrais sugeba išmokti gerai veikiančias agento valdymo strategijas, todėl visi čia pateikiami eksperimentai naudoja šį pakeitimą. Taip pat balansuojant treniravimo laiką ir eksperimentų kiekį kiekvienas algoritmas buvo apmokomas su 10M žaidimo sugeneruotų kadrų. Agento veiksmų pavyzdžius galima matyti prieduose: **priedas. Agento veikimo žaidime „VizDoom Health gathering“ pavyzdys** ir **priedas. Agento veikimo žaidime „Ms Pacman“ pavyzdys.**



3.1 pav. „VizDoom“ „Health gathering“ žaidimo vaizdas kurį mato algoritmas. Vaizdo dydis yra normalizuotas į 84x84 pikselius iš vienos spalvos



3.2 pav. „VizDoom“ „Defend the center“ žaidimo vaizdas kurį mato algoritmas. Vaizdo dydis yra normalizuotas į 84x84 pikselius iš vienos spalvos

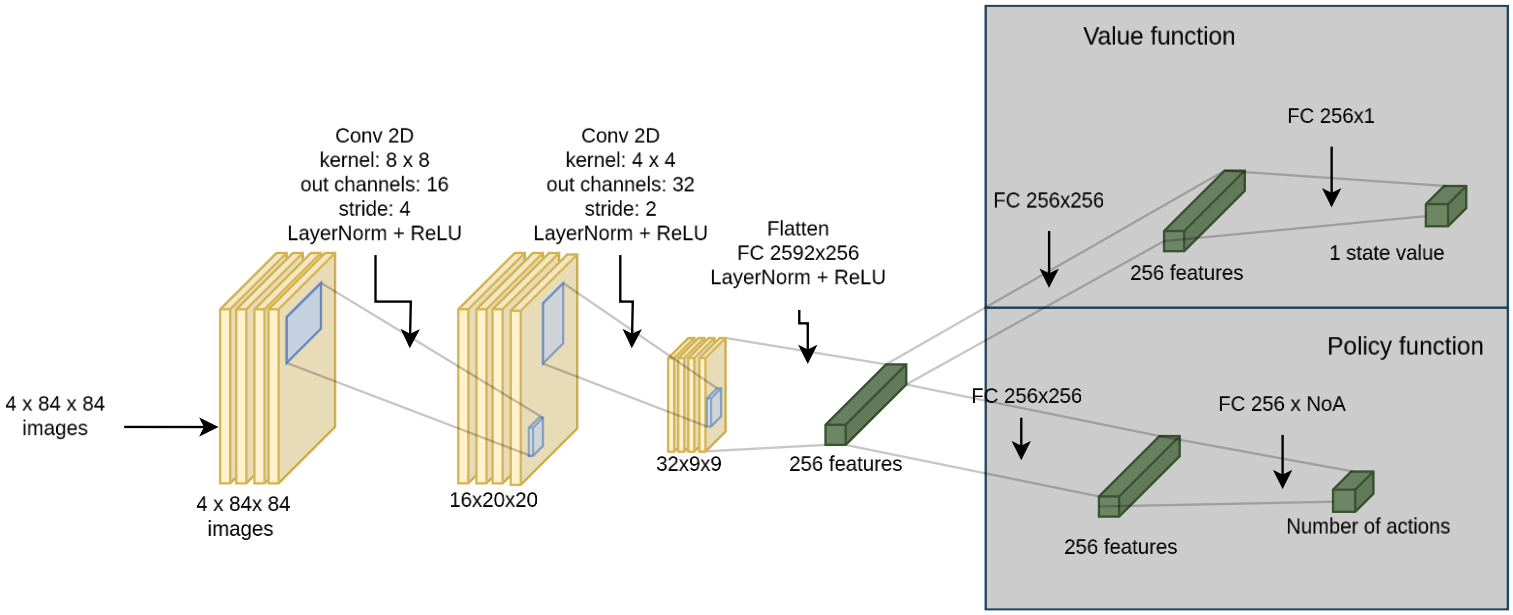


3.3 pav. Žaidimo „Ms Pacman“ vaizdas kurį mato algoritmas. Vaizdo dydis yra normalizuotas į 84x84 pikselius iš vienos spalvos

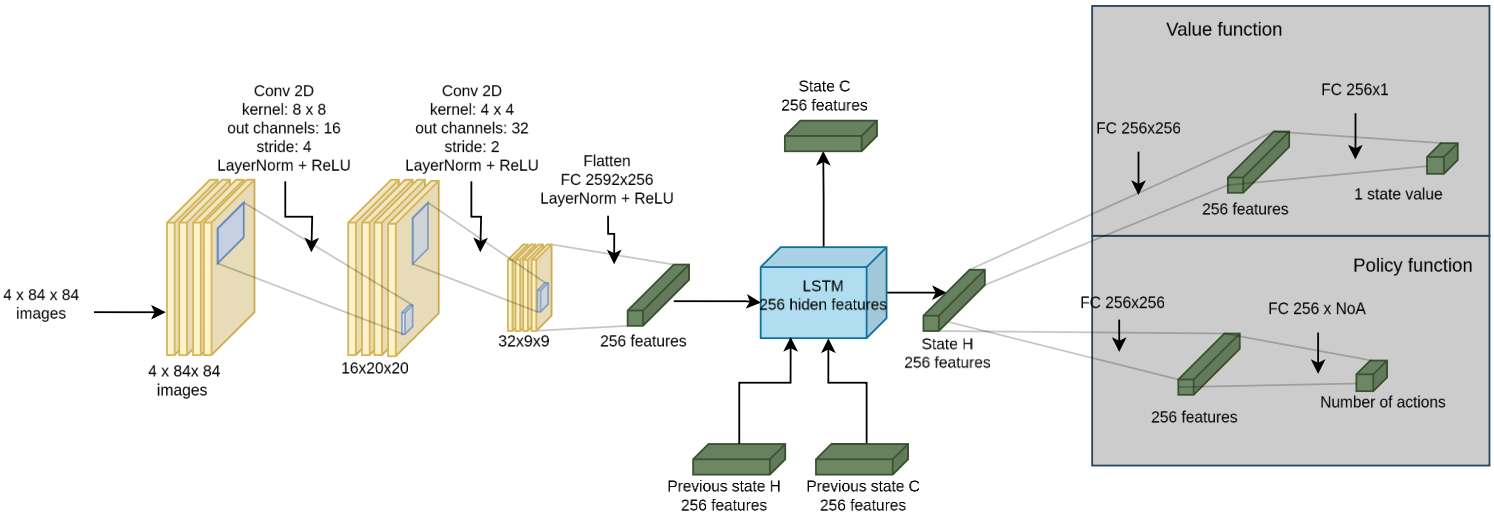
3.1. Neuroninių tinklų architektūros

Prieš atliekant algoritmų modifikacijas reikia išmatuoti kokie yra baziniai rezultatai, bei kokią architektūrą tinkamiausia laikyti bazine, turint omenyje pasirinktas aplinkas. Tinklus papildžius sluoksnio normalizavimo sluoksniu, standartiniu tinklu buvo laikomas **3.4 pav**. Kadangi 2 iš 3 naudojamų žaidimų yra 3D tipo (t.y. vienu metu matoma tik dalis scenos) buvo siekiama

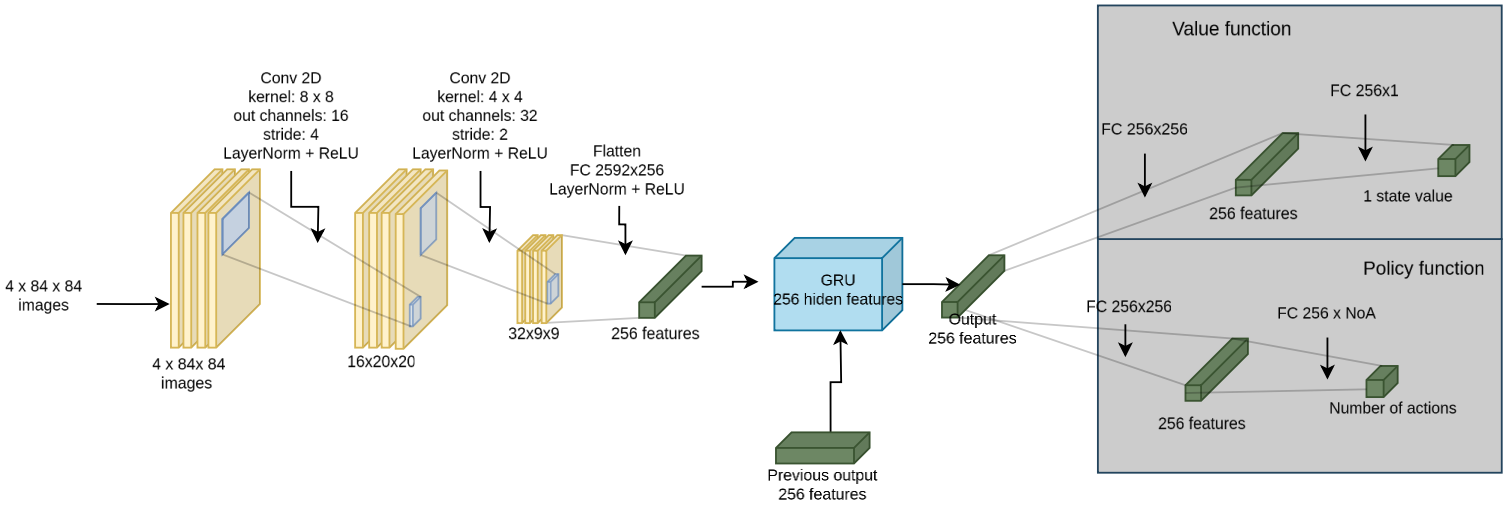
išsiaiškinti ar rekurentiniai neuroniniai tinklai būtų tinkamesni atstovauti šios užduoties pradinį lygį ir būti naudojami kaip pagrindas tolimesnėms modifikacijoms. LSTM ir GRU naudojantys tinklai pavaizduoti **3.5 pav**, 3.6 pav. Keičiant tinklų architektūras, kai kurios architektūros neišvengiamai turėjo turėti daugiau „neuronų“ – apmokomų parametrų, ko pasėkoje jie įgauna nesąžiningą pranašumą, nes tai jiems leidžia tiksliau aproksimuoti sudėtingesnes funkcijas. Siekiant išsiaiškinti šio pokyčio įtaką rezultatui, buvo atlikti eksperimentai su padidinta standartinio tinklo versija kuri siekė kiek galima minimizuoti parametrų kiekio skirtumus **3.7 pav**.



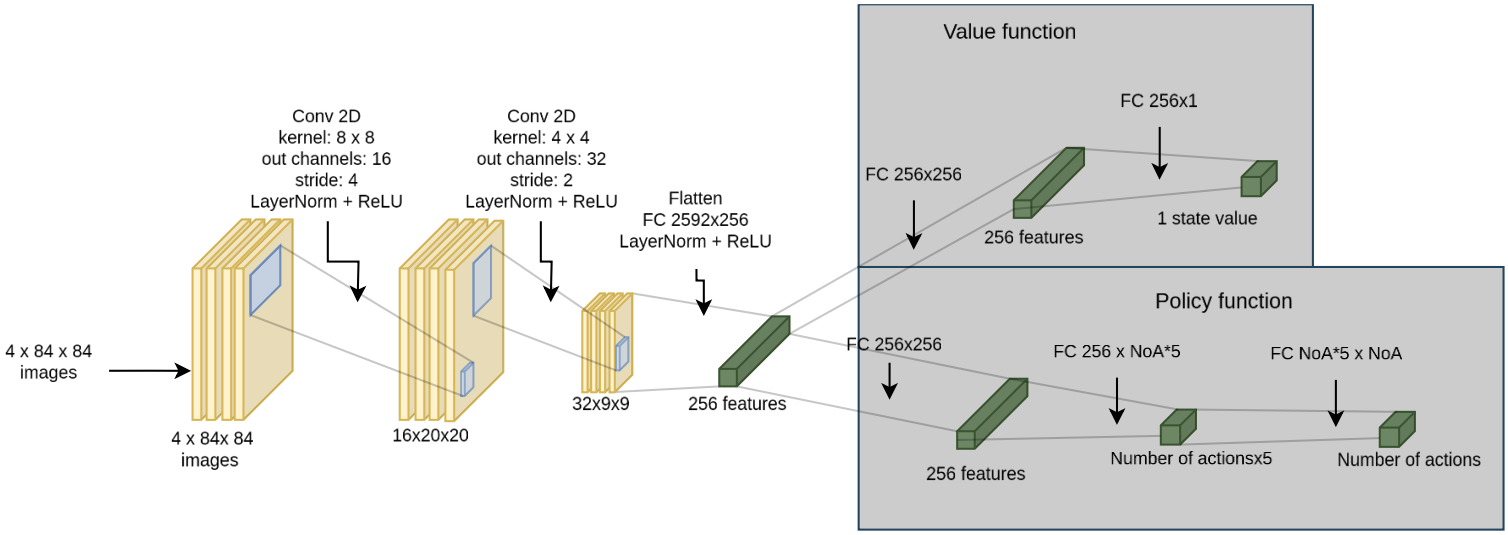
3.4 pav. Nemodifikuota PPO algoritmo neuroninio tinklo versija, tokia kaip naudojama autorių straipsnyje + sluoksnių normalizavimas. „NoA“ žymi skirtingų veiksmų skaičių, kuris priklauso nuo žaidimo



3.5 pav. Neuroninis tinklus su LSTM rekurentiniu (atminties) sluoksniu. Išvestys aukštyn ir iš apačios reiškia masyvus perduodamus į kitą žingsnį laike



3.6 pav. Neuroninis tinklas su GRU rekurentiniu (atminties) sluoksniu. Įvestis iš apačios žymi GRU išvesties masyvo reikšmę iš parėjusio žingsnio

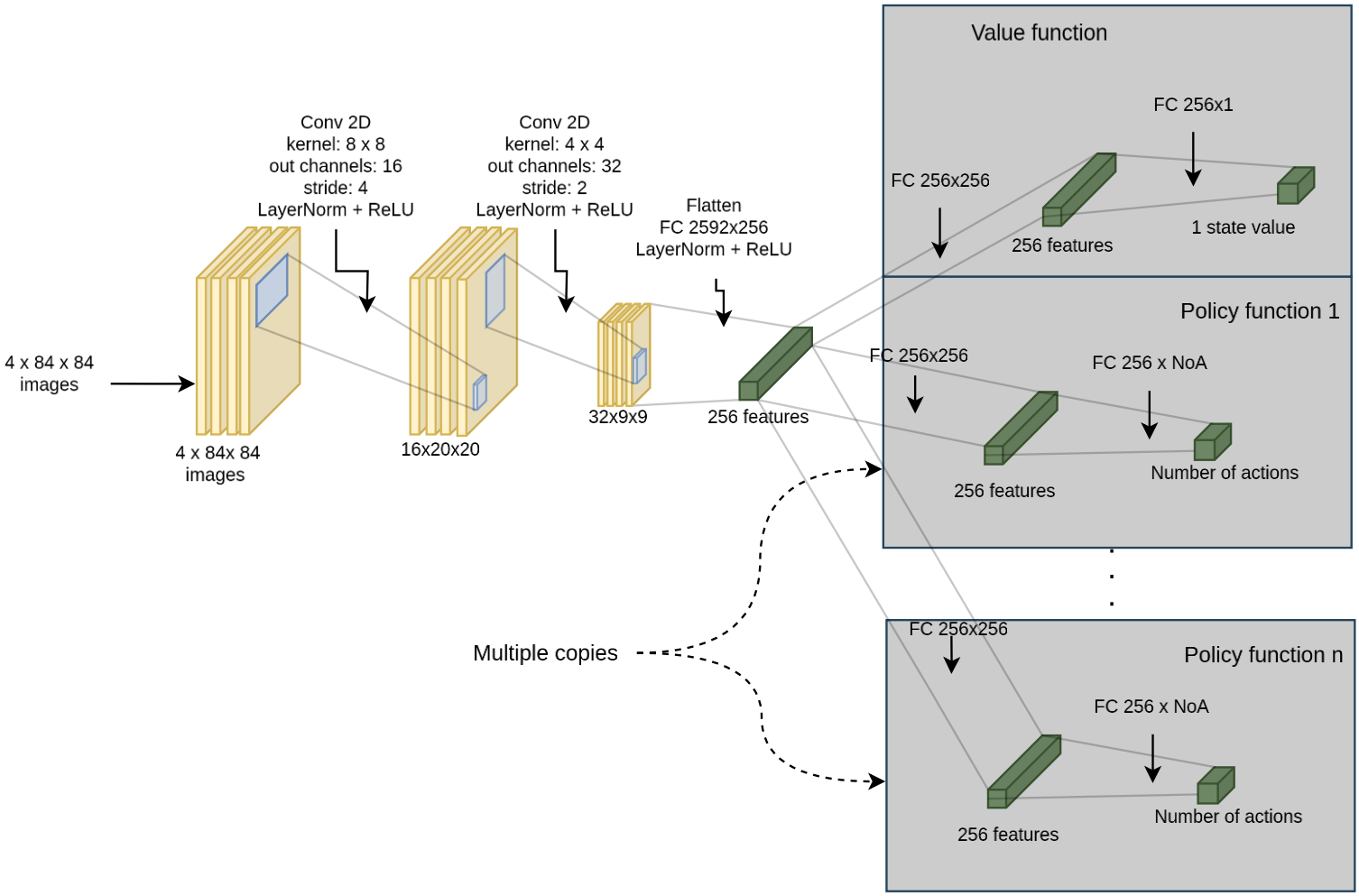


3.7 pav. Padidinta standartinio tinklo versija, strategijos funkcijai pridėtas papildomas sluoksnis taip kad būtų minimizuotas parametrų skirtumas tarp šio ir paskirstytos strategijos tinklo versijos. „NoA“ žymi galimų skirtingų veiksmų kiekį, kuris priklauso nuo žaidimo

3.1.1. Paskirstyta strategijos funkcija

Šiame darbe siūloma modifikacija remiasi idėjomis iš atsitiktinio miško algoritmo (ang. Random Forest), kai naudojama keletas paprastų mašininio mokymosi klasifikatorių vidurkių. Išvedamas vidurkis pašalina individualias klasifikatorių variacijas atsirandančias dėl atsitiktinės iniciacijos, kas duoda geresnius galutinius rezultatus. Šiuo atveju kai naudojamas neuroninis tinklas, atskirus klasifikatorius atitiktų neuroninio tinklo išsišakojimas, tiksliau strategijos funkcijos dalies keletas kopijų **3.8 pav**. Kiekvienos treniravimo iteracijos metu gaunamas atlygis kuris sustiprina arba susilpnina pasirinkto veiksmo tikimybę kitą kartą. Kadangi atlygis yra vienas visam tinklui, tai ir gradientas yra vienodas visoms atšakoms, kas lemia, kad tokiu atveju visos strategijos atšakos išmoktų tą pačią strategiją. Kad to išvengtume, pasinaudojome reparametrizavimo idėja iš variacinio autoenkoderio, kuri leidžia į neuroninio tinklo skaičiavimus įmaišyti atsitiktinumą tuo

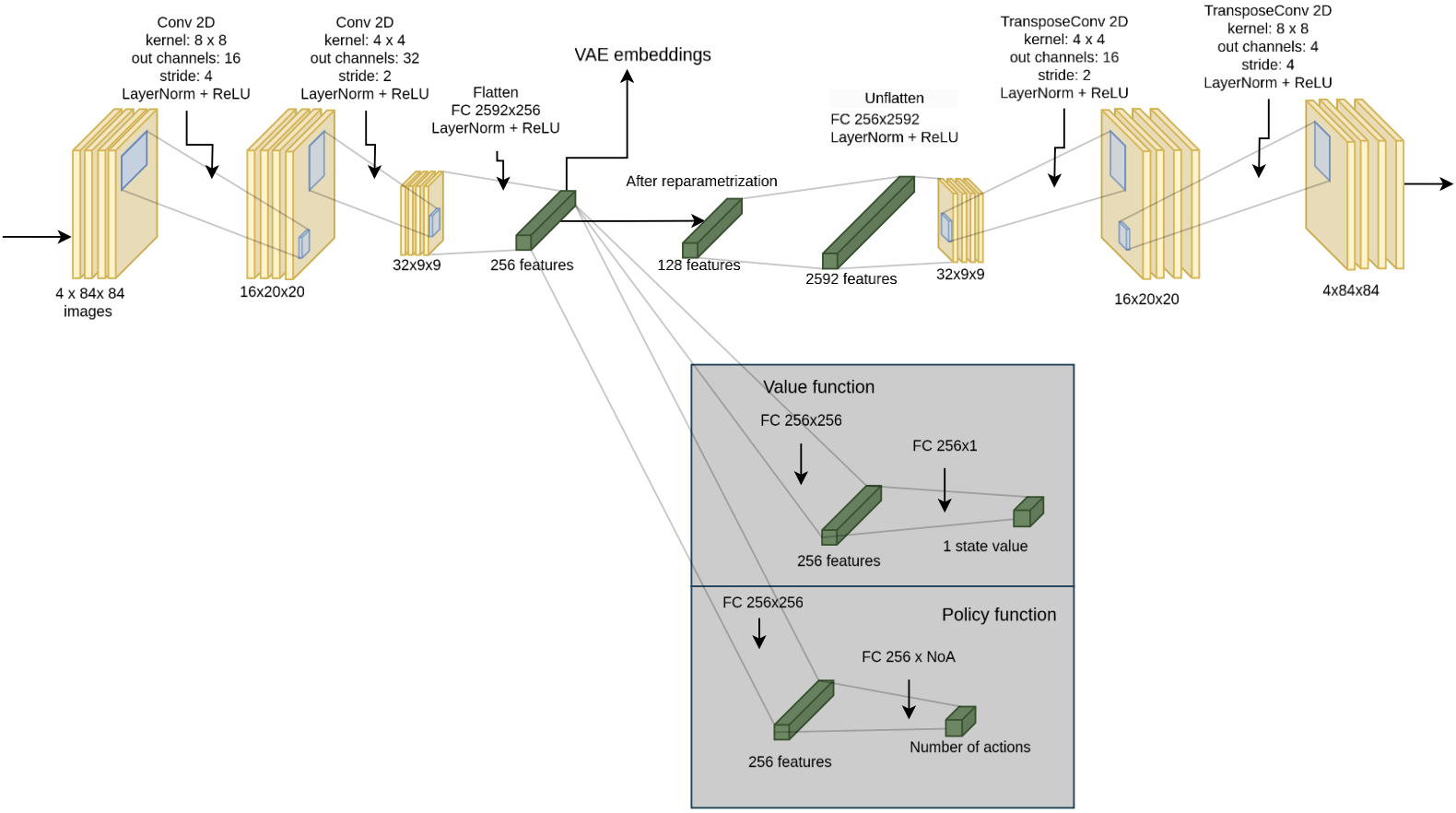
tarpu išlaikant galimybę operacijas diferencijuoti. Galimybė diferencijuoti operacijas yra svarbi norint apskaičiuoti gradientus ir treniruoti tinklo parametrus pagal duomenis. Ši idėja pritaikoma siūlomame sprendime, suteikiant atsitiktinius svarbumo svorius atskiroms strategijos atšakoms kiekviename žingsnyje. Tai lemia, kad skirtingos atšakos apmokomos su skirtingais duomenimis ir daro skirtingas klaidas todėl jų strategijos nėra identiškos. Treniravimo pradžioje bendras tinklas sudarytas iš atskirų strategijų atlieka labiau atsitiktinius veiksmus, galima teigti kad tai taip pat paskatina aplinkos tyrinėjimą. Atsitiktiniai svoriai suteikiami pagal natūralųjį pasiskirstymą su tam tikra dispersija, t.y. viena ar kelios atšakos lemia didžiąją dalį galutinio sprendimo (~60%). Visų svorių suma susideda į 1, tam pasitelkiama Softmax transformacija. Atlikus pirmuosius eksperimentus pastebėta, kad nuolatinis strategijų maišymas viso treniravimo metu neduoda gerų rezultatų, po keliasdešimties pirmųjų epochų mokymasis sustoja, nors pirmosiomis epochomis stebima sparti pažanga. Kaip šios problemos sprendimą įvedėme papildomą hiperparametrą kuris reguliuoja strategijų svarbos pasiskirstymą. Šis parametras tolydžiai mažinamas viso treniravimo metu pagal funkciją $\sigma = (1 - 0.1) * \left(\frac{T_{current}}{T_{max}}\right)^{D} * 0.1$, čia D – žymi mažėjimo greičio hiperparametrą, o T – dabartinį ir paskutinį treniravimo žingsnius. Tai lemia, kad laikui bėgant strategijoms duodamos vis lygesnės vertės ir pereinama prie paprasto aritmetinio vidurkio. Atliekant eksperimentus strategijų atšakų skaičius taip pat buvo reguliuojamas pagal hiperparametrą.



3.8 pav. Paskirstytos strategijos ir VAE integravimo eksperimentai.Paskirstytos strategijos funkcijos modifikacija. Sukuriama keletas identiškų atšakų

3.1.2. VAE grįstas vaizdo suspaudimas

Variaciniu autoenkoderiu grįsta architektūra pasižymi papildoma atšaka nuo suspaudimo dalies, kuri yra atvirkštinė suspaudimo dalies versija. Šis priedas stengiasi išmokti atkartoti pradinius duomenis, ko pasėkoje išmokdamas ir efektyvesnį duomenų suspaudimą. Variacinė dalis skiriasi ir tuo, kad naudojama papildoma reguliarizacija neuroninio tinklo viduryje, kuri priverčia suspaustąjį vektorių laikytis tam tikros struktūros t. y. vienas iš parametrinių pasiskirstymų. Šiuo atveju yra naudojamas Gauso skirstinys. VAE suspausti duomenys pasižymi tuo, kad minimalūs pakeitimai įvestims rezultatams turi tik minimalią įtaką rezultatui. Pilna atnaujinto tinklo architektūra gali būti matoma **3.9 pav**. Kadangi VAE dalies išvestys nedaro įtakos strategijos ir vertės funkcijų, jie gali būti treniruojami nepriklausomai ir su mažiau apribojimų. Pavyzdžiui ši dalis gali būti apmokoma su iš ankščiau surinktais duomenimis – patirties pakartojimas, o kadangi suspaudimo dalis yra bendra visam neuroniniam tinklui, turėtų paspartėti ir RL skirta tinklo dalis.



3.9 pav. Variaciniu autoenkoderiu grįsta architektūra. Šioje modifikacijoje atsiranda galimybė treniruoti vaizdo suspaudimo sluoksnius (pirmi CNN) per vaizdo rekonstrukcijos atšaką, taip pat pasitelkiant patirties pakartojimą – iš anksčiau sukauptus duomenis. (Galutinė versija naudojo SELU aktyvacijos funkcijas vietoje ReLU)

3.1.2.1. VAE integravimo iteracijos

Iš pradžių buvo bandoma treniravimą atlikti vienu metu, su naujausiu, ką tik surinktu duomenų rinkiniu. Šio mokymo metu suspaudimo dalis (enkoderis) gautų gradientus iš dekodavimo ir iš strategijos dalies. Deja atlikus pirmuosius bandymus pastebėta kad VAE konfigūracija neduoda daug naudos nes tinklas vienu metu turi mokintis daug funkcijų dėl ko nukenčia pagrindinės užduoties kokybė.

Kadangi naudojamas neuroninis tinklas yra mažas ir neturi daug sluoksnių – tikėtina kad treniravimosi metu jis išmoksta atpažinti labai specifines vaizdo dalis kurios svarbios užduočiai o sukurti universalią scenos reprezentaciją jam yra per sunku ir neoptimalu. Dėl to kai apmokoma pasitelkus gradientus iš dviejų atšakų jie vienas kitam trukdo. Kaip vieną iš šios problemos sprendimo būdų išbandėme treniravimą dvejais etapais – iš pradžių tinklas apmokomas bendrai kol parametrai dar yra visiškai atsitiktiniai (pirmosios 50-100 iteracijų), po to tinklas apmokomas kaip ir anksčiau – susitelkiant į strategiją ir taškus.

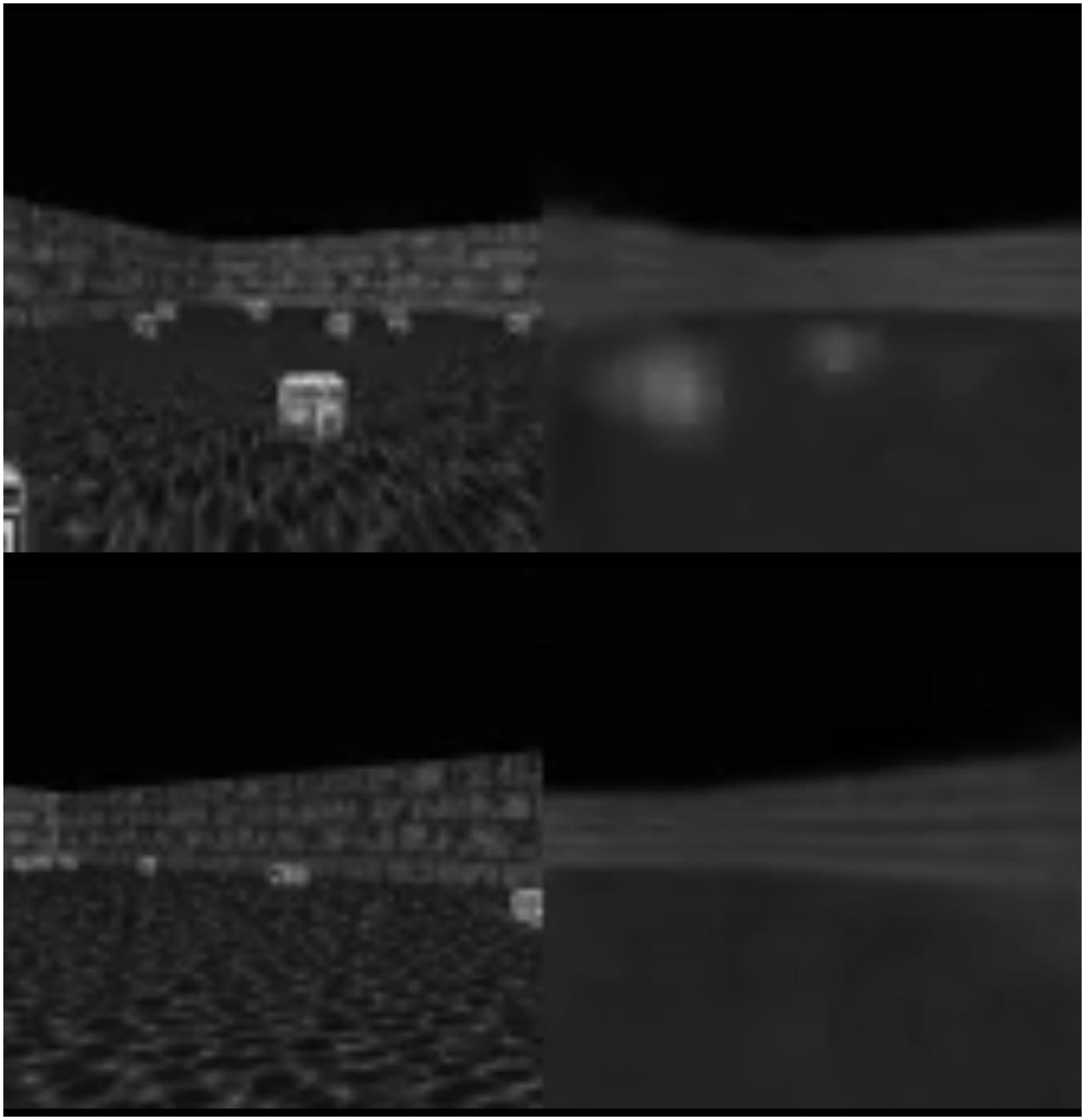
Sekančioje iteracijoje panaudojome pagrindinį VAE pranašumą – galimybę naudoti visus duomenis be apribojimų. Tam buvo panaudotas duomenų pakartojimo buferis kurio talpa 100000 kadrų (vienoje iteracijoje sukuriama 10000 naujų kadrų). Jame duomenys atnaujinami žiedinės eilės principu, kai naujausi įrašai pakeičia seniausius. Kai buferis prisipildo (t. y. po 10 pirmų iteracijų) pradedamas ir VAE mokymas, treniravimas atliekamas su 40000, atsitiktinai iš buferio parinktų, kadrų rinkiniu. Panašiai kaip PPO atveju, rinkinys panaudojamas keletą kartų tačiau su skirtingais hiperparametrais, juos galima matyti **3.1 lentelė**.

3.1 lentelė. VAE naudoti hiperparametrai

Hiperparametras	VAE reikšmė	PPO reikšmė
Mokymosi greitis	0,001	0,0003
Duomenų rinkinio dydis	40000	10000
Mini–rinkinio dydis	2048	1000
Duomenų rinkinio iteracijos	6	8
VAE–beta (Gauso skirstinio reguliarizacijos įtaka)	1	–

Deja, tai vis tiek nepadėjo pasiekti geresnių rezultatų arba pagreitinti bendro neuroninio tinklo treniravimo greičio. Tikėtina, kad problema yra kyla dėl skirtingų gradientų krypčių kurios atsiranda dėl skirtingų ir galbūt nesuderinamų VAE ir PPO optimizavimo tikslų. Norint kad optimizuojamos funkcijos neprieštarautų ir netrukdytų viena kitai, sekančiame bandyme gradientai buvo išjungti tuose PPO sluoksniuose kurie parsidengia su VEA. Šiuo būdu vaizdo suspaudimo dalis būtų apmokoma VAE metodu ir todėl galėtų panaudoti daugiau duomenų nei apmokant vien su PPO, tuo tarpu PPO reikėtų optimizuoti tik 2 paskutinius vertės ir strategijos funkcijų sluoksnius. Tačiau dėl to prasidėjo problemos su treniravimo seka. Pradžioje negalima greitai apmokinti VAE, nes agentas dar nėra išmokęs atlikti net minimalių racionalių veiksmų tokių kaip judėjimas, ko pasėkoje viso epizodo metu ekrane matomas tas pats vaizdas. Pradinio ir rekonstruoto vaizdo pavyzdžius galima matyti **3.10 pav.** . Dėl to kyla problema su treniravimo duomenų įvairove ir VAE išmoksta tik specifinės situacijos vaizdo reprezentaciją ir suspaudimą t. y. kai agentas žiūri pradine kryptimi iš pradinės pozicijos. Pasikeitus pozicijai arba krypčiai tinklas gražina klaidingą vaizdą arba triukšmą. Panaši problema gali kilti ir tada kai agentu leidžiama dalinai apmokyti strategijos funkciją pradžioje, nes kai vėliau pradedamas VAE treniravimas – drastiškai pasikeičia vaizdo reprezentacijos vektoriaus reikšmės, ko pasėkoje išmokta strategija tampa neveiksni arba klaidinga. Rezultatams vieną didžiausių įtakų turėjo aktyvacijos funkcijos pakeitimas iš ReLU į SELU, nes ji gali įgyti ir neigiamas reikšmes kas gali būti reikalinga VAE viduryje kai

modeliuojama Normaliojo skirstinio funkcija. Taip pat buvo išbandyta pridėti papildomų neuronų sluoksnių strategijos ir vertės funkcijoms, manant kad sumažintas PPO treniruojamų sluoksnių skaičius trukdo pasiekti geresnius rezultatus, tačiau reikšmingų pokyčių nepastebėta. Po skirtingų modifikacijų bandymų pastebėta, kad geriausius rezultatus ir stabiliausią mokymąsi pasiekė algoritmas kuris naudojo: SELU aktyvaciją, atskirą VAE ir strategijos treniravimą (strategijos gradientų išjungimas enkoderiui), ir VAE buvo treniruojamas pusę t. y. 500 pirmų iteracijų, vėliau atliekant tik strategijos tobulinimus. Iliustracijoje, **priedas. Ekrano vaizdo ir VAE rekonstruoto vaizdo palyginimas** galima matyti tikrasis ir rekonstruotas vaizdas kinta laike – žaidimo metu. Jame taip pat galima pastebėti, kad objektų pozicija scenoje šiek tiek skiriasi – panašu, kad VAE vaizdas atsilieka laike.



3.10 pav. Agento matomo vaizdo (kairėje) ir VAE rekonstruoto vaizdo (dešinėje) pavyzdys. Nors VAE rekonstruotas vaizdas ir nėra naudojamas galutiniai strategijai, jis gali būti naudojamas kaip apytikslis tinklo

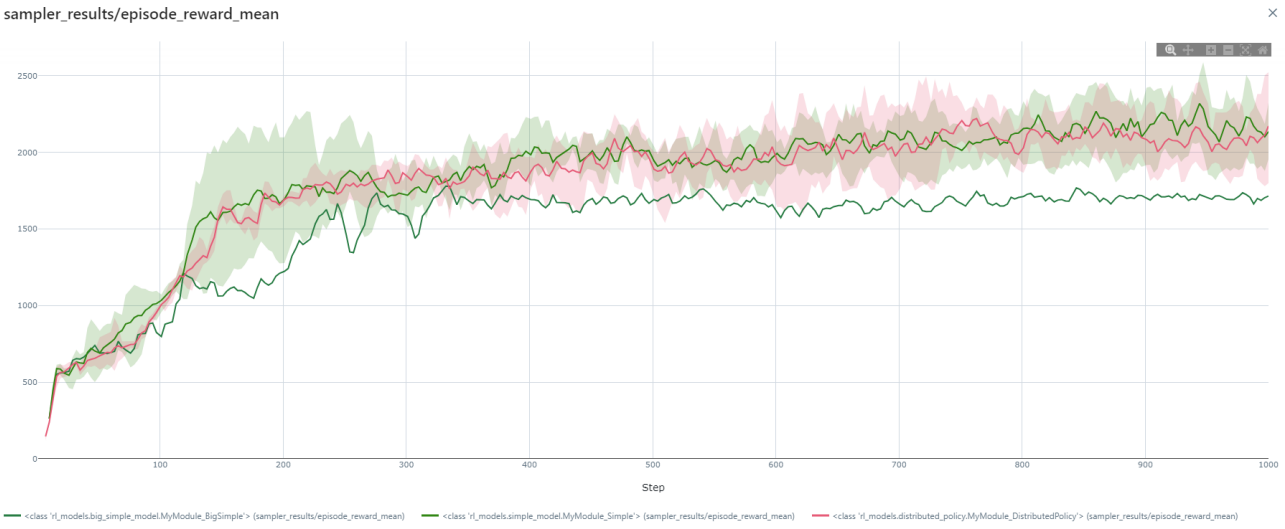
patikimumo įvertis. Iš šio vaizdo galima matyti, kad pagrindinė vaizdinė informacija yra atkuriama teisingai, todėl ir strategijos atšaka turi galimybę išmokti tinkamus veiksmus. Tačiau galima pastebėti, kad smulkesnės arba žaidimo erdvėje toliau esančios detalės rekonstruotame vaizde yra prarastos, todėl agento strategija gali tapti „akla“ toliau esantiems objektams

3.2. Treniravimas ir rezultatai

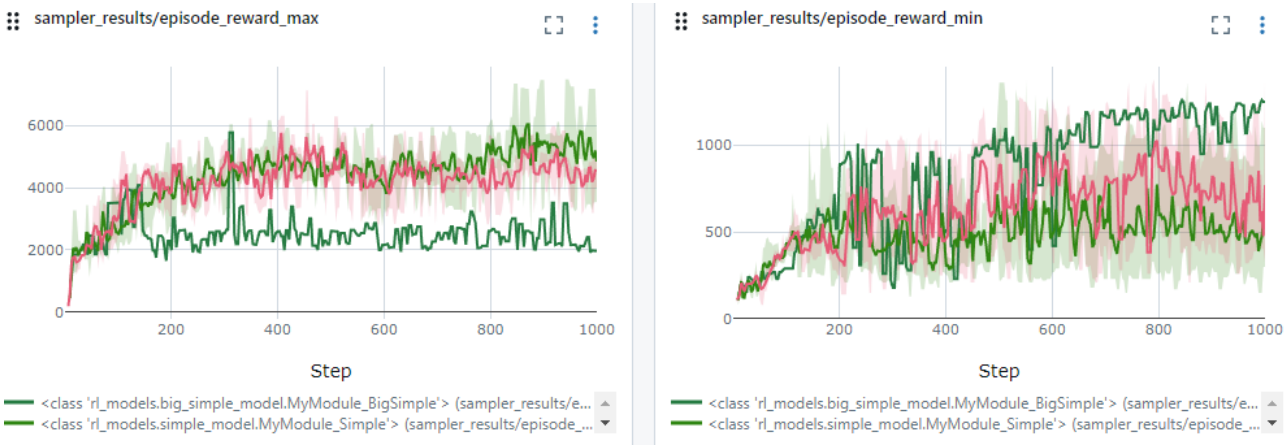
Dėl skirtingų taškų kiekių kiekviename žaidime, žemiau pateikiamos atskiros duomenų suvestines kiekvienam žaidimui. Paskirstytos strategijos atšakų skirtingumas įvertinamas pagal formulę: $\delta = E_{samples}\left(\sum_{k=1}^{actiondim}(A_k^{mean} - A_k^2)\right)^{0.5}$, čia A žymi veiksmo tikimybę.

3.2.1. „Ms Pacman“

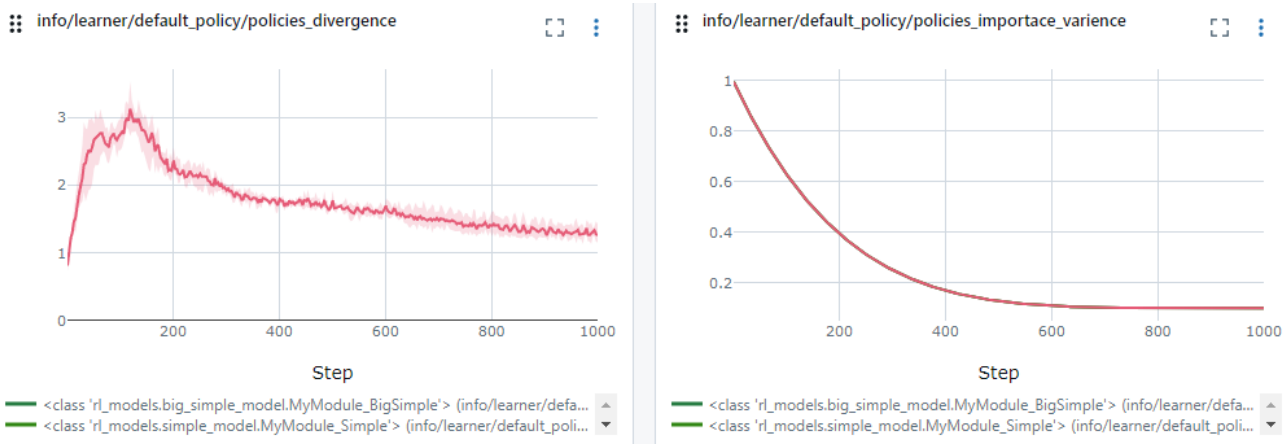
Ši aplinka išsiskiria tuo, kad agentas vienu metu mato visą žaidimo būseną, todėl eksperimentai su LSTM ir GRU tinklais nebuvo atliekami. Kaip galima matyti iš **3.11 pav.** ir **3.12 pav.** nemodifikuoto ir paskirstytos strategijos algoritmų vidutiniai įverčiai yra iš esmės identiški, kas reikštų kad modifikacija nepadarė didesnio skirtumo, tuo tarpu padidintas neuroninis tinklas pasiekė šiek tiek prastesnius rezultatus. Paskirstytos strategijos variacija naudojo 5 strategijos atšakas ir svorių dispersijos mažėjimo hiperparametras buvo lygus 5. Kaip matoma iš **3.13 pav.** atšakos išlaikė skirtingas reikšmes nepaisant to, jog šiame žaidime naudojami diskretūs veiksmai ir tarpinės, netikslios reikšmės neduotų optimalių rezultatų. Šios aplinkos rezultatuose taip pat galima pastebėti, kad didesnis treniruojamų parametrų skaičiaus nepadėjo pasiekti geresnių rezultatų, o kai kuriais atvejais jie pablogėjo. Tai galėjo nutikti dėl to, kad didesnis parametrų kiekis reikalauja daugiau duomenų, kad būtų optimaliai ištreniruotas.



3.11 pav. Vidutinis epizodo įvertis „Ms Pacman“ žaidime treniravimo metu, kai naudojamos skirtingos modifikacijos. Tamsi žalia linija – paprastas padidintas tinklas, šviesiai žalia – paprastas tinklas, raudona – paskirstytos strategijos funkcijos tinklas



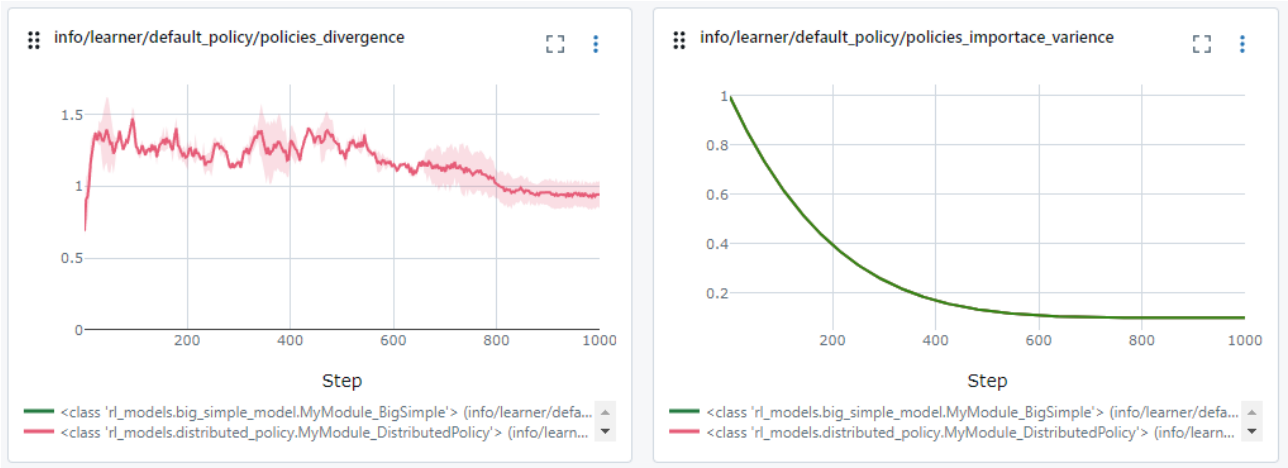
3.12 pav. Maksimalūs ir minimalūs epizodo įverčiai „Ms Pacman“ žaidime treniravimo metu, kai naudojamos skirtingos modifikacijos. Tamsi žalia linija – paprastas padidintas tinklas, šviesiai žalia – paprastas tinklas, raudona – paskirstytos strategijos funkcijos tinklas



3.13 pav. Atskirų strategijos atšakų skirtumai ir atsitiktinių strategijos svorių dispersija „Ms Pacman“ žaidime treniravimo metu, kai naudojamos skirtingos modifikacijos. Tamsi žalia linija – paprastas padidintas tinklas, šviesiai žalia – paprastas tinklas, raudona – paskirstytos strategijos funkcijos tinklas

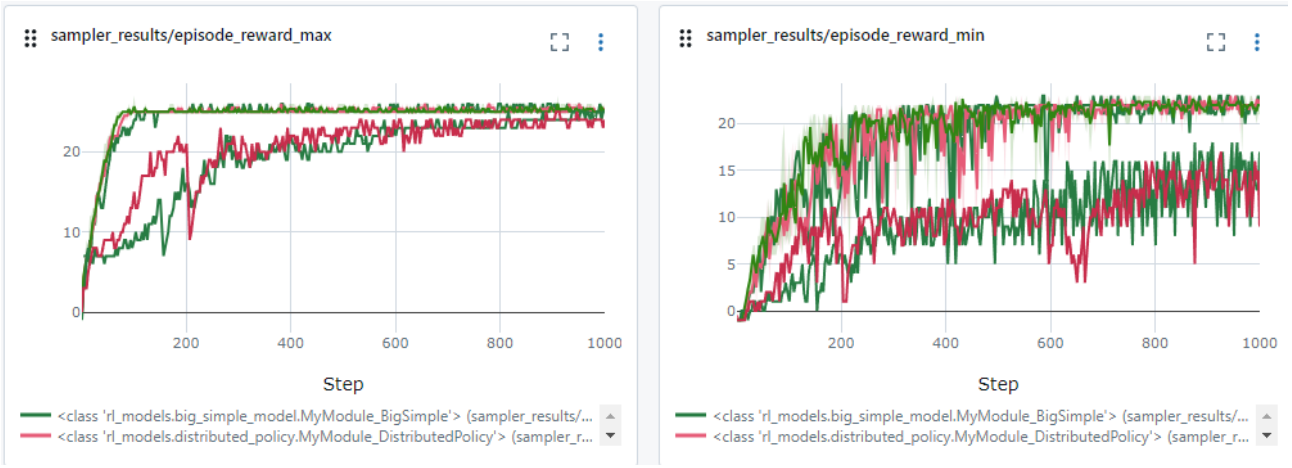
3.2.2. „VizDoom“„Deffend the center“

Šioje aplinkoje agentas taip pat turi naudoti tolydžius kintamuosius, tokius kaip kompiuterio pelės postūmis. Taip pat ši aplinka yra 3D tipo kai agentas nemato visos scenos vienu metu ir turi gebėti atsiminti arba dairytis naujos informacijos. Paskirstytos strategijos variacija naudojo 5 strategijos atšakas ir svorių dispersijos mažėjimo hiperparametras buvo lygus 5. Kaip matyti iš pateiktų **3.15 pav.** ir **3.16 pav.** LSTM ir GRU nepasirodė tinkami šiai užduočiai spręsti, nors jų tobulėjimas stabilius, kitos variacijos sugebėjo pasiekti geresnių rezultatų per trumpesnį laiką (mažiau simuliacijos duomenų). Kaip ir ankstesniame žaidime, galima pastebėti kas tarp paskirstytos ir klasikinės tinklo variacijos nėra didelio skirtumo ir abudu algoritmai atsimuša į dirbtinę žaidimo taškų ribą. **3.14 pav.** matomas mažesnis strategijų skirtumas palyginus su ankstesne aplinka.



3.14 pav. Atskirų strategijos atšakų skirtumai ir atsitiktinių strategijos svorių dispersija „VizDoom deffend the center“ žaidime treniravimo metu, kai naudojamos skirtingos modifikacijos. Tamsi žalia linija – paprastas padidintas tinklas, šviesiai žalia – paprastas tinklas, raudona – paskirstytos strategijos funkcijos tinklas. Žemiau esančios žalia linija – LSTM, raudona GRU

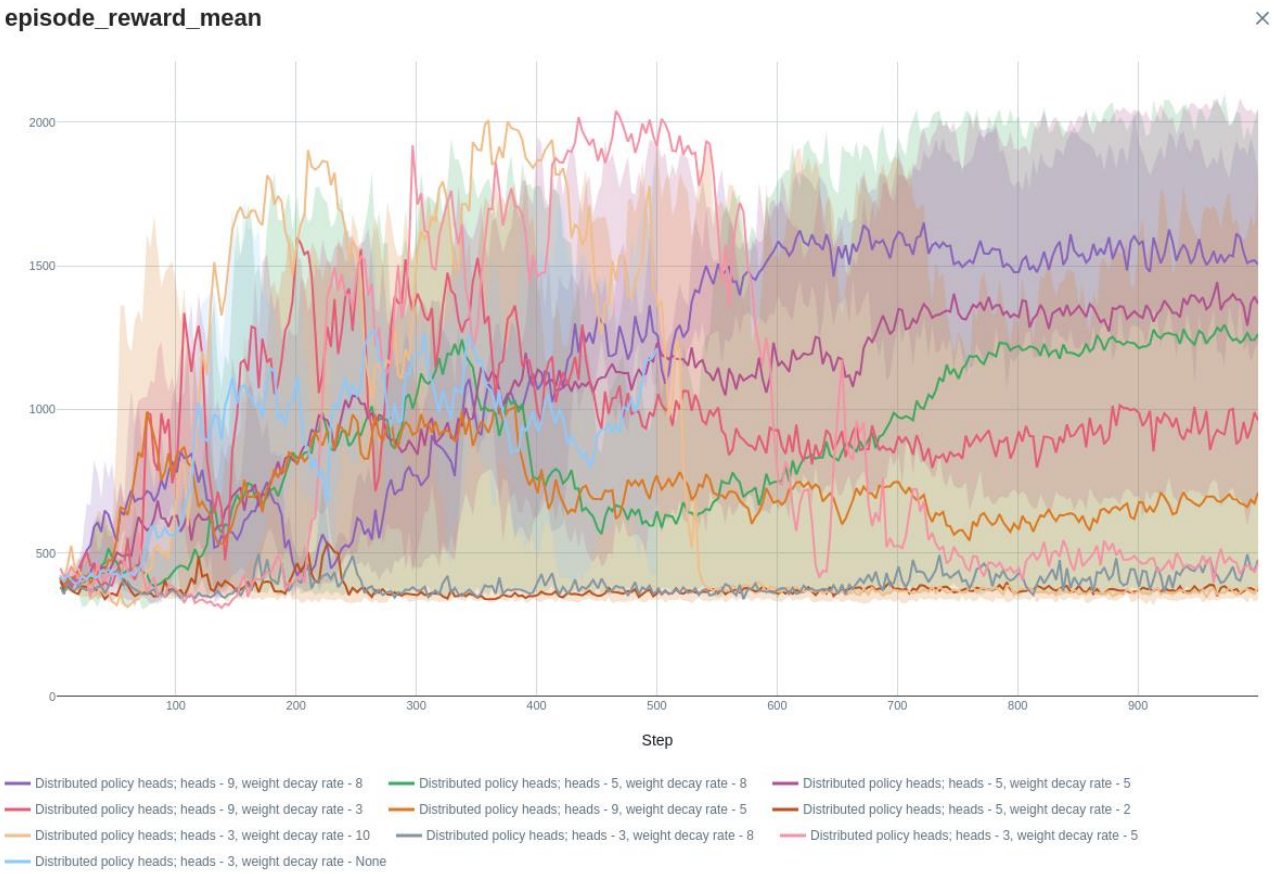
3.15 pav. Vidutinis epizodo įvertis „VizDoom deffend the center“ žaidime treniravimo metu, kai naudojamos skirtingos modifikacijos. Tamsi žalia linija – paprastas padidintas tinklas, šviesiai žalia – paprastas tinklas, raudona – paskirstytos strategijos funkcijos tinklas. Žemiau esančios žalia linija – LSTM, raudona GRU



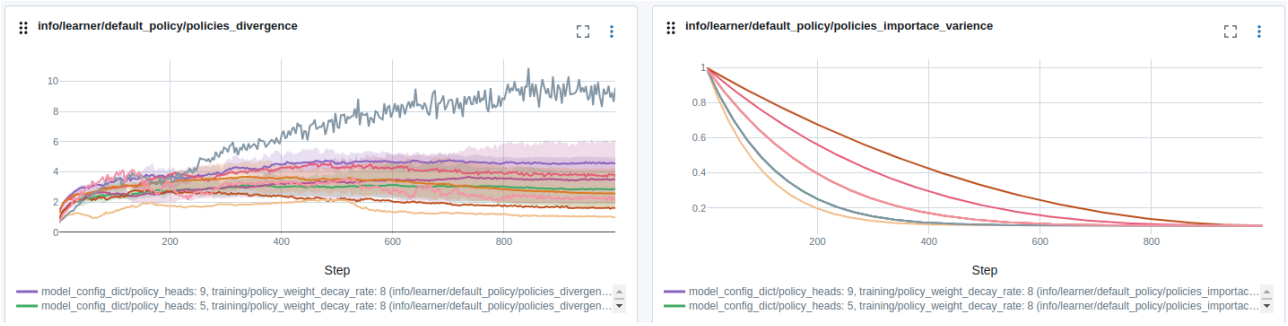
3.16 pav. Maksimalūs ir minimalūs epizodo įverčiai „VizDoom deffend the center“ žaidime treniravimo metu, kai naudojamos skirtingos modifikacijos. Tamsi žalia linija – paprastas padidintas tinklas, šviesiai žalia – paprastas tinklas, raudona – paskirstytos strategijos funkcijos tinklas. Žemiau esančios žalia linija – LSTM, raudona GRU

3.2.3. „VizDoom“„Health gathering“

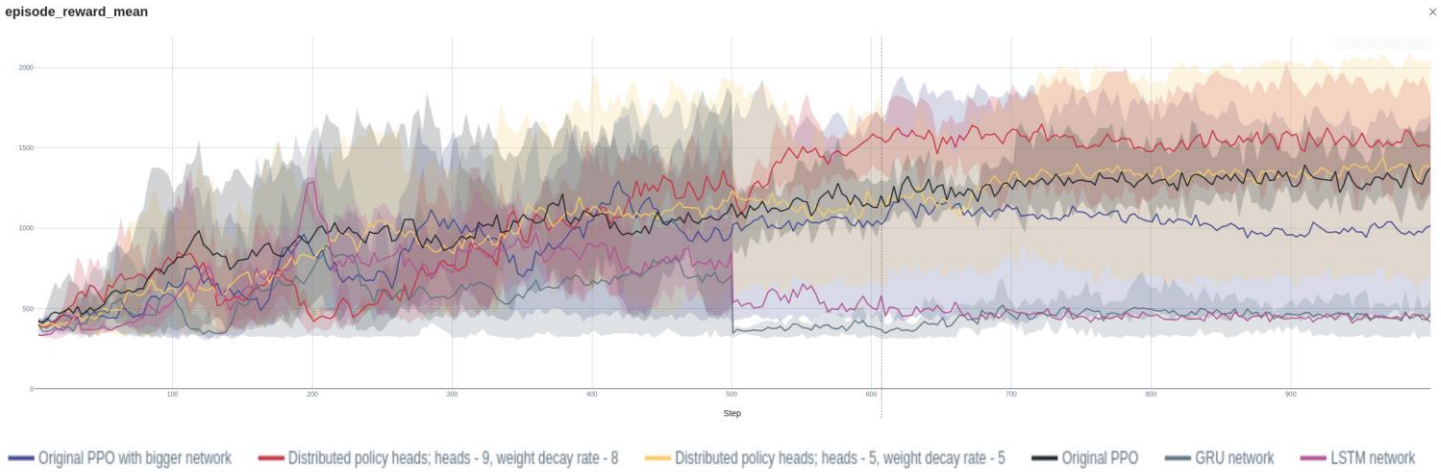
Pasiūlytas paskirstytos strategijos funkcijos PPO algoritmas nuo standartinio varianto skiriasi tuo kad jam reikia parinkti 2 papildomus hiperparametrus. Siekiant išsiaiškinti kokios reikšmės gali duoti teigiamų rezultatų, buvo atlikti eksperimentai su skirtingomis šių reikšmių kombinacijomis. Kaip galima matyti **3.17 pav.** ne visos kombinacijos veikia ir daugelis pasiekia gana prastus rezultatus ir atlieka tik atsitiktinius veiksmus. Iš tirtų strategijos atšakų kiekio ir svorių skirtumų mažėjimo greičio išsiskyrė 3 kombinacijos: (9, 8), (5, 5), (5, 8). Geriausias vidurkis pasiektas su kombinacija (9, 8) tačiau galima pastebėti kad mokymasis turėjo netolygumų, todėl kitiems palyginimo eksperimentams buvo pasirinkta naudoti kombinaciją (5, 5). **3.18 pav.** Galima matyti kaip kinta strategijų pranašumas su skirtingais parametrais, bei kaip atrodo skirtingų laipsnių svorių skirstinio mažėjimas treniravimo metu. **3.19 pav.** Pateikiami bendri rezultatai su kiekvieno algoritmo atliktų eksperimentų vidurkiu. Jame galima matyti didelį ,paskirstytos strategijos PPO, reikšmių intervalą, tačiau galima pastebėti, kad pati aukščiausi riba yra pasiekta su viena iš šios modifikacijos variacijų. Kaip galima matyti **3.20 pav.** dalis hiperparametrų kombinacijų turėjo itin didelę rezultatų įvairovę kai su identiškais parametrais skirtingi eksperimentai pasiekia ir pačius blogiausius ir geriausius rezultatus iš visų bandymų.



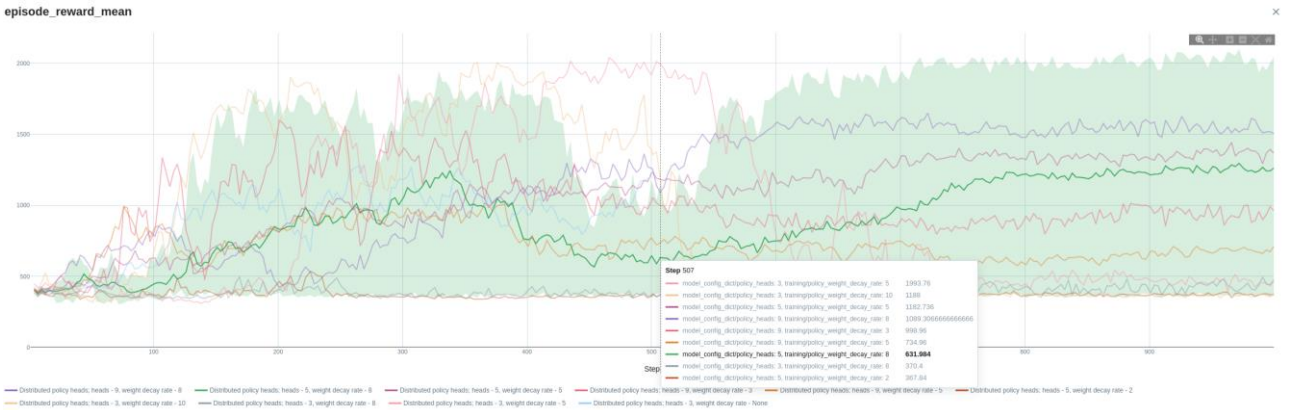
3.17 pav. Vidutinis taškų skaičius per epizodą treniravimo metu, skirtinguose paskirstytos strategijos variantuose. Treniravimas vyksta „VizDoom Health gathering“ aplinkoje, variacijos skiriasi hiperparametrų: strategijų atšakų skaičius ir strategijos svorių dispersijos mažėjimo sparta



3.18 pav. Kairėje: atskirų strategijos atšakų skirtumai treniravimo metu su skirtingais hiperparametrais. Dešinėje : strategijų svorių dispersijos kitimas su skirtingais hiperparametrais



3.19 pav. Bendri skirtingų algoritmų rezultatai (t.y. vidutiniškai surinkti taškai per epizodą) iš „VizDoom Health gathering“ žaidimo. Dvi apatinės linijos: pilka – GRU, violetinė – LSTM, mėlyna – padidintas tinklas, juoda – paprastas PPO tinklas, geltona – paskirstytos strategijos funkcijos PPO su 5 atšakomis ir svorių mažėjimu 5, ir raudona linija yra paskirstytos strategijos funkcijos PPO su hiperparametrais 9, ir 8



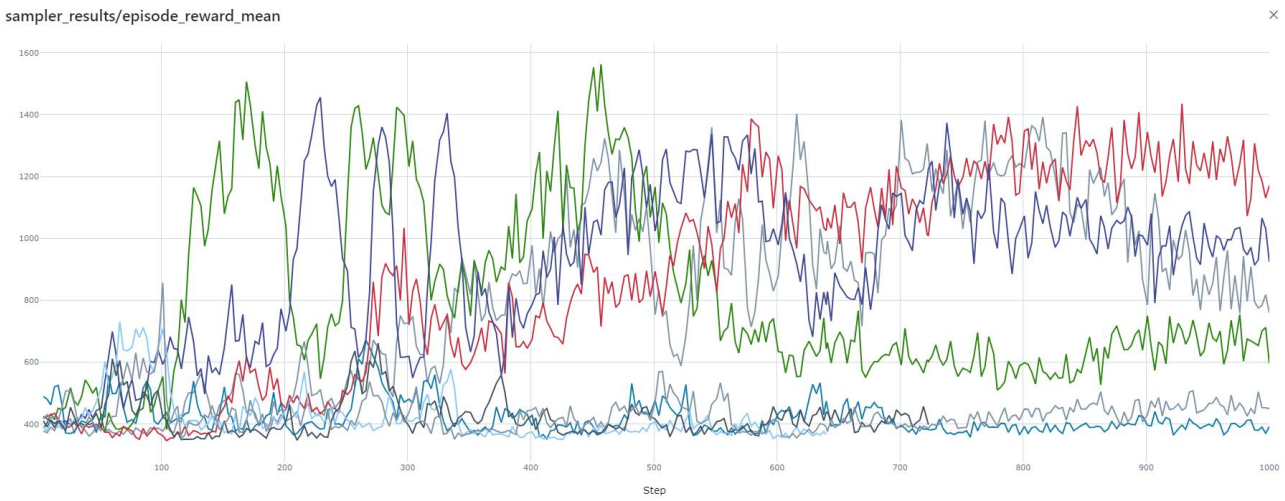
3.20 pav. Paskirstytos strategijos funkcijos rezultatų intervalas su hiperparametrais (5, 8). Eksperimentai su šia variacija pasiekė tiek blogiausius, tiek geriausius rezultatus

3.2.4. VAE grįsto tinklo rezultatai

Bandymai su variacinio autoenkoderio variacija buvo atlikti tik „VizDoom“„Health gathering“ aplinkoje norint išbandyti daugiau algoritmo modifikacijų. **3.2 lentelė** pateikiamos sėkmingiausios variacijos ir jų rezultatai, taip pat **3.21 pav.** pateikiamas skirtingų eksperimentų rezultatų kitimas treniravimo metu. Galima pastebėti, kad geriausi rezultatai buvo pasiekti kai VAE ir PPO dalys atskirai optimizavo jiems priskirtas tinklo dalis, tai atliekant beveik viso treniravimo metu (500 VAE treniravimo iteracijų). Tai žymima raudona linija grafike, kuri taip pat yra viena iš stabiliausiai kreivių. Variacijos su didesniu parametrų kiekiu parodė prastesnius rezultatus, tikėtina dėl to, nes joms reikia didesnio kiekio duomenų ir treniravimo iteracijų, kad tinklas išmoktų imituoti reikiamą funkciją. Dalyje eksperimentų galima pastebėti staigius rezultatų šuolius treniravimo metu, kai maksimalius surinktų taškų reikšmės pasiekiamos po ~200 iteracijų. Panašius rezultatus galima rasti ir standartinio PPO sėkmingesniuose bandymuose, tačiau būtų naudinga ištirti ar panašūs šuoliai atsirastų ir sudėtingesnėse aplinkose. Jei rezultatai tenkina reikiamai užduočiai atlikti, treniravimą būtų galima stabdyti atlikus vos 20% treniravimo – taip taupant resursus.

3.2 lentelė. Skirtingų VAE integracijos variacijų rezultatai

Spalva grafike 3.21 pav.	Vidutiniai taškai per epizodą, po treniravimo	Aprašymas
Šviesiai žydra (640 iteracijų)	366	Tinklas su SELU aktyvacija, VAE treniruojamas pirmą 100 iteracijų kartu su PPO
Violetinė	924	Tinklas su SELU aktyvacija, pradžioje pirmą 100 iteracijų treniruojamas tik VAE, vėliau treniravimas stabdomas ir treniruojamas tik PPO.
Raudona	1171	SELU aktyvacija, VAE treniruojama pirmus 500 iteracijų, PPO treniruojamas taip pat, tačiau gradientai enkoderio dalyje išjungti kol mokomas VAE.
Žalia	597	SELU aktyvacija, padidintos strategijos ir vertės funkcijų dalys pridedant papildomą sluoksnį, VAE treniruojama pirmus 200 iteracijų, PPO treniruojamas taip pat, tačiau gradientai enkoderio dalyje išjungti kol mokomas VAE.
Mėlyna	392	SELU aktyvacija, padidintos strategijos ir vertės funkcijų dalys pridedant papildomą sluoksnį, VAE treniruojama pirmus 200 iteracijų, PPO treniruojamas visą laiką.
Pilka	760	SELU aktyvacija, padidintos strategijos ir vertės funkcijų dalys pridedant papildomą sluoksnį, VAE treniruojama pirmus 500 iteracijų, PPO treniruojamas taip pat, tačiau gradientai enkoderio dalyje išjungti kol mokomas VAE.



3.21 pav. Vidutinis per epizodą surinktų taškų kiekis treniravimo metu. Skirtingos linijos žymi eksperimentus su skirtingomis treniravimo strategijomis, neuroniniais tinklais ir hiperparametrais

Išvados

Šiame darbe buvo atliekami eksperimentai su ‚neuroniniais tinklais grįstais, skatinamojo mokymosi algoritmais ir kompiuterinių žaidimų aplinkomis. Žaidimai suteikia galimybę išbandyti algoritmus aplinkose su skirtingomis sąlygomis, taisyklėmis ir agentui galimais veiksmais, taip pat žaidime esančios taškų sistemos suteikia puikų naudingumo įvertį. Skatinamojo mokymosi algoritmo tikslas yra išmokti surasti optimaliausią veiksmų seką ir surinkti kuo didesnį taškų kiekį bet kokioje situacijoje. Šiuo atveju informacija apie dabartinę agento būseną algoritmui perduodama kaip ekrano vaizdų rinkinys, todėl jis yra universalus visiems atvejams, kai veiksmai priklausi nuo regos. Viena pagrindinių šios srities problemų yra didelio treniravimo duomenų kiekio poreikis, todėl atliktos modifikacijos siekė sumažinti reikiamų duomenų ir simuliacijų poreikį efektyviau panaudojant jau surinktą informaciją.

Bandymai buvo atlikti 3 aplinkose su skirtingomis savybėmis: „Ms Pacman“ 2D tipo žaidimas su pilnu matomumu, „VizDoom Deffend the center“ 3D žaidimas su tolygiais veiksmais, „VizDoom Health gathering“ 3D žaidimas su diskrečiais veiksmais. Baziniu lygiu buvo pasirinkta laikyti PPO [10] su standartine neuroninio tinklo architektūra, kuri susideda iš 3 CNN sluoksnių ir 3 pilnai jungiųjų sluoksnių, vienintelis atliktas pakeitimas buvo sluoksnio normalizavimo sluoksniai, nes tai žymiai stabilizavo treniravimą ir pagerino rezultatus. Galutiniai apmokymai buvo atliekami su 10M žaidimo kadrų, ir kiekviena konfigūracija išbandoma bet po 3 kartus, kad būtų atsižvelgta į variacijas tarp atsitiktinai geresnių ar prastesnių rezultatų.

Buvo pristatytos ir ištirtos 2 neuroninio tinklo architektūros ir treniravimo eigos modifikacijos:

1. **Paskirstyta strategijos funkcija** – naujai pasiūlyta tinklo architektūra kurioje, strategijos funkcija modeliuojama iš keleto atšakų vidurkio. Tai remiasi idėjomis iš ansamblio mašininio mokymosi metodų, kai naudojama keletas paprasto klasifikatoriaus kopijų, kad būtų sumažintas šališkumas (ang. bias). Taip pat treniravimo pradžioje skirtingos strategijos priverčia agentą ištyrinėti skirtingesnius veiksmus, ko pasėkoje padidėja treniravimo duomenų įvairovė ir kokybė. Dėl sudėtingesnės tinklo struktūros atsirado 2 papildomi hiperparametrai, kuriuos riekia optimizuoti – tai yra strategijos funkcijos atšakų kiekis ir atšakų svorių dispersijos mažėjimo greitis. Atlikus bandymus optimaliausios atšakų ir greičio parametrų kombinacijos atitinkamai buvo (9, 8) ir (5, 5). Žaidime „VizDoom Deffend the center“ ir originalus ir modifikuotas tinklas pasiekė vienodus galutinius rezultatus, o rezultatai treniravimo metu kito identiškai. Tai galėjo nutikti dėl per paprastos aplinkos kuri nesugebėjo pasiekti algoritmo limitų. Didžiausi skirtumai pastebėti „VizDoom Health gathering“ žaidime, kur geriausia variacija per epizodą vidutiniškai surinko po 1518 taškus, o nemodifikuotas algoritmas surinko 1377 taškus, kas rodo ~10% pranašumą. Visos algoritmo variacijos turėjo mokymosi nestabilumų ir konverguodavo į geresnius arba blogesnius rezultatus, tačiau modifikuoto algoritmo rezultatų dispersija pasirodė daug didesnė. Modifikuotas algoritmas geriausiu bandymu surinko 2046 taškus, o nemodifikuotas 1666, kas atitinka ~23% geresnius rezultatus. Tai gali būti naudinga, kai reikalingas kuo geresnis algoritmas ir yra galimybė atlikti keletą treniravimo eksperimentų. Galiausiai, žaidime „Ms Pacman“ abi variacijos pasiekė vienodą taškų kiekio per epizodą vidurkį, tačiau, matuojant mažiausiai per epizodą surinktų taškų kiekį, modifikuota versija vidutiniškai surinko ~18% daugiau taškų.

2. **Variaciniu autoenkoderio grįstas vaizdo suspaudimas** – tai tinklo architektūra, kurioje panaudojama papildoma atšaka, kuri suformuoja pilną VAE modulį. Šią dalį vėliau galima treniruoti su visais turimais duomenimis, nepriklausomai nuo to, su kokia strategijos funkcija jie buvo sugeneruoti. Pagrindinė idėja yra sumažinti reikiamų duomenų kiekį, įgalinant juos treniravimui panaudoti daug kartų, kaip tai daroma Q–funkcija naudojančiuose algoritmuose. Tam buvo panaudotas masyvas talpinantis 100k naujausių iš žaidimo surinktų kadrų, vėliau iš jo atsitiktinai pasirenkama 40k kadrų rinkinys ir su juo atliekamas treniravimas variacinio autoenkoderio atšakoje. Atliekant bandymus pastebėta, kad skirtingos tikslo funkcijos iš VAE vaizdo rekonstrukcijos ir PPO strategijos sukuria skirtingus gradientus ir trukdo viena kitai mokytis. Vienas iš paaiškinimų gali būti tai, jog naudotas tinklas yra per mažas išmokti bendrines vaizdo suspaudimo abstrakcijas ir iškart optimizuojasi atpažinti tik sprendimui svarbias detales. Atlikus eksperimentus su skirtingomis variacijomis geriausius rezultatus parodė variacija, kuri tinklą optimizuoja kaip 2 atskiras dalis – vieną vaizdo suspaudimui su VAE, o strategiją su PPO. Ši variacija naudojama pusę treniravimo laiko, 500 iteracijų, o vėliau visas optimizavimas perduodamas PPO, kad jis galėtų geriau pakoreguoti pagrindinę algoritmo funkciją. Pastebėtas ir vienas svarbus šio metodo trūkumas, jog negalima per anksti ir per greitai atlikti VAE treniravimo, kol agentas dar nėra garai ištyrinėjęs aplinkos, nes tai lemia mažą duomenų įvairovę ir VAE negebėjimą atkurti ir reprezentuoti naujos aplinkos ir objektų, dėl ko tolimesnė agento strategija tampa „akla“. Dauguma šios variacijos apmokymų pasirodė esą labai nestabilūs, su staigiais surinktų taškų pokyčiais treniravimo metu. Geriausia variacija per epizodą vidutiniškai surinko ~1200 taškų aplinkoje „VizDoom Health gathering“.
3. Buvo išbandyta ir keletas variacijų su LSTM ir GRU rekurentiniais sluoksniais, tačiau jie pasirodė prasčiau už kitus algoritmus, nors ir dvi iš trijų aplinkų buvo 3D tipo ir rekurentinių tinklų atminties savybės jiems turėjo praversti

Ateityje būtų naudinga išbandyti kaip šios algoritmų modifikacijos veikia vizualiai ir strategiškai sudėtingesnėse aplinkose. Įdomu, ar paskirstytos strategijos funkcijos modifikacija vis dar būtų naudinga žaidimuose, kur reikalingas ilgas planavimas į priekį. Taip pat, VAE galima pabandyti pakeisti kvantuoto vektoriaus variacija VQ–VAE [25]ir išbandyti ar diskrečios reikšmės padeda optimaliau suspausti vaizdinius ir atskirti scenoje vaizduojamus objektus.

Literatūros sąrašas

[1] N. Justesen, P. Bontrager, J. Togelius ir S. Risi, „Deep Learning for Video Game Playing," August 2017.

[2] M. T. Regehr ir A. Ayoub, „An Elementary Proof that Q-learning Converges Almost Surely," August 2021.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg ir D. Hassabis, „Human-level control through deep reinforcement learning," *Nature,* t. 518, p. 529–533, February 2015.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra ir M. Riedmiller, „Playing Atari with Deep Reinforcement Learning," December 2013.

[5] K. Arulkumaran, M. P. Deisenroth, M. Brundage ir A. A. Bharath, „A Brief Survey of Deep Reinforcement Learning," *IEEE Signal Processing Magazine,* t. 34, p. 26–38, 19 November 2017.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver ir K. Kavukcuoglu, „Asynchronous Methods for Deep Reinforcement Learning," *ICML 2016,* February 2016.

[7] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver ir K. Kavukcuoglu, „Reinforcement Learning with Unsupervised Auxiliary Tasks," November 2016.

[8] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu ir N. de Freitas, „Sample Efficient Actor-Critic with Experience Replay," November 2016.

[9] J. Schulman, P. Moritz, S. Levine, M. Jordan ir P. Abbeel, „High-Dimensional Continuous Control Using Generalized Advantage Estimation," June 2015.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford ir O. Klimov, „Proximal Policy Optimization Algorithms," July 2017.

[11] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar ir D. Silver, „Rainbow: Combining Improvements in Deep Reinforcement Learning," October 2017.

[12] A. Graves, „Generating Sequences With Recurrent Neural Networks," August 2013.

[13] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink ir J. Schmidhuber, „LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems (Volume: 28, Issue: 10, Oct. 2017) Pages: 2222 - 2232,* t. 28, p. 2222–2232, 13 October 2015.

[14] J. L. Ba, J. R. Kiros ir G. E. Hinton, „Layer Normalization," July 2016.

[15] S. Ioffe ir C. Szegedy, „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," February 2015.

[16] T. Salimans ir D. P. Kingma, „Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," February 2016.

[17] J. Zhu, F. Wu ir J. Zhao, „An Overview of the Action Space for Deep Reinforcement Learning,“ December 2021.

[18] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan ir I. Stoica, „Ray: A Distributed Framework for Emerging AI Applications,“ December 2017.

[19] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan ir I. Stoica, „Ray RLlib: A Framework for Distributed Reinforcement Learning,“ December 2017.

[20] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg ir K. Kavukcuoglu, „IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures,“ February 2018.

[21] M. Kempka, M. Wydmuch, G. Runc, J. Toczek ir W. Jaśkowski, „ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning,“ įtraukta *IEEE Conference on Computational Intelligence and Games*, Santorini, 2016.

[22] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, K. G. Arjun, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen ir O. G. Younis, *Gymnasium,* Zenodo, 2023.

[23] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins ir A. Lerchner, „Understanding disentangling in β-VAE,“ April 2018.

[24] D. P. Kingma ir M. Welling, „Auto-Encoding Variational Bayes,“ December 2013.

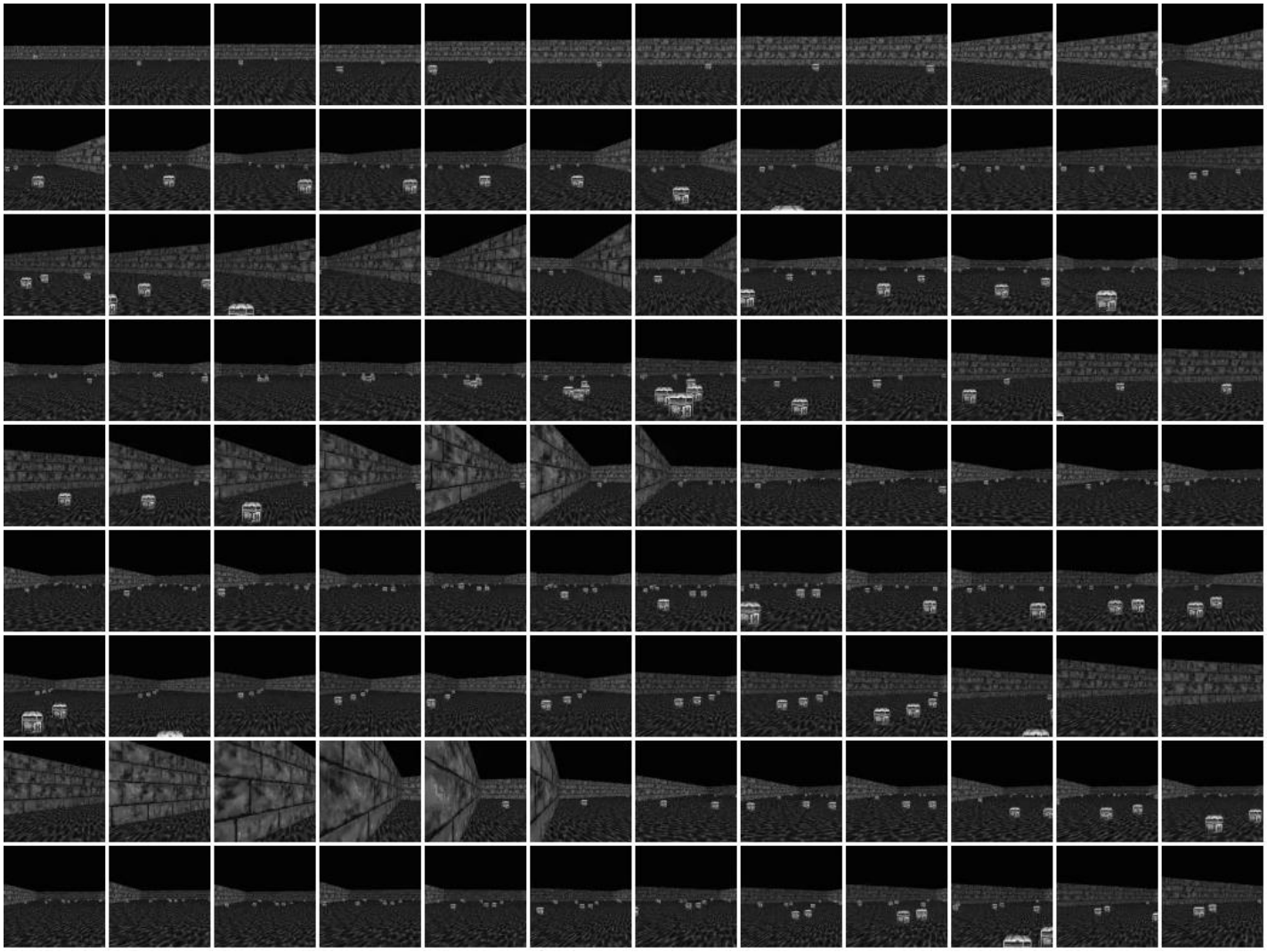
[25] A. v. d. Oord, O. Vinyals ir K. Kavukcuoglu, „Neural Discrete Representation Learning,“ November 2017.

Informacijos šaltinių sąrašas

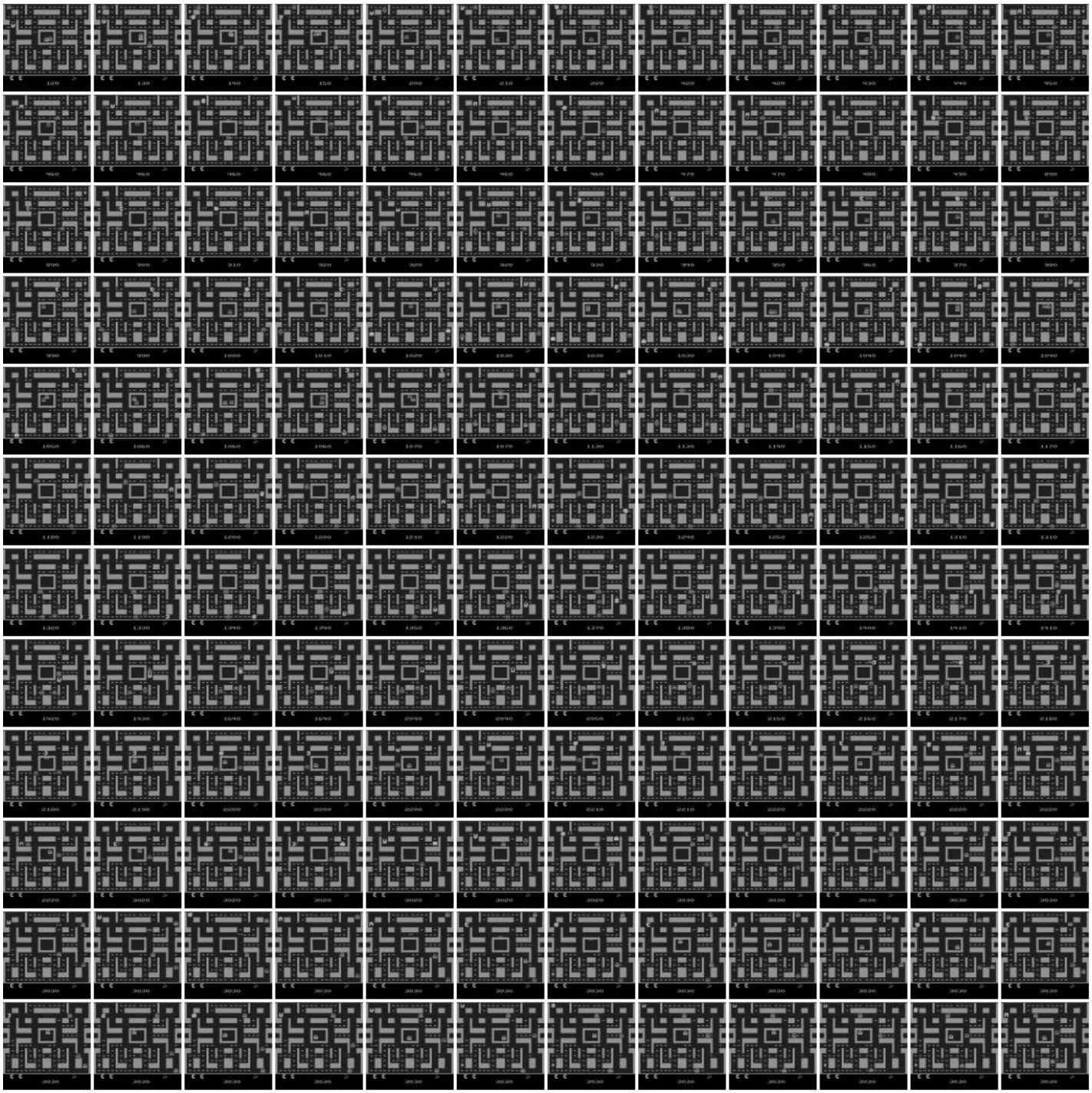
1. https://mcneela.github.io/math/2018/04/18/A-Tutorial-on-the-REINFORCE-Algorithm.html
2. https://lilianweng.github.io/posts/2018-08-12-vae/
3. https://blog.evjang.com/2016/08/variational-bayes.html
4. https://spinningup.openai.com/en/latest/user/algorithms.html
5. https://docs.ray.io/en/latest/rllib/index.html
6. https://www.youtube.com/@AndrejKarpathy/videos

Priedai

1 priedas. Agento veikimo žaidime „VizDoom Health gathering“ pavyzdys



2 priedas. Agento veikimo žaidime „Ms Pacman“ pavyzdys



3 priedas. Ekrano vaizdo ir VAE rekonstruoto vaizdo palyginimas