**Kauno technologijos universitetas**

Matematikos ir gamtos mokslų fakultetas

# Verslo procesų žaidybinimas ir imitacinis modeliavimas pritaikant skatinamąjį mokymąsi ir sintetinius duomenis

Baigiamasis magistro studijų projektas

**Justas Juozelskis**

Projekto autorius

**Prof. Dr. Robertas Alzbutas**

Vadovas

**Doc. Dr. Raminta Vaitiekūnienė**

Vadovė

**Kaunas, 2025**

**Kauno technologijos universitetas**

Matematikos ir gamtos mokslų fakultetas

# Verslo procesų žaidybinimas ir imitacinis modeliavimas pritaikant skatinamąjį mokymąsi ir sintetinius duomenis

Baigiamasis magistro studijų projektas

Didžiųjų verslo duomenų analitika (6213AX001)

**Justas Juozelskis**

Projekto autorius

**Prof. Dr. Robertas Alzbutas**
Vadovas

**Doc. Dr. Raminta Vaitiekūnienė**
Vadovė

**Doc. Dr. Kristina Šutienė**
Recenzentė

**Doc. Dr. Neringa Gerulaitienė**
Recenzentė

**Kaunas, 2025**


**Kauno technologijos universitetas**

Matematikos ir gamtos mokslų fakultetas

Justas Juozelskis

# Verslo procesų žaidybinimas ir imitacinis modeliavimas pritaikant skatinamąjį mokymąsi ir sintetinius duomenis

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 (-usi);

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.

Justas Juozelskis

*Patvirtinta elektroniniu būdu*

Juozelskis Justas. Verslo procesų žaidybinimas ir imitacinis modeliavimas pritaikant skatinamąjį mokymąsi ir sintetinius duomenis. Magistro studijų baigiamasis projektas / vadovas prof. dr. Robertas Alzbutas, vadovė doc. dr. Raminta Vaitiekūnienė; Kauno technologijos universitetas, Matematikos ir gamtos mokslų fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Taikomoji matematika (Matematikos mokslai).

Reikšminiai žodžiai: skatinamasis mokymasis, verslo procesų imitacinis modeliavimas, žaidybinimas, sintetiniai duomenys.

Kaunas, 2025. 63 p.

## Santrauka

Baigiamajame projekte analizuojamas skatinamojo mokymosi metodų taikymas imituojant verslo procesus. Tyrimo aktualumas – skatinamojo mokymosi taikymas imitaciniam verslo procesų modeliavimui sudaro sąlygas modeliuoti sudėtingus ir neapibrėžtus verslo procesus, naudojant sintetinius duomenis, kurie atvaizduoja eksperimentinių sąlygų rezultatus ir leidžia tirti procesus be realios rizikos, o žaidybinimas padeda įtraukti dinamiškos aplinkos elementus, būdingus realioms verslo situacijoms.

Tyrime aprašytos dvi imitacinės verslo procesų aplinkos. Pirmojoje aplinkoje yra įtraukiamos aukštos vertės klientų valdymo situacijos, o antrojoje pridedamas rizikingų užsakymo aptarnavimas esant rinkos kliūtims. Abiejose aplinkose skatinamojo mokymosi agentas mokomas naudojant Markovo sprendimų proceso struktūrą, kur būsenos aprašytos daugiamačiais vektoriais, įtraukiančiais klientų, kliūčių, užsakymų ir įmonės būsenos aplinkoje indikatorių duomenis. Veiksmų erdvė apibrėžta keturiomis judėjimo kryptimis, o atlygio funkcija sudaryta iš penkių dedamųjų – bazinio atlygio už klientų aptarnavimą, atlygio už efektyvų judėjimą, baudos už neefektyvų judėjimą, kliūčių neišvengimą ir rizikingų užsakymų vykdymą. Žaidybinimas naudojamas kaip metodologinis pagrindas kuriant imitacinės aplinkos logiką pagal klasikinio „Gyvatėlės“ žaidimo principus, papildytus verslo logika. Skatinamojo mokymosi agentų mokymui pritaikyti DQN ir PPO metodai, realizuoti naudojant *Stable-Baselines3* biblioteką. Agentų mokymas vyko trimis etapais – mokymo, tyrinėjimo ir taikymo, vertinant įprastų ir aukštos vertės klientų aptarnavimą, bendrą atlygį ir klaidų dažnį.

Aukštos vertės klientų aptarnavimo aplinkoje PPO metodas parodė geresnius rezultatus, vidutiniškai aptarnaujant 14 įprastus ir 4,1 aukštos vertės klientus, o maksimalus epizodinis atlygis siekė 2011,2 reikšmę. Tuo tarpu DQN vidutiniškai aptarnavo tik 10,8 klientų, o atlygis siekė 1076,4 reikšmę. PPO metodas ne tik sumažino aukštos vertės klientų neaptarnavimo atvejus, bet ir optimizavo judėjimo logiką, parodydamas aukštesnę 75 % kvartilio reikšmę už DQN metodą daugiau nei 2,7 karto. Rizikingų užsakymų aptarnavimo aplinkoje PPO rezultatai išliko reikšmingai aukštesni. Vidutinis PPO atlygis taikymo etape siekė 708,9, kai DQN pasiekė 288,9 reikšmę. PPO beveik visiškai išvengė susidūrimų su kliūtimis taikymo etape, kai DQN patirdavo bent vieną susidūrimą su kliūtimi kas septintą epizodą. PPO metodas ne tik generuoja aukštesnį atlygį, bet ir pasižymi gebėjimu išlaikyti elgsenos nuoseklumą, įtraukti aukštos vertės klientus ir rizikingų sprendimų dinamiką į taikomą strategiją. Tuo tarpu DQN rodo per dažną lokalų optimizavimą ir negebėjimą efektyviai prisitaikyti prie verslo logikos.

Juozelskis Justas. Gamification and Simulation of Business Processes Using Reinforcement Learning and Synthetic Data. Master's Final Degree Project / supervisor prof. dr. Robertas Alzbutas, supervisor doc. dr. Raminta Vaitiekūnienė; Faculty of Mathematics and Natural Sciences, Kaunas University of Technology.

Study field and area (study field group): Applied Mathematics (Mathematical Sciences).

Keywords: reinforcement learning, business process simulation, gamification, synthetic data.

Kaunas, 2025. 63 p.

**Summary**

The final project analyzes the application of reinforcement learning methods in simulating business processes. The relevance of the study lies in the fact that applying reinforcement learning to business process simulation enables the modeling of complex and uncertain business scenarios using synthetic data. Synthetic data reflects the outcomes of experimental conditions and allows process analysis without real-world risks, while gamification helps incorporate elements of a dynamic environment typical of real business situations.

The study describes two simulated business process environments. The first environment involves scenarios related to high-value client management, while the second introduces the service of risky orders under market constraints. In both environments, the reinforcement learning agent is trained using a Markov Decision Process structure, where states are represented by multidimensional vectors incorporating data on clients, obstacles, orders, and the company's status in the environment. The action space is defined by four movement directions, and the reward function consists of five components: base reward for servicing clients, reward for efficient movement, penalties for inefficient movement, collision with obstacles, and execution of risky orders.

Gamification is used as a methodological foundation in designing the simulation environment logic, based on principle of the classic "Snake" game, enhanced with business logic. DQN and PPO methods, implemented using Stable-Baselines3 library, are applied for training the reinforcement learning agents. The agent training process is divided into three stages – training, exploration and application during evaluating the service of regular and high-value clients, total rewards, and error frequency.

In the high-value client service environment, the PPO method demonstrated better results, averaging 14 regular and 4.1 high-value clients served, with a maximum episodic reward of 2011.2. In contrast, DQN averaged only 10.8 clients served, with a reward of 1076.4. The PPO method not only reduced the number of unserved high-value clients but also optimized movement logic, showing a 75th percentile value more 2.7 times higher than DQN method result.

In the risky order service environment, PPO results remained significantly superior. The average PPO reward during the application phase was 708.9, compared to DQN's 288.9. PPO almost entirely avoided obstacle collisions in the application phase, whereas DQN encountered at least one obstacle in every seventh episode. The PPO method not only generates higher rewards but also maintains behavioral consistency integrates high-value clients, and dynamically adapts to risky decision-making within its applied strategy. Meanwhile, DQN exhibits frequent local optimization and an inability to effectively adapt to business logic.

**Turinys**

## Lentelių sąrašas

## Paveikslų sąrašas

# Įvadas

Šiuolaikinėje verslo aplinkoje, kuriai būdingas dinamiškumas, neapibrėžtumas ir sprendimų sudėtingumas, strategijų testavimas tampa neatsiejama veiklos optimizavimo dalimi. Vien istoriniais duomenimis grįsti sprendimai nepakankamai atspindi kintančios rinkos sąlygas, todėl organizacijos vis dažniau siekia eksperimentuoti naudojant imitacinį modeliavimą. Vienas pažangiausių eksperimentinio strategijų testavimo metodų yra skatinamasis mokymasis, paremtas agento gebėjimu išmokti optimalią elgseną naudojant gaunamą grįžtamąjį ryšį iš struktūriškai apibrėžtos dinaminės sistemos, kuriose elgseną lemia tiek nuspėjami, tiek nenuspėjami veiksniai.

Šio tyrimo aktualumas – skatinamojo mokymosi taikymas imitaciniam verslo procesų modeliavimui sudaro sąlygas modeliuoti sudėtingus ir neapibrėžtus verslo procesus, naudojant sintetinius duomenis, kurie atvaizduoja eksperimentinių sąlygų rezultatus ir leidžia tirti procesus be realios rizikos, o žaidybinimas padeda įtraukti dinamiškos aplinkos elementus, būdingus realioms verslo situacijoms.

Tačiau praktikoje šio metodo taikymas susiduria su esminiu iššūkiu – ribota prieiga prie kokybiškų, tinkamai struktūruotų duomenų. Realūs verslo duomenys dažnai yra nepasiekiami dėl konfidencialumo arba netinkami dėl nepakankamos struktūros. Siekiant išvengti šių kliūčių, plačiai taikomas imitacinis modeliavimas, kuris leidžia generuoti sintetinius, eksperimentavimui tinkamus duomenis.

Imitacinis modeliavimas yra efektyvus metodas elgsenos analizei, kai siekiama įvertinti agento gebėjimą mokytis ir priimti sprendimus imituojamoje aplinkoje. Tokia aplinka atspindi verslo logikos veikimo principus – joje apibrėžiamos sąveikos taisyklės, būsenų perėjimai, tikslų struktūra bei grįžtamojo ryšio mechanizmai. Šis struktūrinis pagrindas leidžia taikyti skatinamojo mokymosi metodus, grindžiamus Markovo sprendimų procesu, kuriame agentas, remdamasis kaupiama patirtimi, išmoksta politiką, maksimizuojančią ilgalaikį atlygį.

Mokslinės literatūros analizė rodo, kad imitacinio modeliavimo, skatinamojo mokymosi ir sintetinių duomenų sąsaja jau sėkmingai taikoma nuo gamybos logistikos iki paslaugų sektoriaus strateginio valdymo sričių. Sudėtingų scenarijų modeliavimas apima įvairių sąlyginių taisyklių, apribojimų ar tikslų įvedimą į aplinką. Tokiose situacijose svarbu įvertinti, kaip agentas geba valdyti veiksmų sekas, išlaikyti pusiausvyrą tarp išteklių naudojimo ir rezultatų optimizavimo bei priimti nuosekliai pagrįstus sprendimus kintančioje aplinkoje.

Pagrindinis darbo tikslas – sukurti sužaidybintų verslo procesų imitacinį modelį ir optimizuoti strateginius sprendimus taikant skatinamojo mokymosi metodus bei sintetinius duomenis.

Pagrindiniai darbo uždaviniai tikslui pasiekti:

1. Apžvelgti mokslinę literatūrą, kurioje aprašomi skatinamojo mokymosi metodai ir sintetinių duomenų generavimo ir taikymo būdai.
2. Sukurti ir aprašyti verslo procesų imitavimo modelį ir imitacinę aplinką, pritaikius skatinamojo mokymosi metodus.
3. Pritaikyti skatinamojo mokymosi metodus sužaidybintų verslo procesų optimizavimo uždaviniui ir įvertinti jų efektyvumą.
4. Įvertinti eksperimentinių rezultatų tikslumą, suformuluoti rekomendacijas tolimesniam modelių bei metodų tobulinimui.

Problematikos ir literatūros apžvalgos skyriuje analizuojamas skatinamojo mokymosi ir sintetinių duomenų taikymas verslo procesų imitaciniame modeliavime. Metodų ir pasiūlytos metodikos skyriuje aprašoma skatinamojo mokymosi metodų struktūra bei jų taikymas strateginių sprendimų optimizavimui. Imitacinio modeliavimo pagrindu pasirinkta naudoti „Gyvatėlės“ žaidimo struktūrą, pritaikytą verslo procesų imitavimui. Tyrimo rezultatų analizės skyriuje pateikiama verslo procesų imitacinio modeliavimo ir agento mokymosi rezultatų analizė.

# 1. Problematikos ir literatūros apžvalga

## 1.1. Verslo procesų imitacinio modeliavimo reikšmė šiuolaikiniuose verslo modeliuose

Verslo procesų imitacinis modeliavimas tampa vienu pagrindinių įrankių šiuolaikinių organizacijų veikloje, siekiant didinti veiklos efektyvumą, mažinti operacinę riziką ir priimti duomenimis grįstus sprendimus. Imitacinis modeliavimas leidžia sistemingai analizuoti, kaip organizacijos veiksmai, taisyklės ir tarpusavio sąveikos lemia galutinį rezultatą. Tai suteikia galimybę eksperimentuoti be tiesioginio poveikio realiems procesams. Verslo procesų imitavimas ypač naudingas, kai eksperimentavimas realiomis sąlygomis yra pernelyg rizikingas ar brangus.

Tinkamai parinkti modeliai leidžia, kintant aplinkos sąlygoms, prognozuoti galimus rezultatų scenarijus ir įvertinti netikėtus ar sunkiai iš anksto numatomus veiksnius. Remiantis Cassidy ir kt. sistemine apžvalga, agentų pagrindu sukurti modeliai leidžia modeliuoti agentų elgseną, jų sprendimų priėmimo logiką ir tarpusavio sąveiką, kuria grindžiamas sistemos elgesys, o sistemų dinamikos metodai suteikia galimybę analizuoti grįžtamojo ryšio ciklus bei laiko vėlavimus sveikatos sistemose [1]. Praktikoje agentų pagrindu sukurti modeliai buvo taikyti skirtinguose sektoriuose. Tyrime analizuotas agentų pagrindu sukurtas modelis buvo pritaikytas farmacijos gamybos ir pardavimų tiekimo grandinėje, siekiant ištirti gamybos padalinių ir pardavimo skyrių sąveiką bei jos poveikį tiekimo tinklui [2]. Gauti rezultatai parodė, kad esama tiekimo grandinė nėra pakankamai lanksti ir negali greitai reaguoti į pokyčius. Modelis parodė, kad gavus itin didelį užsakymą, esant ribotiems ištekliams, užsakymo įvykdyti nepavyktų dėl laiko ir žaliavų trūkumo. Teranu ir kt. atliktame tyrime agentų pagrindu sukurti modeliai buvo taikyti mažmeninėje prekyboje. Sukurtas klientų imitavimo įrankis *ABISS* imitavo individualių pirkėjų judėjimą parduotuvėje, apsipirkimo sprendimus bei reakciją į parduotuvės aplinkos pokyčius [3]. Remiantis realaus prekybos centro duomenimis buvo pritaikytas modelis, po to atliktas imitacinis modeliavimas, keičiant parduotuvės išdėstymą ir reklamos vietas. Tyrimo rezultatai parodė, kad pirkėjų srautai ir pardavimai priklauso nuo parduotuvės išplanavimo bei vidinių reklamos priemonių išdėstymo. Pavyzdžiui, optimizavus prekių lentynų išdėstymą, tam tikrų prekių pardavimai pastebimai išaugo dėl padidėjusio pirkėjų pritraukimo į atitinkamas prekybos centro zonas. Agentų pagrindu sukurtų modelių taikymas tyrimuose skiriasi agentų interpretacija, tačiau nepaisant šio skirtumo, visais minėtais atvejais imitacinis modeliavimas leido įvertinti sistemines pasekmes, kylančias iš verslo procesų, ir saugiai eksperimentuoti su galimais pokyčiais, nesukeliant rizikos realiai veiklai.

Diskrečiųjų įvykių modeliavimas yra vienas iš plačiausiai taikomų metodų verslo procesų analizei, kai būtina tiksliai modeliuoti įvykių seką, procesų trukmę ir išteklių panaudojimą. Šis metodas aprašo sistemą kaip įvykių seką, vykstančią tam tikrais laiko momentai, kur kiekvienas įvykis keičia sistemos būseną. Diskrečiųjų įvykių imitavimas išplėtotas sveikatos priežiūros sektoriuje. Sisteminis 483 straipsnių tyrimas parodė, kad 1981–2014 m. laikotarpiu šis metodas naudotas 22 skirtingose sveikatos priežiūros srityse, įskaitant pacientų srautų, išteklių paskirstymo ir gydymo procesų analizę [4]. Viename iš aptariamų straipsnių sukurtas modelis, skirtas intensyvios terapijos skyriaus veiklos analizei, leido imituoti pacientų atvykimus, gydymo trukmę, reikalingų lovų skaičių, remiantis pacientų įrašais [5]. Procesų imitavimas leido įvertinti, kad padidinus lovų skaičių nuo 23 iki 28, vidutinis lovų užimtumas sumažėja iki 70 %, o tai reiškia žymiai mažesnę perkrovos riziką normaliomis sąlygomis. Vis dėlto diskrečiųjų įvykių modeliavimas parodė, kad jei pacientų srautai kasmet didėtų maždaug 4 %, net ir 28 lovų skyrius greitai taptų perpildytas. Šis modeliavimas suteikė tikslingos informacijos vadovybei, planuojant investicijas į infrastruktūrą ir procesų tobulinimą

remiantis modelio prognozėmis. Zupano ir Herakovičiaus atliktame tyrime, taikant diskrečiųjų įvykių modeliavimą, atlikta gamybos įmonės dviejų surinkimo linijų analizė, siekiant optimizuoti jų darbo efektyvumą [6]. Modeliavimo metu įvertinti technologinių operacijų vykdymo trukmės rodikliai, darbo išteklių paskirstymo tvarka bei įrangos naudojimo intensyvumas. Remiantis gautais rezultatais, atliktas surinkimo linijų darbo eigos optimizavimas, kuris reikšmingai padidino gamybos produktyvumą, o įmonės finansiniai rezultatai pasikeitė iš nuostolio į pelną. Bottanio ir Montanario atliktame tyrime sudarytas penkių lygių tiekimo grandinės imitacinis modelis, kuriame buvo modeliuojami gamintojo, tarpinių paskirstymo grandžių ir mažmeninės prekybos tarpusavio ryšiai [7]. Tyrimo eigoje buvo nagrinėtos 30 skirtingų tiekimo grandinės konfigūracijų, kuriose buvo sistemingai keičiami pagrindiniai parametrai. Rezultatai parodė, kad integruotas informacijos perdavimas ir centralizuota užsakymų valdymo sistema mažina paklausos svyravimo efektą ir optimizuoja bendrąsias logistikos sąnaudas. Diskrečiųjų įvykių modeliavimas leidžia tiksliai imituoti operacinių procesų dinamiką. Pagrindinis šio metodo privalumas yra gebėjimas identifikuoti veiklos sutrikimus, taip pat testuoti pokyčius prieš juos įgyvendinant praktikoje.

Verslo procesuose Monte Karlo metodas dažnai naudojamas situacijose, kai būtina įvertinti galimų išlaidų, pajamų ar procesų trukmių svyravimus ir priimti sprendimus, remiantis rizikos pasiskirstymu. Tyrime, kuriame nagrinėta maisto pramonė, siekiant optimizuoti metinį gamybos biudžetą ir valdyti finansinę riziką naudojant Monte Karlo modeliavimą, remtasi istoriniais duomenimis ir tikimybinėmis tankio funkcijomis [11]. Lyginant su prieš tai taikyta gamybos biudžeto planavimo strategija, Monte Karlo modeliavimo rekomenduotas gamybos planas padidino prognozuojamą pelną daugiau kaip 30 procentinių punktų. Modelis taip pat parodė rizikos veiksnius, kurie anksčiau nebuvo identifikuoti.

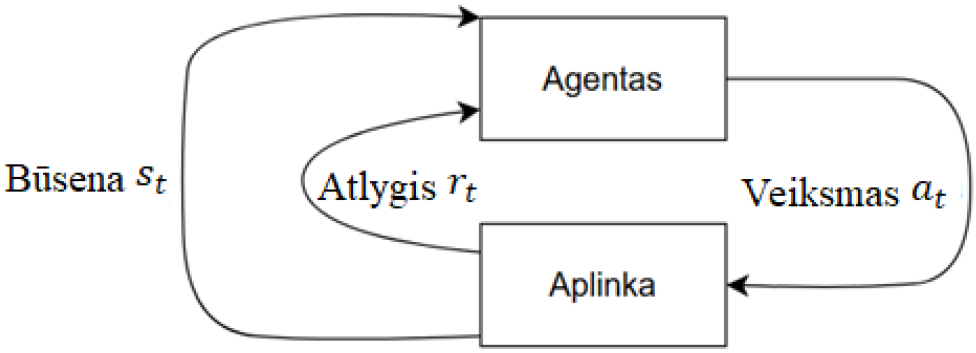
Skatinamojo mokymosi metodai leidžia agentui iteratyviai įgyti veikimo strategiją per aplinkos stebėseną ir gaunamą grįžtamąjį ryšį. Šie metodai glaudžiai susiję su skaitmeninių analogų ir sintetinių duomenų generavimo technologijų plėtra. Skaitmeniniai analogai, apibrėžiami kaip realių sistemų dinaminiai modeliai, naudojami kuriant realiuoju laiku atnaujinamas imitacines aplinkas, paremtas tiek istoriniais, tiek sintetiniais duomenimis [8, 9]. Skatinamojo mokymosi taikymas buvo nagrinėtas plieno gamybos sektoriuje [10]. Tyrime buvo taikomas skatinamojo mokymosi metodas su ansambline mokymosi struktūra, skirtas juostinio plieno valcavimo staklių plokštuminiam valdymui optimizuoti. Tai sudėtingas realaus laiko valdymo uždavinys, kuriame tradiciniai automatinio valdymo metodai dažnai reikalauja sudėtingo parametrų derinimo remiantis inžinierių patirtimi. Šiame procese buvo taikomas giliojo skatinamojo mokymosi metodas – PPO (*angl. Proximal Policy Optimization*), kurio veikimo efektyvumą pagerino ansamblio mokymosi struktūra. Vienu metu keli skatinamojo mokymosi agentai mokėsi ir po kiekvieno mokymosi epizodo būdavo išsaugoma tik geriausiai pasirodžiusio agento patirtis, kuria atnaujinami kitų agentų parametrai. Modeliavimo rezultatai parodė, kad skatinamojo mokymosi pagrindu sukurta valdymo sistema pasižymėjo mažesne produkcijos kokybės rodiklių dispersija ir efektyvesniu sistemos reagavimu į išorinių ir vidinių parametrų svyravimus, lyginant su tradiciniais automatinio valdymo metodais. Sukurtas modelis sudarė sąlygas vykdyti procesų prisitaikymą realiuoju laiku, įtraukiant kintančių žaliavų savybių bei įrangos eksploatacinių parametrų įtakos vertinimą.

Imitacinio modeliavimo metodai, tokie kaip diskrečiųjų įvykių modeliavimas, agentų pagrindu sukurti modeliai, sistemų dinamika, Monte Karlo metodas ir skatinamasis mokymasis, taikomi verslo procesų analizėje. Šie metodai skiriasi kintamųjų aprašymo detalumu, tačiau visi sudaro prielaidas

kiekybiniam galimų būsenų įvertinimui. Metodų derinimas leidžia sudaryti modeliavimo sistemas, kuriose taikoma scenarijų analizė, parametrų jautrumo vertinimas ir optimalių strategijų atranka.

### 1.2. Skatinamojo mokymosi bendroji teorija

Skatinamasis mokymasis yra vienas iš trijų pagrindinių mašininio mokymosi paradigmų, greta prižiūrimojo ir neprižiūrimojo mokymosi. Šis metodas taikomas sprendimo priėmimo užduotyse, kai aplinkos grįžtamasis ryšys nėra momentinis, o sprendimų pasekmės išryškėja per tam tikrą laiką. Tokiame kontekste modelis turi ne tik apdoroti duomenis, bet ir mokytis iš ilgalaikių savo veiksmų padarinių. Skatinamojo mokymosi pagrindą sudaro agento sąveikos su aplinka ciklas. Agentas atlikdamas veiksmus, gauna atlygį, o sukaupta patirtis naudojama veiksmų politikos optimizavimui, siekiant maksimalios kaupiamosios naudos per ilgalaikį laikotarpį. Skatinamojo mokymosi veikimo logika dažnai formuojama pagal Markovo sprendimų proceso struktūrą, kuriai būdinga tai, kad agento sprendimų kelias grindžiamas pagal būsenų, atlygio reikšmių ir tikimybinio perėjimo taisyklių sąveiką. Agento elgsena formuojama vadovaujantis pasirinkta strategija, o tikslas – ilgainiui padidinti bendrą gautą naudą (žr. 1 pav.).



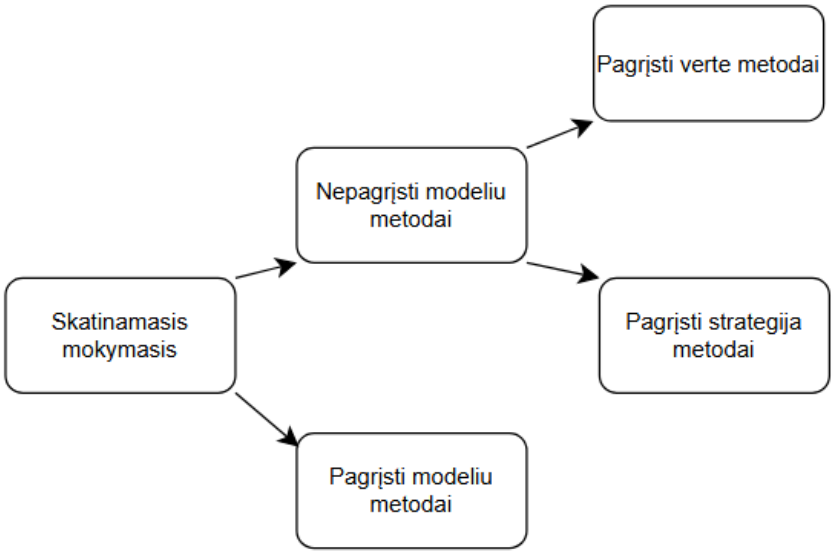

**1 pav.** Agento ir aplinkos sąveikos ciklas skatinamojo mokymosi sistemoje

Skatinamasis mokymasis išsiskiria iš kitų mašininio mokymosi paradigmų tuo, kad jo pagrindinis tikslas nėra susijęs su regresijos, klasifikacijos ar duomenų grupavimo užduočių sprendimu. Vietoj to, šis metodas orientuotas į atlygiu grįstų sprendimų paiešką. Agento veikimas formuojamas nuosekliai kaupiant patirtį per bandymų ir klaidų principu grindžiamą sąveiką su aplinka. Nuolatinė pusiausvyros paieška tarp tyrinėjimo (*angl. exploration*) ir išnaudojimo (*angl. exploitation*) leidžia agentui prisitaikyti dinamiškoje aplinkoje ir palaipsniui tobulinti sprendimų strategiją. Kiekvienas skatinamojo mokymosi metodas veikia dviem etapais:

1. Mokymosi fazė, kurioje agentas sąveikauja su aplinka, kaupia informaciją apie savo veiksmų pasekmes ir iš jų mokosi. Šis etapas apima nuoseklų bandymų ir klaidų ciklą, kuris leidžia suformuoti patirties pagrindu veikiančią sprendimų priėmimo sistemą.
2. Taikymo fazė, kai sukaupta patirtis yra naudojama praktiniam sprendimų įgyvendinimui, tačiau pats modelis tuo metu jau nebėra atnaujinamas – agentas veikia remdamasis iki tol suformuota strategija.

Skatinamojo mokymosi metodai klasifikuojami pagal tai, ar jie remiasi aplinkos modeliu. Ši klasifikacija leidžia atskirti modeliu pagrįstus metodus, kurie naudoja iš anksto apibrėžtą perėjimų

tikimybę ir atlygio funkciją, ir modeliu nepagrįstus metodus, kurie mokosi iš tiesioginės sąveikos su aplinka (žr. 2 pav.).




**2 pav.** Skatinamojo mokymosi metodų klasifikacija pagal modelio pobūdį ir politikos taikymą

Modeliu pagrįsti metodai turi galimybę imituoti aplinkos elgseną ir numatyti skirtingų veiksmų pasekmes dar prieš juos atliekant. Tai leidžia priimti sprendimus pagal prognozuojamus ilgalaikius rezultatus, tačiau tokių metodų tikslumas stipriai priklauso nuo modelio kokybės – netiksliai apibrėžta aplinka gali lemti klaidingus rezultatus. Šie metodai tinka verslo procesų imitacijoje, kai modeliuojamos gerai suprantamos sistemos, leidžiančios tiksliai apibrėžti būsenų perėjimus ir tikėtinas pasekmes. Tuo tarpu nepagrįstų modeliu metodų veikimas paremtas tiesiogine sąveika su aplinka, be iš anksto žinomo perėjimų modelio ar atlygio funkcijos, todėl sprendimai formuojami remiantis sukaupta patirtimi mokymosi eigoje. Tai leidžia mokytis net ir sudėtingose, neapibrėžtose ar dažnai kintančiose sistemose [13]. Atsižvelgiant į tai, kad agentas priima sprendimus, modeliu nepagrįsti metodai skirstomi į dvi grupes:

- Verte pagrįsti metodai naudoja laiko skirtumo (*angl. temporal difference*) mokymąsi, kurio metu vertinama, kiek naudos duos konkretus veiksmas, atsižvelgiant į būsimas būsenas.
- Strategija pagrįsti metodai modeliuoja veiksmų pasirinkimo politiką, siekdami tiesiogiai rasti optimalią strategiją.

Murphy‘io 2024 metais atliktame tyrime nustatyta, kad sprendimų erdvėse, kuriose veiksmai ir būsenos yra tęstiniai, o tarpinių būsenų vertinimas reikalauja reikšmingų skaičiavimo išteklių, tradiciniai sprendimų priėmimo metodai tampa neefektyvūs dėl sistemos neapibrėžtumo, kintamumo ir didelės būsenų erdvės [14]. Tokiais atvejais tinkamais laikomi skatinamojo mokymosi metodai, kurie, remdamiesi nuolatine sąveika su aplinka, geba optimizuoti veiksmų pasirinkimo strategiją kintančiose sistemose.

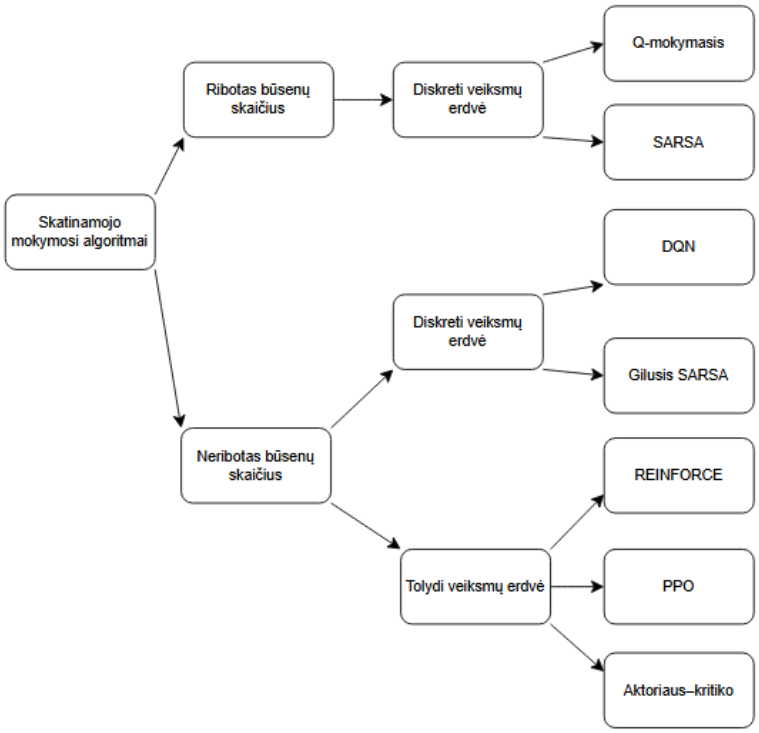
Skatinamasis mokymasis taikomas praktiniuose modeliuose, veikiančiuose realiomis sąlygomis, dėl gebėjimo dinamiškai reaguoti į aplinkos pokyčius ir prisitaikyti prie jų. Jis sėkmingai integruojamas

su verslo procesų imitaciniais modeliais, sudarant sąlygas kurti pažangias, adaptacines strategijas, kuriose geba ne tik reaguoti į pokyčius, bet ir iš jų mokytis. Tokios sistemos leidžia ne tik optimizuoti procesus, bet ir padidinti organizacijų gebėjimą veikti neapibrėžtose, besikeičiančiose aplinkose. Skatinamasis mokymasis tampa svarbiu įrankiu sprendimų palaikymo sistemose, kuriose svarbus ne vien tikslumas, bet ir lankstumas, operatyvumas bei gebėjimas kaupti ir naudoti patirtį.

### 1.2.1. Skatinamojo mokymosi metodų apžvalga verslo procesų modeliavime

Skatinamojo mokymosi metodai leidžia agentui įvertinti savo veiksmų pasekmes ne iš anksto, o per praktinę sąveiką su sistema, palaipsniui optimizuojant savo strategiją pagal didžiausią ilgalaikę naudą suteikiančius veiksmus. Atsižvelgiant į taikomą mokymosi strategiją ir turimą informaciją apie aplinką, skatinamojo mokymosi metodai skirstomi į tris pagrindines kategorijas: pagrįsti verte, pagrįsti strategija ir pagrįsti modeliu metodai [14, 15]. Kiekviena iš šių kategorijų pasižymi skirtingais taikymo principais ir tinkamumu įvairių sudėtingumo lygių uždaviniams.

Be šios klasifikacijos, metodai taip pat skirstomi pagal veiksmų erdvės pobūdį, būsenų skaičiaus apribojimus ir naudojamą architektūrą. Metodų tarpusavio struktūriniai ryšiai ir jų praktinis pritaikomumas pavaizduoti klasifikacijos schemoje (žr. 3 pav.).




**3 pav**. Skatinamojo mokymosi metodų klasifikacija pagal būsenų ir veiksmų erdves

Vertės pagrindu grindžiami metodai optimizuoja sprendimus, vertindami būsenoje prognozuojamą ilgalaikį atlygį. Agentas neturi tiesioginės strategijos, vietoj to jis renkasi veiksmą, kurio numatoma vertė yra didžiausia. Vienas iš klasikinių šios kategorijos metodų yra Q-mokymasis, leidžiantis iteratyviai atnaujinti Q-reikšmes pagal aplinkos grąžinamą atlygį. Šio metodo paprastumas ir stabilumas leidžia jį plačiai taikyti uždaviniuose, kai sistemai būdinga ribota būsenų ir veiksmų

kombinacijų aibė. Ši problema sprendžiama taikant gilųjį Q-mokymąsi (DQN), kuris vietoje Q-reikšmių matricos naudoja neuroninį tinklą vertės funkcijai apskaičiuoti [16]. Tai leidžia taikyti metodą sudėtingose, net vaizdiniais duomenimis pagrįstose aplinkose. Tokie metodai yra tinkami verslo procesų imitaciniuose modeliuose, kuriuose reikalingas nuoseklus, grįžtamuoju ryšiu paremtas sprendimų sekų optimizavimas esant ribotai ir gerai apibrėžtai veiksmų erdvei.

Kita metodų grupė grindžiama tiesioginiu veiksmų politikos modeliavimu. Šiuo atveju, agentas nenaudoja aiškiai apibrėžtos vertės funkcijos, o mokosi optimizuoti tikimybiniu principu grindžiamą strategiją, kuri kiekvienoje būsenoje generuoja tikimybes skirtingiems veiksmams. Tokie metodai leidžia išvengti sudėtingo tarpinių vertės įverčių skaičiavimo, kuris dažnai tampa neefektyvus arba netinkamas taikant jį tęstinėse ar itin didelėse veiksmų erdvėse [17]. Proksimalinis politikos optimizavimas (PPO) yra plačiai taikomas politika grįstų metodų grupės metodas, kurio politikos atnaujinimai ribojami nustatyta reikšmių riba, siekiant išlaikyti mokymosi proceso stabilumą ir išvengti per didelių nuokrypių nuo ankstesnės strategijos. Politika grįsti metodai dažnai įgyvendinami aktoriaus-kritiko struktūroje. Aktorius generuoja veiksmus pagal esamą politiką, o kritikas įvertina jų naudą, taip suteikdamas mokymosi signalą. Tokia struktūros architektūra leidžia efektyviau spręsti uždavinius su sudėtinga veiksmų erdve ir paspartinti strategijos priartėjimą prie optimalios politikos.

Skatinamojo mokymosi metodai taip pat gali būti grindžiami aplinkos modeliu, kuris apibrėžia būsenų perėjimo taisykles ir atlygio funkciją. Šio tipo metodai leidžia agentui planuoti veiksmų sekas ne tik pagal realią patirtį, bet ir pasitelkiant imitavimo procesą. Turint pakankamai tikslų aplinkos dinamikos modelį, galima taikyti optimizavimo metodus geriausiai veiksmų sekai identifikuoti, o politikos sudarymas tampa skaičiavimo požiūriu efektyvesnis, nes mažėja priklausomybė nuo empirinių duomenų gavimo per tiesioginę sąveiką su sistema [18]. Modeliu pagrįsti metodai yra tinkami, kai sistemos būsenų perėjimo funkcija ir tikimybiniai dėsningumai yra iš anksto žinomi arba gali būti tiksliai apskaičiuojami. Tačiau pagrindinis šių metodų apribojimas yra jų jautrumas modeliavimo netikslumams. Modelio struktūros netikslumai, atsirandantys dėl netinkamai parinktų parametrų ar ribotos stebėjimų imties, gali nuosekliai plėtotis modeliavimo procese ir lemti sprendimų nuokrypį nuo teorinio optimumo. Dėl to siekiant stabilios prisitaikančios sprendimų sistemos, būtina įvertinti modelio tikslumą ir patikrinti jo gebėjimą tiksliai atkartoti stebimų procesų dėsningumus.

Skatinamojo mokymosi metodai, iš pradžių nagrinėti teoriniuose mašininio mokymosi ir kontrolės sistemų modeliuose, palaipsniui įsitvirtina kaip praktiniai įrankiai sprendimų optimizavimui sudėtingose verslo aplinkose. Šie metodai leidžia modeliuoti nuoseklaus sprendimų priėmimo procesus, kuriuose sprendimai daro ilgalaikį poveikį ir turi būti priimami reaguojant į neapibrėžtą bei dinamiškai kintančią verslo aplinką. Vienas iš skatinamojo mokymosi metodų taikymo pavyzdžių yra atsargų valdymo optimizavimas decentralizuotose tiekimo grandinėse [19]. Tyrime pasiūlytas gilusis Q-mokymosi metodas, skirtas „alaus žaidimo“ scenarijui modeliuoti.

Konkurencinėje kainodaros aplinkoje, kurioje vartotojų elgsena kinta priklausomai nuo kainos, o konkurentų priimami sprendimai nėra iš anksto žinomi, buvo analizuojami skirtingi skatinamojo mokymosi metodai – PPO ir DQN [20]. Šiame kontekste vertinta, kaip šie metodai sugeba prisitaikyti prie sprendimų neapibrėžtumo, ribotų skaičiavimo išteklių ir kintančios aplinkos. Nustatyta, kad aktoriaus-kritiko metodai efektyviau stabilizuoja politiką, išvengdami pernelyg rizikingų ar atsargių sprendimų, kurie dažniau pasitaiko klasikiniuose metoduose. Dar sudėtingesnis optimizavimo uždavinys kyla tuomet, kai dinaminės kainodaros sprendimai integruojami su atsargų valdymo

strategija greitai gendančių prekių tiekimo grandinėse. PPO metodas buvo pritaikytas verslo scenarijuje, kuriame agentas turėjo priimti sprendimus esant ribotam prekių galiojimo laikui ir galimiems tiekimo trikdžiams [21]. Sprendimų erdvė buvo sudaryta kaip dviejų veiksmų aibė, apimanti kainų koregavimo bei atsargų papildymo sprendimus, kurių bendras optimizavimas leido prisitaikyti prie paklausos dinamikos ir sumažinti veiklos neefektyvumą. Politika išmoko subalansuoti užsakymo sąnaudas, prekių laikymo kaštus ir nuostolius dėl neįvykdytų užsakymų, o PPO metodas leido sumažinti skaičiavimo išteklių poreikį maždaug 75 %, lyginant su klasikiniais dinaminio programavimo metodais. Tai parodė, kad skatinamasis mokymasis gali tapti realiai pritaikomu sprendimų įrankiu net ir sudėtinguose logistikos uždaviniuose.

Skatinamojo mokymosi metodai buvo pritaikyti analizuojant decentralizuotų tiekimo grandinių dinamiką, naudojant klasikinį alaus paskirstymo žaidimo modelį (*Beer Distribution Game*), kuriame pasireiškia paklausos svyravimų poveikis skirtingoms tiekimo grandinėms [19]. Šio proceso imitacinėje aplinkoje agentas turėjo prisitaikyti prie kitų tiekimo grandinės dalyvių veiksmų, kurie dažnai pasižymėjo ribotu racionalumu. Rezultatai parodė, kad skatinamojo mokymosi agentas išmoko formuoti užsakymų strategijas, kurios beveik visiškai panaikino paklausos srautų disbalansą tiekimo grandinėje. Tai reikšminga, nes agentas sugebėjo veikti decentralizuotoje sistemoje be išankstinio koordinavimo. Be to, išmokta politika išliko stabili esant kaštų struktūros pokyčiams ir galėjo būti pritaikyta skirtingose tiekimo grandinės dalyse, beveik nereikalaujant papildomo mokymo. Verslo procesų optimizavimas apima ir tokias sritis kaip gamybos ciklų planavimas bei įrangos prevencinės priežiūros planavimas. Optimizavimo procesas grindžiamas atlygio funkcija, kurioje įtraukiami tiek ekonominiai kaštai, tiek operacinio stabilumo kriterijai [22]. Tyrimo rezultatai rodo, kad tokia metodologija leidžia padidinti tiekimo sistemos efektyvumą ir sumažinti atsargų lygio dispersiją, atsirandančią dėl blogos sąveikos tarp tiekimo grandinės struktūrinių komponentų.

Tokiuose scenarijuose agento mokymasis leidžia pereiti nuo fiksuotų taisyklių prie elgsenos, kuri prisitaiko prie laike kintančių aplinkos veiksnių. Alaus paskirstymo žaidimo modelis buvo integruotas į internetinę platformą, kurioje jis veikė kaip autonominė sprendimų priėmimo sistema, realiuoju laiku reaguojanti į vartotojų pateikiamas užklausas. Taikant skatinamojo mokymosi metodus sudėtingų sprendimų priėmimo procesuose, sudaromos sąlygos sistemoms prisitaikyti prie aplinkos pokyčių, nereikalaujant keisti priimamų sprendimų formavimo logikos.

### 1.2.2. Skatinamojo mokymosi taikymai verslo procesuose

Klientų užklausų paskirstymo uždavinys gali būti aprašomas kaip Markovo sprendimų procesas, kurio struktūroje agentas kiekvienai įeinančiai užklausai priskiria aptarnavimo kanalą, siekdamas maksimizuoti sistemos efektyvumą. Būsena apima kliento charakteristikas bei esamą kanalų apkrovą, o veiksmas reiškia sprendimą priskirti užklausą vienam iš galimų aptarnavimo kanalų. Taikant giluįjį Q-mokymąsi, sistema mokosi efektyviai paskirstyti užklausas, adaptuojasi prie pasikeitusių klientų srautų bei mažina kanalų perkrovos tikimybę, kartu didindama paslaugų prieinamumą ir klientų pasitenkinimą [23].

Panašiu principu skatinamasis mokymasis taikomas ir darbo jėgos planavimo kontekste, ypač tokiose srityse kaip skambučių centrų veikloje ar logistikos koordinavime, kur būtina greitai reaguoti į dinamiškai besikeičiančius užklausų srautus. Rekurentinių neuroninių tinklų pagrindu sukurta sistema buvo naudojama prognozuoti būsimą skambučių srautą, o DQN metodas optimizavo operatorių pamainų pradžios laiką ir darbuotojų skaičių kiekvienai pamainai. Sistemos veikimas

pranoko patyrusių vadovų sudarytus grafikus. Taip pat buvo pasiekta aukštesnė aptarnavimo kokybė ir reikšmingai sutrumpintas vidutinis laukimo laikas [24].

Kainodaroje skatinamasis mokymasis taikomas siekiant automatizuoti sprendimų priėmimą esant kintančiai paklausai, konkurencinei aplinkai ir dinamiškai vartotojų elgsenai. Tokiose situacijose dažnai modeliuojamas Markovo sprendimų procesas, kuriame būsena apima laiko momentą, atsargų kiekį ir paklausos parametrus, o veiksmas atitinka konkretaus produkto ar paslaugos kainos nustatymą. Atliekant duomenimis grįstą analizę prioritetinių paslaugų kainodaroje, skatinamojo mokymosi agentas sprendė, kokio dydžio papildomą mokestį taikyti už privilegijuotą prieigą, kartu optimizuodamas šių paslaugų paskirstymą [25]. Šiam tikslui pasiekti buvo naudojamas klasikinis Q-mokymosi metodas ir politikos gradientų metodai, leidžiantys efektyviai veikti tęstinėje veiksmų erdvėje. Remdamasis sukaupta patirtimi bei prisitaikydamas prie kintančių aplinkos sąlygų, agentas pranoko žmogaus sudarytus kainodaros sprendimus tiek generuojamų pajamų, tiek vartotojų pasitenkinimo rodiklių atžvilgiu.

Finansų sektorius – viena sudėtingiausių verslo sričių, kur skatinamojo mokymosi pritaikymas reikalauja itin pažangių metodų. Investicijos portfelio optimizavime, kur būsenos yra tolydžios, o rinkos dinamika nepastovi, sėkmingai taikomi gilieji agento-kritiko metodai. Agentai, treniruojami naudojantis istorinių finansinių duomenų imitaciniu modeliavimu, geba formuoti strategijas, prisitaikančias prie kintančių rinkos sąlygų. Šios strategijos pasirodo pranašesnės už fiksuotų taisyklių ir klasikines optimizavimo strategijas, kai reikalingas greitas reagavimas į rinkos pokyčius ir nuoseklus rizikos kontrolės palaikymas. Skatinamojo mokymosi pagrindu veikiantys agentai demonstravo gebėjimą generuoti didesnę grąžą nei *Dow Jones* pramonės indeksas, tuo pačiu sumažindami portfelio grąžos dispersiją [25].

Skatinamojo mokymosi taikymas versle gali reikšmingai pagerinti sprendimų priėmimo efektyvumą, tačiau šių sistemų priklauso nuo duomenų patikimumo, tinkamų būsenų ir veiksmų erdvės modeliavimo bei tinkamai apibrėžtos atlygio funkcijos. Kadangi agentas mokosi iš sąveikos su aplinka, jis yra pažeidžiamas netiesioginių ar neišsamių duomenų, o tai gali lemti strategijų, kurios neužtikrina maksimalaus kaupiamojo atlygio išmokimą. Dėl to, siekiant užtikrinti šių metodų patikimumą, būtina ne tik taikyti pažangius metodus, bet ir užtikrinti aukštos kokybės duomenų paruošimą bei sistemingą praktinį metodų tikrinimą realiomis sąlygomis.

### 1.3. Sintetinių duomenų apibrėžimas ir generavimo metodai

Skatinamojo mokymosi aplinkoje sintetiniai duomenys yra sistemingai generuojami agentui vykdant veiksmų sekas imitacinėje aplinkoje. Kiekvienas agento veiksmas ir su juo susijusi būsena bei gautas grįžtamasis atlygis sudaro mokymuisi būtinus duomenis, kurie iteratyviai kaupiami ir naudojami elgsenos politikos atnaujinimui. Bendresniame kontekste sintetiniai duomenys suprantami kaip dirbtinai sugeneruoti duomenys, kurie išlaiko esmines realiųjų duomenų struktūrines ir statistines savybes, tačiau neturi tiesioginio ryšio su tikrais duomenimis.

Vienas iš bazinių metodų sintetinių duomenų generavimui yra Bajeso tinklų pagrindu sudaryti tikimybiniai modeliai. Šie modeliai išmoksta kintamųjų tikimybinį pasiskirstymą ir tarpusavio priklausomybes, o tai leidžia užtikrinti naujai sugeneruotų duomenų kokybę. Bajeso tinklų pagrindu veikiantys įrankiai tokie kaip *DataSynthesizer*, sukuria duomenis su aiškiai apibrėžtomis kintamųjų aibėmis ir įrašų eilutėmis, išsaugodami vidinę priklausomybių struktūrą tarp kintamųjų [20].

Generatyviniai priešininkų tinklai (*angl. Generative Adversarial Networks*) sudaro atskirą sintetinių duomenų generavimo metodų klasę, pasižyminčią gebėjimu kurti tikroviškos struktūros duomenų rinkinius. Generatyvinių priešininkų tinklų (GAN) architektūra grindžiama dviejų neuroninių tinklų sąveika, kur generatorius yra atsakingas už duomenų imitavimo procesą, o diskriminatorius siekia atskirti sintetinius duomenis nuo realių. Ši konkurencinė struktūra sudaro sąlygas iteratyviam mokymuisi, kurio metu generatorius gerina išvesties kokybę, prisitaikydamas prie diskriminatoriaus pateikiamo grįžtamojo ryšio. Specializuoti GAN variantai, tokie kaip TGAN orientuoti į laiko eilučių struktūrą, CTGAN naudojami kurti duomenų lenteles [26]. Vertinant sintetinių duomenų kokybę, taikomi kelių tipų rodikliai, apimantys tiek statistinį panašumą, tiek praktinį pritaikomumą. Viena vertinimo kryptis susijusi su realių ir sintetinių duomenų pasiskirstymų palyginimu, siekiant nustatyti, kiek sintetiniai duomenys atspindi pradinių duomenų struktūrą. Taip pat tikrinama, ar tinkamai atkuriama trūkstamų reikšmių struktūra. Kiekybinio naudingumo įvertinimas atliekamas lyginant, kaip tiksliai sintetiniai duomenys leidžia išmokyti modelius, lyginant su mokymu naudojant realius duomenis [27]. Sintetiniai duomenys gali tiek mažinti, tiek didinti duomenų disbalansą. Jie gali būti naudingi sprendžiant klasių disbalanso problemas – generuojant daugiau mažumos klasės pavyzdžių, kad modelis netaptų asimetriškas. Tačiau jei pradinis duomenų rinkinys pasižymi tendencingumu, o imitacinis modelis šią asimetriją atkuria, problema išlieka. Todėl yra kuriami metodai, kurių pagrindinis tikslas yra pašalinti nepageidaujamas koreliacijas ir išlaikyti reikšmingus ryšius [28].

Sintetinių duomenų generavimo metodai naudojami siekiant užtikrinti duomenų konfidencialumą, kai dirbama su jautriais informacijos šaltiniais, tokiais kaip sveikatos apsaugos ar finansiniai duomenys. Tokie metodai leidžia generuoti duomenų rinkinius, kurie statistiškai atspindi originalių duomenų struktūrą ir kintamųjų tarpusavio priklausomybes, tačiau neleidžia identifikuoti konkrečių asmenų informacijos. Tai leidžia kurti ir testuoti analitinius modelius be tiesioginės prieigos prie realių duomenų, taip atitinkant privatumo ir duomenų apsaugos teisinių reikalavimų nuostatas. Sintetiniai duomenys taikomi telemedicinos sistemų plėtrai, kibernetinio saugumo sprendimams ir sukčiavimo rizikos vertinimo metodams finansų srityje [29]. Platesnėje analizėje išskiriamos pagrindinės sritys, kuriose sintetinių duomenų generavimas taikomas kaip būdas išspręsti informacijos prieinamumo ir privatumo problemas. Tarp jų – medicininių vaizdų analizė, ligų plitimo modeliavimas, kibernetinio saugumo grėsmių imitavimas. Kiekviena iš šių sričių pasižymi unikaliu duomenų tipu, reikalaujančiu specialių generavimo ir vertinimo metodų:

**1 lentelė.** Sintetinių duomenų generavimo taikymas

| Taikymo sritis | Tikslas | Duomenų pobūdis | Pagrindiniai iššūkiai |
|---|---|---|---|
| Medicina | Saugių sintetinių pacientų duomenų panaudojimas siekiant pagerinti medicininius tyrimus ir personalizuotą sveikatos priežiūrą [30]. | Elektroniniai sveikatos įrašai, medicininiai vaizdai, tekstiniai duomenys. | Sintetinių duomenų informacijos kokybės užtikrinimas, teisiniai ir etikos reikalavimai. |
| Švietimas | Sukurti privatumo nepažeidžiančius švietimo duomenų rinkinius mokymosi procesų analizei ir švietimo technologijų tobulinimui [31]. | Mokymosi valdymo sistemų fiksuojami duomenys, studijų rezultatai, edukaciniai testai ar užduotys. | Suderinti duomenų naudingumą su griežtais privatumo reikalavimais. |
| Autonominis vairavimas | Autonominių transporto sistemų mokymui ir testavimui generuoti įvairius vairavimo scenarijus imituojančius duomenų rinkinius [32]. | Kameros vaizdai, sintetiniai miesto gatvių vaizdai, nuotolio matavimo jutiklių duomenys naudojant lazerio impulsus objektų atstumui ir aplinkos geometrijai nustatyti. | Užtikrinti, kad sugeneruoti scenarijai pakankamai gerai atspindėtų realaus pasaulio sudėtingumą. |
| Kibernetinis saugumas | Pagerinti tinklo įsilaužimų aptikimo sistemų efektyvumą, generuojant sintetinius kenkėjiškos veiklos duomenis [33]. | Tinklo srauto metaduomenys, sistemos žurnalai, kibernetinių atakų pėdsakai. | Užtikrinimas, kad modeliai, mokyti su sintetiniais duomenimis, būtų veiksmingi bei duomenų įvairovė. |
| Mažmeninė prekyba (e-komercija) | Analizuoti pirkėjų elgseną ir testuoti dirbtinio intelekto sprendimus prekyboje neturint prieigos prie komercinių duomenų [34]. | Laiko eilučių duomenys, atspindintys klientų pirkimo elgseną, pirkimų srautus ir sąveikas su pardavimo sistemomis. | Vartotojų elgsenos imitavimas, užtikrinant statistinių dėsningumų atkartojimą be realių klientų ar verslo informacijos atskleidimo. |
| Finansai | Leisti finansų sektoriaus institucijoms modeliuoti bei analizuoti finansinius procesus, neatskleidžiant konfidencialių duomenų [35]. | Mokėjimų operacijos, klientų profiliai, finansų rinkų duomenys. | Būtina atitikti teisinius reikalavimus ir duomenų konfidencialumo apsaugą, kartu išlaikant sintetinių duomenų realistiškumą. |
| Pramonė (gamyba) | Pašalinti gamybos duomenų trūkumo problemą ir sustiprinti procesų analizę naudojant sintetinius duomenis [36]. | Gamybos įrenginių jutiklių fiksuojami duomenys bei gamybos linijų vaizdų duomenys. | Sintetinių duomenų kokybės įvertinimo ir validacijos standartų trūkumas. |

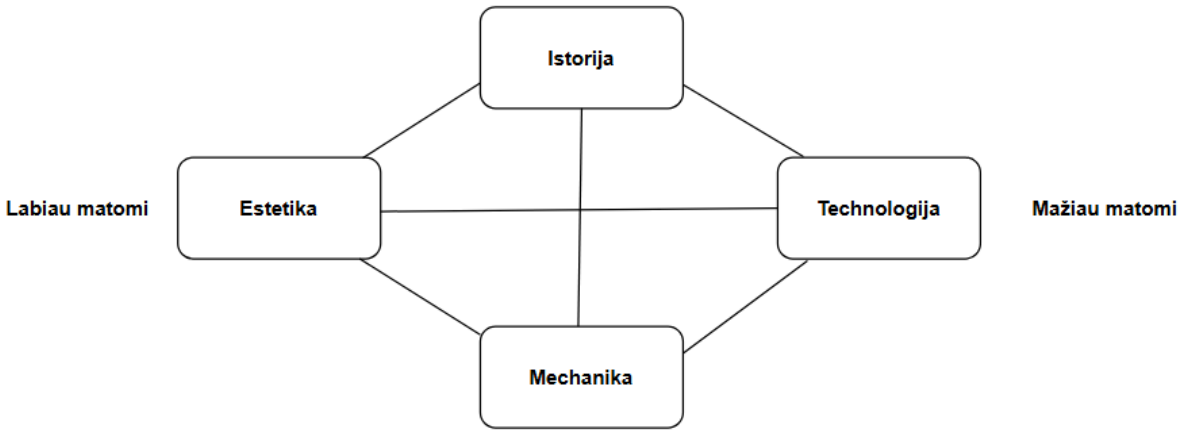
Apibendrinta taikomųjų pavyzdžių analizė parodo, kad sintetinių duomenų panaudojimas peržengė eksperimentinio testavimo etapą ir tapo praktikoje taikomų sistemų sudedamąja dalimi. Tai ypač aktualu tose srityse, kur duomenų prieinamumas yra ribotas, taikomi griežti privatumo reikalavimai ar galioja reglamentuoti naudojimo apribojimai. Dabartinė plėtra orientuota ne tik į duomenų generavimo metodų tobulinimą, bet ir į objektyvių vertinimo kriterijų sisteminimą, leidžiantį pagrįstai įvertinti modeliavimo kokybę.

## 1.4. Žaidybinimu grįstų mokymosi scenarijų kūrimas ir taikymas

Žaidybinimo (*angl. gamification*) principai mokymo procesuose taikomi dviem kryptimis. Vienu atveju mokymo veiklose integruojami žaidimo elementai, siekiant padidinti mokymosi motyvaciją ir

aktyvų įsitraukimą į mokymosi procesą, kitu atveju pati žaidimo struktūra naudojama kaip imitacinė aplinka, skirta procesų modeliavimo eksperimentams. Tokios mokymosi formos yra glaudžiai susijusios su žaidimų dizaino elementų perkėlimu į ne žaidybines situacijas, jų tikslas – sukurti sąveiką, kuri skatina giluminį mokymąsi, emocinį įsitraukimą ir aktyvų dalyvavimą mokymosi procese. Žaidybinimo samprata dažniausiai apibrėžiama kaip žaidimo elementų, pavyzdžiui, taškų, lygių, pasiekimų ar konkurencijos mechanikų integravimas į mokymosi ar darbo aplinkas [37].

Toks žaidybinimo modelis plačiai taikomas švietimo sistemose ir elektroninio mokymosi platformose. Studentai kaupia taškus, atlieka testus, atveria naujus modulius arba gauna simbolinius apdovanojimus už užduočių atlikimą. Pritaikytas žaidybinimas leidžia ne tik palaikyti motyvaciją, bet ir palaikyti aiškią mokymosi proceso struktūrą bei skatinti bendradarbiavimą. Žaidybinimo elementai taip pat taikomi verslo kontekste – darbuotojų mokymuose, kvalifikacijos kėlime ir net vadovų strateginio mąstymo lavinimui. Tačiau svarbu pabrėžti, kad paviršutiniškai pritaikytas žaidybinimas dažnai neduoda tinkamo rezultato, tam būtina turėti stiprų žinių pagrindą apie žaidimų dizaino principus ir sistemingai juos pritaikyti tikslinės auditorijos poreikiams. Vienas iš naudingų žaidybinimo analizės įrankių – žaidimo elementų modelis, išskiriantis keturias sudėtines dalis [38]:




**4 pav.** Žaidybinimo komponentų matomumo modelis

Estetiniai ir istoriniai elementai yra tiesiogiai matomi naudotojui ir padeda formuoti emocinę bei vizualinę patirtį. Tuo tarpu technologiniai bei mechaniniai komponentai dažniausiai veikia fone, apibrėždami veikimo logiką bei sąveikos taisykles.

Toliau nuo tradicinio žaidybinimo pereinama prie struktūriškai atvirkštinio proceso – žaidimų logikos taikymo realių procesų modeliavimo tikslams. Silverio ir kt. atliktame tyrime, kuriame pristatytas bendro pobūdžio skatinamojo mokymosi algoritmas *AlphaZero*, skaitmeninis analogas buvo naudojamas kaip imitacinė aplinka žaidimų strategijoms vystyti [39]. Ši aplinka leido agentui mokytis iš savarankiško žaidimo prieš save patį, nenaudojant jokių žmogaus pateiktų duomenų ar strategijų.

Pritaikius šią struktūrą verslo procesų modeliavimui, agentas gali savarankiškai generuoti ir vertinti veiksmų sekas, leidžiančias identifikuoti neakivaizdžias optimizavimo galimybes. Aplinkos taisyklės tampa tarsi žaidimo lenta, kurioje agentas sistemingai bando skirtingus veiksmus ir stebi jų pasekmes. Skirtingai nuo tradicinių žaidybinimo formų, kur orientuojamasi į žmogaus motyvacijos didinimą, čia prioritetas teikiamas modelio mokymuisi. Tokiu būdu žaidybinė imitacinė sistema tampa

mokymosi infrastruktūra, kurioje generuojami sintetiniai duomenys, naudojami verslo procesų optimizavimui. Šios dvi metodinės kryptys nėra viena kitai prieštaraujančios – priešingai, jos kaip tik papildo viena kitą ypač sudėtingose mokymosi ar sprendimų priėmimo aplinkose, kuriose derinamas žmogaus priimamų sprendimų interpretavimas ir dirbtinio intelekto metodų prisitaikymas.

## 2. Metodai ir pasiūlyta metodika

### 2.1. Markovo sprendimų procesas

Markovo sprendimų procesas apibrėžia situacijas, kai sistemos būsena kinta priklausomai nuo pasirinktų veiksmų ir atsitiktinių aplinkos veiksnių. Šis procesas plačiai taikomas skatinamojo mokymosi kontekste, nes suteikia teorinį pagrindą agento ir aplinkos sąveikai modeliuoti. Jis leidžia struktūruotai siekti maksimalaus kaupiamojo atlygio per nuoseklią veiksmų seką. Markovo sprendimų proceso pagrindą sudaro Markovo savybė, kuri nusako, jog tikimybė pereiti į būseną $s_{t+1}$ priklauso tik nuo dabartinės būsenos $s_t$ ir atlikto veiksmo $a_t$, nepriklausomai nuo ankstesnių būsenų sekos [16]:

$$P(s_{t+1} = s' | s_t = s, a_t = a) = P(s' | s, a). \tag{1}$$

Ši savybė užtikrina, kad sprendimų modelis būtų pagrįstas dabartine informacija, o ne istorinių duomenų kaupimu. Markovo sprendimų procesas apibrėžiamas kaip penkių komponentų rinkinys $(S, A, P, R, \gamma)$, kur $S$ žymi visų galimų aplinkos būsenų aibę, $A$ – veiksmų aibę, $P(s' | s, a)$ – tikimybę pereiti iš būsenos $s$ į būseną $s'$ pasirinkus veiksmą $a$, $R(s, a, s')$ – atlygį, kurį agentas gauna už tokį perėjimą, o $\gamma \in (0,1]$ – ilgalaikio atlygio svorio parametras, nurodantis būsimo atlygio svarbos sumažėjimą laike. Kiekviename žingsnyje agentas pasirenka veiksmą $a$, kuris priklausomai nuo tikimybės pasiskirstymo $P$, nulemia naują būseną $s'$ ir gautą atlygį $R$. Markovo sprendimų proceso tikslas yra pasirinkti tokią veiksmų seką, kuri leistų sukaupti kuo didesnį ilgalaikį atlygį. Šiam tikslui naudojama vertės funkcija $V(s)$, kuri nusako tikėtiną bendrą atlygį, jei agentas nuo būsenos $s$ veikia optimaliai. Optimalios politikos atveju ši vertė tenkina Belmano optimalumo lygtį [16]:

$$V^*(s) = \max_a \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma V^*(s')]. \tag{2}$$

Ši išraiška reiškia, kad agentas turi pasirinkti veiksmą $a$, kuris maksimizuotų tiesioginio atlygio $R(s, a, s')$ ir būsimos vertės $V^*(s')$ sumą, įvertintą pagal perėjimo tikimybę. Vertės funkcijoje kiekviena būsena įvertinama proporcingai tikimybei, kad ji bus pasiekta. Agentas, kuris taiko tokią strategiją, vadovaujasi optimalia politika $\pi^*$, apibrėžiančia veiksmų parinkimo taisyklę kiekvienai būsenai $s$. Ši politika apibrėžiama taip [16]:

$$\pi^*(s) = arg \max_a \sum_{s'} P(s' | s, a)[R(s, a, s') + \gamma V^*(s')]. \tag{3}$$

Kadangi sprendimų efektyvumas priklauso nuo veiksmų pasekmių, praktikoje dažnai naudojama ne tik vertės funkcija $V(s)$, bet ir Q-funkcija $Q(s, a)$, kuri įvertina kiekvieno konkretaus veiksmo naudą būsenoje $s$. Ji skaičiuojama pagal [16]:

$$Q^*(s, a) = \sum_{s'} P(s' | s, a) \left[R(s, a, s') + \gamma \max_{a'} Q^*(s', a')\right], \tag{4}$$

Q-funkcija leidžia tiesiogiai palyginti visus galimus veiksmus konkrečioje būsenoje, todėl ji itin naudinga ankstyvosiose mokymosi stadijose, kai dar nėra žinomos optimalios strategijos. Tokiu atveju taikomas Q-mokymosi metodas, kuris leidžia iteratyviai atnaujinti veiksmų vertes remiantis patirtimi, neturint išankstinės informacijos apie perėjimo tikimybes $P$ ar atlygio funkciją $R$. Po kiekvieno agento žingsnio, kai fiksuojama patirtis $(s_t, a_t, r_t, s_{t+1})$, Q-vertė atnaujinama pagal šią formulę [16]:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha)Q_t(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q_t(s_{t+1}, a')\right], \tag{5}$$

čia $\alpha$ žymi mokymosi greitį. Rekursinė taisyklė suteikia agentui galimybę artėti prie optimalios politikos neturint iš anksto žinomos perėjimų tikimybių funkcijos, nes veiksmų vertės koreguojamos remiantis tiesioginiais sąveikos su aplinka stebėjimais. Augant sąveikų kiekiui, vertinimų tikslumas didėja, o tai ilgainiui lemia strategijos priartėjimą prie optimalių veiksmų pasirinkimo.

### 2.2. Skatinamasis mokymasis

Skatinamasis mokymasis yra apibrėžiamas kaip sekos optimizavimo uždavinys, kuriame agentas sąveikauja su aplinka laiko momentais $t$. Aplinka modeliuojama kaip Markovo sprendimų procesas, tačiau šiuo atveju daroma prielaida, kad perėjimų tikimybės $P(s'|s, a)$ ir atlygio funkcija $R(s, a)$ yra nežinomos.

Agentas kiekvienu žingsniu būna būsenoje $s_t$, pasirenka veiksmą $a_t$, gauna atlygį $r_t$ ir pereina į naują būseną $s_{t+1}$. Šis sąveikos ciklas kartojasi, o agento tikslas – maksimizuoti ilgalaikį kaupiamąjį atlygį $R_t$, apibrėžtą kaupiamąja būsimo atlygio suma [16]:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{6}$$

kur $\gamma \in (0,1]$ yra ilgalaikio atlygio svorio parametras, nusakantis būsimo atlygio svarbos sumažėjimą. Vertės funkcija, žymima $V^\pi(s)$, nusako tikėtiną atlygį, kai agentas pradeda iš būsenos $s$ ir toliau laikosi politikos $\pi$ [16]:

$$V^\pi(s) = \mathbb{E}_\pi[R_t|s_t = s]. \tag{7}$$

Veiksmo vertės funkcija apibrėžiama kaip:

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a], \tag{8}$$

čia $\pi$ – veiksmų pasirinkimo strategija. Pagrindinis tikslas yra surasti optimalią politiką $\pi^*$, kuri maksimizuoja atlygį kiekvienoje būsenoje:

$$\pi^*(s) = arg \max_a Q^*(s, a), \tag{9}$$

kur optimali Q-funkcija tenkina Belmano optimalumo lygtį [16]:

$$Q^*(s,a) = \mathbb{E}\left[r_t + \gamma \max_{a'} Q^*(s',a') \mid s_t = s, a_t = a\right].$$

(10)

Kadangi pereinamosios tikimybės ir atlygio funkcijos nežinomos, o Q-funkcija yra apskaičiuojama remiantis stebimais duomenimis. Vienas paprasčiausių metodų – Q-mokymasis, kuris atnaujina vertes po kiekvieno žingsnio remiantis formule [16]:

$$Q_{t+1}(s_t,a_t) = (1-\alpha)Q_t(s_t,a_t) + \alpha\left[r_t + \gamma \max_{a'} Q_t(s_{t+1},a')\right],$$

(11)

kur mokymosi koeficientas $\alpha \in (0,1]$. Ši formulė leidžia agentui, neturint žinių apie aplinkos modelį per laiką priartėti prie optimalaus sprendimų politikos pasirinkimo.

### 2.3. Gilusis Q-mokymasis

Gilusis Q-mokymasis (DQN) yra modeliu nepagrįstas skatinamojo mokymosi metodas, kuriame Q-funkcija apskaičiuojama naudojant neuroninį tinklą. Šis metodas leidžia taikyti Q-mokymosi principus situacijose, kuriose veiksmų ir būsenų erdvės yra per didelės, kad būtų galima naudoti tradicinę Q-reikšmių matricą. Standartinis DQN metodas naudoja parametrizuotą Q-funkciją [16]:

$$Q(s,a;\theta) \approx Q^*(s,a),$$

(12)

čia $\theta$ – neuroninio tinklo parametrai, mokomi apskaičiuoti optimalią veiksmo vertės funkciją. Tikslo funkcija DQN atveju yra vidutinis kvadratinis nuokrypis tarp tikslo reikšmės ir prognozuotos reikšmės [40]:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\left[\left(y - Q(s,a;\theta)\right)^2\right],$$

(13)

kur tikslo reikšmė:

$$y = r + \gamma \max_{a'} Q(s',a';\theta^-),$$

(14)

$\theta^-$ – tai tikslinės Q-funkcijos tinklo parametrai, kurie atnaujinami rečiau nei pagrindinio tinklo $\theta$, siekiant stabilizuoti mokymosi procesą. Toks papildomas tinklas vadinama tiksliniu tinklu. Kita svarbi DQN savybė – patirties atmintis (*angl. experience replay*), kurioje kaupiami agento veiksmai, būsenos, atlygiai ir pereinamosios būsenos. Mokymosi proceso metu patirtys atsitiktinai atrenkamos iš patirties atminties, siekiant sumažinti duomenų koreliaciją tarp nuoseklių stebėjimų. Tai padeda pagerinti modelio mokymosi proceso stabilumą. DQN metodo veikimo principas apibendrinamas taip [40]:

- Inicijuojamas neuroninis tinklas su atsitiktiniais parametrais $\theta$.
- Kiekviename žingsnyje agentas stebi būseną $s$ ir pasirenka veiksmą $a$.
- Atlikęs veiksmą, agentas gauna atlygį $r$ ir pereina į naują būseną $s'$.
- $(s,a,r,s')$ įrašomas į patirties atmintį.

– Atsitiktinai parenkama agento būsenos, veiksmo, atlygio ir pereinamosios būsenos kombinacija ir apskaičiuojamas nuostolis $L(\theta)$.
– Atliekamas atgalinio sklidimo (angl. backpropagation) žingsnis ir atnaujinami tinklo parametrai.
– Kas nustatytą žingsnių skaičių atnaujinami tikslinio tinklo $\theta^-$ parametrai.

### 2.4. Proksimalinis politikos optimizavimas

Proksimalinis politikos optimizavimas (PPO) yra strategija grįstų skatinamojo mokymosi metodų grupės metodas. Jo tikslas – tiesiogiai optimizuoti politiką $\pi_\theta(a|s)$, maksimizuojant tikėtiną sukauptą atlygį. Terminas proksimalus (*lot. proximus – artimiausias*) reiškia, kad naujai mokoma politika neturi pernelyg skirtis nuo ankstesnės, nes per dideli pokyčiai gali sukelti mokymosi nestabilumą. PPO metodas šią sąlygą realizuoja ne per griežtus apribojimus, o per specialiai sukonstruotą funkciją su apribojimu (*angl. clipping*). Tai padeda stabilizuoti mokymosi procesą ir apsaugo nuo per didelių strategijos nuokrypių [41].

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t], \tag{15}$$

kur $r_t$ yra politikos tikimybių santykis tarp naujosios $\pi_\theta$ ir senosios $\pi_\theta{}'$ politikos, $A_t$ yra pranašumo įvertis, o $\epsilon$ parametras, kuris riboja leidžiamą politikos pokytį vieno atnaujinimo metu [42].

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\prime}(a_t|s_t)}, \tag{16}$$

$clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ funkcija riboja $r_t$ reikšmę intervale $[1-\epsilon, 1+\epsilon]$, taip neleidžiant pernelyg dideliam politikos atnaujinimui. Jei pranašumas $A_t$ yra teigiamas, didesnė $r_t$ reikšmė reikštų naudą, tačiau tokia nauda būtų apribojama iki $1+\epsilon$. Analogiškai, jei $A_t$ yra neigiama, politika negalės per daug sumažinti veiksmo tikimybės, nes $r_t$ bus apribotas iki $1-\epsilon$.

### 2.5. Sintetinių duomenų generavimas

Skatinamojo mokymosi aplinkoje sintetiniai duomenys generuojami agentui sąveikaujant su imituojama aplinka. Tokios sąveikos metu sukuriami duomenų rinkiniai, kuriuose kiekvienas žingsnis laikomas atskiru duomenų tašku, o visa proceso imitavimo eiga – epizodu. Epizodai formuojami agento veiksmais, kuriuos jis pasirenka remdamasis savo politika bei grįžtamuoju ryšiu. Toks sintetinių duomenų generavimas leidžia kurti neribotą duomenų kiekį nepažeidžiant privatumo, saugumo ar prieigos problemų. Kiekvieną epizodą galima apibrėžti kaip baigtinę seką [16]:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_T). \tag{17}$$

$T$ – epizodo pabaigos momentas. Ši seka sugeneruojama naudojant politiką $\pi(a|s)$, kuri nusako veiksmų paskirstymą, atsižvelgiant į esamą būseną.

Kiekvienas epizodas $\tau$ yra laikomas nepriklausomu sintetinės patirties atveju, kuris naudojamas modelio mokymui arba politikos formavimui. Sukauptų epizodų rinkinys sudaro patirties atmintį $D = \{\tau_1, \tau_2, \dots, \tau_n\}$, kuri leidžia vykdyti nuoseklų mokymosi procesą tiek realiuoju laiku, tiek pasitelkiant

atsitiktinai pasirinktus ankstesnius patirties pavyzdžius. Tokia praktika mažina duomenų tarpusavio priklausomybę ir prisideda prie stabilesnio mokymosi proceso.

## 2.6. Skatinamo mokymosi bibliotekos pasirinkimas

Skatinamojo mokymosi tyrimuose bibliotekos pasirinkimas yra metodologiškai reikšmingas sprendimas, turintis tiesioginę įtaką metodų realizacijos kokybei ir rezultatų interpretavimo pagrįstumui. Vertinant skatinamojo mokymosi bibliotekas, būtina atsižvelgti į palaikomų metodų įvairovę, suderinamumą su pasirinkta aplinka, plėtros galimybes bei dokumentacijos išsamumą. Atsižvelgiant į šiuos kriterijus, atlikta kelių skatinamojo mokymosi bibliotekų analizė, įvertinant jų tinkamumą verslo procesų žaidybinimo ir imitacinio modeliavimo kontekstui.

- *Stable-Baselines3* yra atvirojo kodo biblioteka, sukurta naudojant *PyTorch* karkasą, apimanti pagrindinius giluminio skatinamojo mokymosi metodus. Ši biblioteka išsiskiria nuoseklia architektūra ir suderintomis programinėmis sąsajomis, kurios sudaro sąlygas keisti metodų parametrus ir pritaikyti juos įvairioms imitacinėms aplinkoms. Vienas iš pagrindinių šios bibliotekos privalumų – paprastas naudojimas ir aiški dokumentacija. Šioje bibliotekoje integruotas *TensorBoard* įrankis, kuris leidžia vizualizuoti mokymosi dinamiką. *Stable-Baselines3* biblioteka sukurta mokyti vieną agentą, todėl paskirstyto mokymo bei sudėtingų kelių agentų sąveiką apimančių sistemų palaikymas šios architektūros rėmuose nėra numatytas [43].
- *RLib* – tai atvirojo kodo skatinamojo mokymosi biblioteka, sukurta *Ray* pagrindu, pritaikyta vykdyti mokymo procesus su lygiagretaus skaičiavimo palaikymu. Ji suderinta su *PyTorch* ir *TensorFlow* sistemomis ir leidžia atlikti mokymąsi naudojant klasterius [44]. Bibliotekos architektūra pagrįsta aktorių modeliu, kuriame atskiri mokymosi uždaviniai vykdomi lygiagrečiai, užtikrinant jų sąveiką per bendrą valdymo struktūrą.
- *KerasRL* – skatinamojo mokymosi metodų biblioteka, sukurta naudojant *Keras* neuroninių tinklų struktūrą. Biblioteka palaiko klasikinius skatinamojo mokymosi metodus ir yra pritaikyta mažo sudėtingumo, diskrečių veiksmų erdvėmis pasižymintiems uždaviniams spręsti. Ši biblioteka pasižymi lankstumu ir galimybe kontroliuoti agento veikimo logiką. Vis dėlto, ji nėra pritaikyta lygiagretiems skaičiavimams ir kelių agentų sąveikoms modeliuoti. Ribotas techninis palaikymas bei neišsami dokumentacija lemia, kad ši priemonė tinkamesnė tyrimams, orientuotiems į metodų veikimą ir rankinį parametrų valdymą, o ne į sudėtingų sistemų diegimą.

Atsižvelgiant į minėtų bibliotekų savybes, tolimesniam tyrimui pasirinkta naudoti *Stable-Baselines3*. Ši biblioteka užtikrina reikiamų metodų įvairovę bei suderinamumą su individualiai sukurta verslo procesų imitacine aplinka. Nors *RLib* pasižymi išplėstinėmis skaičiavimo resursų paskirstymo galimybėmis ir pritaikomumu sudėtingoms sistemoms, jos architektūrinis sudėtingumas šio darbo kontekste buvo laikytas pertekliniu. *KerasRL*, nors ir pasižymi tam tikru lankstumu, ši biblioteka nėra tinkama naudoti dėl nepakankamos dokumentacijos ir riboto pritaikomumo sudėtingesnių architektūrų ar ilgesnio mokymosi trukmės scenarijuose. Atsižvelgiant į skatinamojo mokymosi bibliotekos funkcionalumą, realizuojamų metodų įvairovę, naudojimo paprastumą ir suderinamumą su pasirinkta aplinka, *Stable-Baselines3* pasirinkta kaip tinkamiausia alternatyva, nes leidžia efektyviai vykdyti verslo proceso imitacinį modeliavimą ir susitelkti į tyrimo tikslus.

### Stable-Baselines3 architektūros struktūra ir veikimo logika

Skirtingai nei kai kurios kitos skatinamojo mokymosi bibliotekos, *Stable-Baselines3* pasižymi nuoseklia vidine struktūra ir lengvai pritaikoma standartizuota programine sąsaja, kuri leidžia efektyviai taikyti bei testuoti skirtingas mokymosi strategijas toje pačioje aplinkoje. Dėl savo aiškios architektūros, *Stable-Baselines3* yra tinkama naudoti moksliniuose tyrimuose, kuriuose būtina užtikrinti metodų struktūrinį aiškumą ir suderinamumą su imitacine aplinka. Skatinamojo mokymosi metodų taikymo ribas dažnai apibrėžia jų suderinamumas su konkrečiais aplinkos duomenų tipais. Priklausomai nuo to, ar veikiama diskrečioje, tolydžioje ar mišrioje veiksmų erdvėje, skirtingi metodai pasižymi nevienodu pritaikomumu. Siekiant aiškumo, žemiau pateikiamas metodų ir jų suderinamų veiksmų bei būsenų erdvių palyginimas.

**2 lentelė.** Stable-Baselines3 skatinamojo mokymosi metodų savybių palyginimas

| Metodas | Box | Discrete | MultiDiscrete | MultiBinary | Multi Processing |
|---|---|---|---|---|---|
| ARS | + | + | - | - | + |
| A2C | + | + | + | + | + |
| DDPG | + | - | - | - | + |
| DQN | - | + | - | - | + |
| HER | + | + | - | - | + |
| PPO | + | + | + | + | + |
| QR-DQN | - | + | - | - | + |
| SAC | + | - | - | - | + |
| TD3 | + | - | - | - | + |

Bibliotekos architektūra paremta objektinio programavimo principais. Visi metodai paveldi bendrąją klasę BaseAlgorithm, kurią sudaro funkcijų rinkinys [43]:

- *Learn()* – vykdo modelio mokymą, kol pasiekiamas nurodytas žingsnių skaičius.
- *Predict()* – pateikus tam tikrą aplinkos būseną, modelis apskaičiuoja numatomą veiksmą.
- *Save()* – įrašo treniruotą modelį į pasirinktą katalogą ar failą.
- *Load()* – įkelia anksčiau išsaugotą modelį iš disko arba duomenų struktūros. Jei modelis bus taikomas tik veiksmų prognozei, aplinkos objektas nėra būtinas, tačiau tęsiant mokymą, aplinkos nurodymas yra privalomas, kad būtų išlaikyta suderinama sąveikos struktūra.

Kiekvienas metodas *Stable-Baselines3* bibliotekoje veikia su atskiru politikos moduliu – tai neuroninio tinklo architektūra, naudojama veiksmų generavimui remiantis aplinkos būsena. Standartiniai politikos šablonai apima:

- *MlpPolicy* – tai daugiasluoksnio perceptrono pagrindu sukurta politika, naudojama tada, kai agento stebėjimai pateikiami skaitinės vektorinės formos. Architektūra susideda iš kelių pilnai sujungtų neuroninių sluoksnių ir yra tinkama klasikinėms skatinamojo mokymosi aplinkoms, kur stebėjimai pateikiami kaip skaitiniai vektoriai.
- *CnnPolicy* – ši politika sukurta naudoti su vaizdinėmis įvestimis. Naudojami konvoliuciniai sluoksniai, kurie automatiškai išgauna reikšmingus bruožus iš vaizdo duomenų.
- *MultiInputPolicy* – taikoma tada, kai agento įvestis susideda iš kelių skirtingo tipo duomenų.

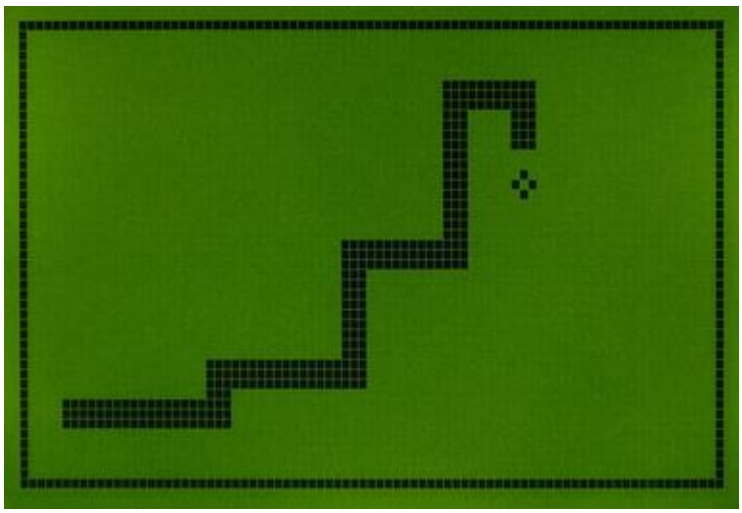
Kadangi šiame darbe modeliuojama verslo proceso imitacinė aplinka yra pagrįsta struktūrizuota, vektorinės formos stebėjimų erdve, kur agentas gauna konkrečias skaitines reikšmes apie savo padėtį

ar aplinkos būsenas, pasirinkta naudoti *MlpPolicy*. Ji tinkamiausia PPO ar DQN metodams, kai aplinka nepateikia vaizdinių duomenų reprezentacijos.

Mokymosi eiga *Stable-Baselines3* aplinkoje yra automatizuota. Agentas inicijuojamas nurodant aplinką, pasirinktą metodą ir politikos tipą. Bibliotekos vidinis mechanizmas valdo visą sąveiką su aplinka, patirties kaupimą ir tinklų svorių atnaujinimą. Ši biblioteka taip pat pasižymi aukštu testavimo padengimu apie 95 % automatizuotais testais [43]. Modelių svoriai ir parametrai išsaugomi, tai leidžia perkelti agentus tarp eksperimentinių aplinkų ar tęsti mokymą vėlesniuose etapuose. *TensorBoard* integracija bibliotekoje atlieka sisteminę duomenų registravimo ir analizės funkciją. Mokymosi metu generuojami kiekybiniai rodikliai automatiškai įrašomi į specialius binarinius duomenų rinkinius. Tokia architektūra leidžia realiuoju laiku stebėti mokymo eigą, analizuoti mokymosi stabilumą ir vertinti agento pažangą remiantis empiriniais duomenimis. Vizualizacija padeda ne tik sekti metrikų dinamiką, bet ir koreguoti hiperparametrus ir identifikuoti galimas mokymosi problemas. Dėl aiškios architektūros, standartizuotos API ir pakankamos metodų įvairovės ši biblioteka suteikia galimybę eksperimentuoti, keisti agento elgsenos parametrus ir objektyviai vertinti skirtingų strategijų efektyvumą, užtikrinant moksliškai pagrįstą patikimumą ir praktinį pritaikomumą.

## 2.7. Gyvatėles žaidimo aplinka

Gyvatėlės žaidimas (*angl. Snake*) yra klasikinė diskrečios dinamikos sistema. Šis žaidimas atsirado XX a. aštuntajame dešimtmetyje ir tapo itin populiarus mobiliuosiuose įrenginiuose.



**5 pav**. Gyvatėlės žaidimo aplinka Nokia įrenginyje [44]

Pagrindinis šio žaidimo tikslas – valdyti nuolat judančią gyvatę, kurios tikslas surinkti aplinkoje atsirandančius maisto objektus. Kiekvieną kartą surinkus maistą, gyvatėlę pailgėja vienu segmentu, tokiu būdų žaidimo sudėtingumas išauga, kadangi sumažėja aplinkoje likusi erdvė gyvatėlės judesiams atlikti. Gyvatėlės galvos segmentas, kuris tarsi skatinamojo mokymosi agentas laiko momente $t$, pasirenka vieną iš keturių veiksmų: judėjimą aukštyn, žemyn, į kairę arba į dešinę. Gyvatė juda nenutrūkstamai, todėl strateginis planavimas tampa būtinybe, augant gyvatės ilgiui.

Žaidimo erdvė modeliuojama kaip baigtinio dydžio dvimatis tinklas $G \in \mathbb{Z}^{M \times N}$, kur kiekvienas tinklo elementas gali būti tuščias $g_{i,j} = 0$, užimtas gyvatės segmentu $g_{i,j} = 1$, arba žymėti maisto segmentą $g_{i,j} = 2$. Gyvatė sąveikauja su aplinka vykdydama diskrečius veiksmus, o būsena $s_t$ kinta priklausomai nuo ankstesnių veiksmų sekos. Atlygio funkcija $r(s_t, a_t)$ apibrėžiama:

$$R_t = \sum_{k=0}^{K} \gamma^k r_{t+k}.$$

(18)

Dėl savo aiškios struktūros, ribotų taisyklių ir galimybės tiksliai apibrėžti būsenas bei veiksmus, ši aplinka yra plačiai naudojama skatinamojo mokymosi eksperimentuose. Ji leidžia analizuoti sprendimų sekas, kuriose būtina planuoti veiksmus atsižvelgiant į ilgalaikę naudą bei grėsmes, kylančias dėl augančios gyvatės struktūros.

### 2.8. Aukštos vertės klientų aptarnavimo imitacinė aplinka

Šiame tyrimo etape apibrėžiama skatinamojo mokymosi imitacinė aplinka, sukurta remiantis klasikinio „Gyvatėlės“ žaidimo principais ir išplečiant verslo procesų imitavimo su aukštos vertės klientais (VIP) logiką. Skatinamojo mokymosi agentas veikia kaip įmonė, kurios tikslas – efektyviai aptarnauti klientus ir maksimizuoti bendrą atlygį. Aplinkos dinamika modeliuojama taip, kad atspindėtų tiek įprastų, tiek aukštos vertės klientų aptarnavimo poreikius ir išteklių paskirstymo ribojimus.

Aukštos vertės klientų valdymo imitacinė aplinka apibrėžiama kaip Markovo sprendimų procesas, kur aplinkos būsena $s_t \in S$ kiekvienu laiko momentu, aprašoma kaip keturiolikos dimensijų vektorius. Būsenos vektorius saugo informaciją apie normalizuotus atstumus iki įprastų ir aukštos vertės klientų, keturis dvejetainius indikatorius, žyminčius, ar artimiausia pozicija kiekviena galima kryptimi yra užblokuota kliūtimi ar įmonės segmentu, keturis dvejetainius indikatorius, nurodančius paskutinę įmonės judėjimo kryptį, santykį tarp įmonės ir aplinkos dydžio ir aukštos vertės kliento aktyvumo indikatorių, rodantį, ar aukštos vertės klientas egzistuoja aplinkoje laiko momentu $t$.

Veiksmų aibė $a_t \in \{0, 1, 2, 3\}$ sudaryta iš keturių krypčių, kurias įmonė gali pasirinkti, kad pasiekti klientą. Veiksmų atlikimas yra aprašytas logine taisykle – veiksmai, kurie yra priešingi paskutinei pasirinktai judėjimo krypčiai, yra draudžiama siekiant išvengti neleistinų apsisukimų.

Bendrojo atlygio funkcija $r_t$ sudaryta iš 5 dedamųjų, įmonė gauna atlygį už klientų aptarnavimą ir efektyvų judėjimą link kliento, taip pat patiria baudas už neefektyvumą ar klaidas. Bazinis atlygis už įprasto kliento aptarnavimą:

$$r_{klientas} = 10,$$

(19)

tačiau papildomai taikomas papildomas bonusas už efektyvų kliento pasiekimą, proporcingas aptarnavimo greičiui:

$$r_{efektyvus\ judėjimas} = max\left(0, 1 - \frac{\Delta t}{100}\right) \cdot 5,$$

(20)

kur $\Delta t$ žymi žingsnių skaičių nuo paskutinio aptarnauto kliento. Papildomai skaičiuojama augimo premija, priklausanti nuo įmonės ilgio ir aplinkos dydžio:

$$r_{augimo\ premija} = \frac{įmonės_{ilgis}}{aplinkos_{ilgis} + aplinkos_{plotis}}.$$

(21)

Aptarnavus aukštos vertės klientą, atlygis yra ženkliai didesnis ir apibrėžiamas:

$$r_{vip} = 100 + \max\left(0{,}1 - \frac{t_{vip}}{30}\right) \cdot 50,$$

(22)

kur $t_{vip}$ – žingsnių skaičius nuo aukštos vertės kliento pasirodymo aplinkoje. Aukštos vertės klientai pasirodo aptarnavus tris įprasto tipo klientus, o jų aptarnavimas yra apribotas 30 žingsnių. Jei šios kategorijos klientas neaptarnaujamas laiko, laikoma, kad svarbus sandoris prarastas, ir įmonei skiriama bauda $r_t = -10$. Įmonė gauna atlygį už kiekvieną priartėjimą prie kliento. Net ir tais atvejais, kai klientas nėra pasiekiamas laiku, įmonė yra skatinama judėti efektyviai – už kiekvieną žingsnį, priartinantį prie kliento pagal Manheteno atstumą, suteikiamas papildomas atlygis:

$$r_{artėjimo\ nauda} = 0{,}5 \cdot \Delta d,$$

(23)

kur $\Delta d$ žymi atstumo pokytį tarp įmonės senos ir naujos pozicijos. Vis dėlto, norint apriboti bereikalingą judėjimą, kiekvienas žingsnis turi papildomą baudos įvertį:

$$r_{neefektyvus\ žingsnis} = -0.02.$$

(24)

Jeigu įmonė susiduria su aplinkos ribomis arba savo segmentu, epizodas nutraukiamas, o atlygis:

$$r_t = -10.$$

(25)

Įmonės tikslas išmokti politiką $\pi(s)$, kuri maksimizuoja kaupiamąją naudą epizodo metu:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

(26)

Optimali politika apibrėžiama:

$$\pi^*(s) = arg\max_a Q^*(s,a),$$

(27)

kur $Q^*(s,a)$ žymi optimalią veiksmo vertės funkciją, tenkinančią Belmano lygtį. Tokiu būdu įmonės mokymosi procesas aprašomas kaip optimizavimo uždavinys, kurio sprendimo metu įmonė išmoksta strategijų, leidžiančių efektyviausiai aptarnauti klientus, valdyti išteklius ir reaguoti į prioritetinius poreikius kintančioje aplinkoje.

## 2.9. Verslo procesų imitavimas esant rinkos kliūtims ir rizikingiems užsakymams

Antroji tiriamoji aplinka grindžiama ta pačia verslo imitavimo logika, tačiau papildant aplinka rizikingų užsakymų ir rinkos kliūčių elementais. Būsenos vektorius padidinimas iki 18 dimensijų, įtraukiant papildomus komponentus, žyminčius artimiausio rizikingo užsakymo ir artimiausios rinkos kliūties padėties atstumo skirtumą nuo įmonės segmento atsakingo už judėjimo krypties pasirinkimą. Tokiu būdu įmonei suteikiama papildoma informacija apie aplinkoje esančius rizikos veiksnius, leidžianti priimti strateginius sprendimus dėl judėjimo krypties, siekiant minimizuoti galimus nuostolius.

Rizikingų užsakymų aptarnavimas sukelia tiesioginį nuostolį, tai imituoja klaidingus arba nenaudingus verslo sprendimus, kurių įmonė turėtų vengti. Atlikus tokio užsakymo aptarnavimą, taikomas atlygis:

$$r_{rizikingas\ užsakymas} = -10. \tag{28}$$

Taip pat į aplinką integruotos dinaminės rinkos kliūtys, kurios atsiranda tuomet, kai aukštos vertės klientai nėra aptarnaujami per nustatytą 30 žingsnių limitą. Tai atitinka rinkos pokyčius ar konkurencijos augimą, kurie atsiranda dėl nesėkmingo strateginio sprendimo. Rinkos kliūtys turi ribotą gyvavimo trukmę, tačiau įmonei susidūrus su šia kliūtimi epizodas nutraukiamas ir skiriama bauda:

$$r_{rinkos\ kliūtis} = -10. \tag{29}$$

Tokiu būdu įmonė mokosi ne tik aptarnauti klientus bei reaguoti į aukštos vertės klientų užsakymus, bet ir įvertinti veiksmų pasekmes, vengti nuostolingų sprendimų ir prisitaikyti prie netikėtų aplinkos pokyčių.

## 2.10. DQN metodo taikymas skatinamojo mokymosi imitacinėse verslo aplinkose

Pirmojoje imitacinėje aplinkoje agento mokymuisi buvo taikytas gilusis Q-mokymasis (DQN), kuris leidžia apytiksliai įvertinti veiksmo vertės funkciją $Q(s, a)$, naudojant dirbtinį neuroninį tinklą, kuris išmoksta prognozuoti būsimos naudos reikšmes kiekvienai veiksmų alternatyvai esamoje būsenoje.

Metodas buvo realizuotas naudojant *Stable-Baselines3* biblioteką. Naudojant DQN metodą kartu su aukštos vertės klientų valdymo imitacine aplinka, buvo pasirinkti tokie parametrai:

**3 lentelė.** DQN metodo parametrai

| Parametras | Reikšmė | Aprašymas |
|---|---|---|
| learning_rate | 0.0001 | Mokymosi greitis $\alpha$, lemiantis Q-verčių atnaujinimo greitį. |
| buffer_size | 50000 | Patirties atmintis nurodo kiek ankstesnių veiksmų ir būsenų saugoma mokymosi metu. |
| learning_starts | 100 | Žingsnių skaičius prieš pradedant mokymo etapą, šie duomenys neįtraukiami į patirties atmintį. |
| batch_size | 32 | Patirties atminties elementų skaičius, naudojamas vienam mokymosi ciklui. |
| gamma | 0.99 | Nuolaidos koeficientas $\gamma$, rodo agento orientavimą į ilgalaikę naudą. |
| train_freq | 4 | Modelio atnaujinimo dažnis |
| target_update_interval | 1000 | Q-verčių atnaujinimo intervalas, taikomas stabilumo užtikrinimui. |
| exploration_fraction | 0.4 | Per pirmuosius 40 % epizodų mažinama tyrinėjimo tikimybė. |
| exploration_final_eps | 0.05 | Mažiausia tyrinėjimo tikimybė |

Modelio rezultatai buvo stebimi naudojant *Callback* funkciją, kuri registravo epizodinius duomenis į *csv* formato duomenų dokumentą ir *Tensorboard* įrankį. Epizodai buvo apibrėžti kaip mokymosi sekos, trunkančios iki 100 žingsnių be aptarnauto kliento arba bendrai epizodo trukmei pasiekus 400 žingsnių. Kiekvieno epizodo pabaigoje fiksuoti tokie rodikliai kaip bendras atlygis, aptarnautų klientų skaičius, sukauptas atlygis už efektyvų judėjimą ir baudos už neefektyvų judėjimą.

Verslo procesų imitavimo aplinkai esant rinkos kliūtims ir rizikingiems užsakymams, buvo sumažinta tyrinėjimo trukmė iki 10 % visos mokymosi trukmės, o galutinė tyrinėjimo reikšmė sumažinta iki 0.01. Toks sprendimas buvo priimtas remiantis nauja aplinkos struktūra – rizikingi užsakymai, rinkos kliūtys, pasireiškia kaip neigiami veiksniai. Mažesnis tyrinėjimas buvo taikomas siekiant išvengti pernelyg dažno atsitiktinio elgesio. Kartu su anksčiau fiksuotais epizodiniais rodikliais, papildomai buvo įtraukta sekti rizikingų užsakymų, susidūrimo su kliūtimis, atsiradusių kliūčių ir išnykusių kliūčių duomenis.

## 2.11. PPO metodo taikymas skatinamojo mokymosi imitacinėse verslo aplinkose

PPO metodas imitacinėse aplinkose buvo taikytas tokiu pačiu principu kaip ir DQN metodas. Kadangi epizodų struktūra, atlygio komponentai ir aplinka išliko nepakitę, esminiai metodologiniai skirtumai tarp PPO ir DQN metodų yra tik mokymosi architektūroje ir taikytų parametrų struktūroje. PPO metodas abejuose imitacinėse aplinkose buvo naudotas su vienodais modelio parametrais:

**4 lentelė.** PPO metodo parametrai

| Parametras | Reikšmė | Aprašymas |
|---|---|---|
| n_steps | 2048 | Veiksmų sekos ilgis, surenkamas prieš kiekvieną politikos atnaujinimą. |
| batch_size | 64 | Nurodo kiek veiksmų įtraukiama į vieną ciklą. Daroma atsitiktinė atranka iš n_steps. |
| n_epochs | 10 | Politikos optimizavimo pakartojimų skaičius. |
| gamma | 0.99 | Nuolaidos koeficientas $\gamma$, rodo agento orientavimą į ilgalaikę naudą. |
| gae_lambda | 0.95 | Pranašumo įvertinimo slopinimo koeficientas. Kuo arčiau 1, tuo labiau įvertinamas tolimesnis atlygis |
| ent_coef | 0.01 | Tyrinėjimo tikimybė |

Modelio treniravimas vykdytas iteratyviai, kas 10 tūkstančių žingsnių. Kiekvieno epizodo pabaigoje fiksuoti tokie patys rodikliai kaip ir DQN metode, abiejų imitacinių aplinkų kontekste. Abu skatinamojo mokymosi metodai veikia trijuose epizodų intervaluose. Mokymo etapą sudaro pirmi 20000 epizodai, tyrinėjimo etapą sudaro epizodai nuo 20000 iki 60000 epizodai ir taikymo etapą sudaro paskutiniai 40000 epizodai.

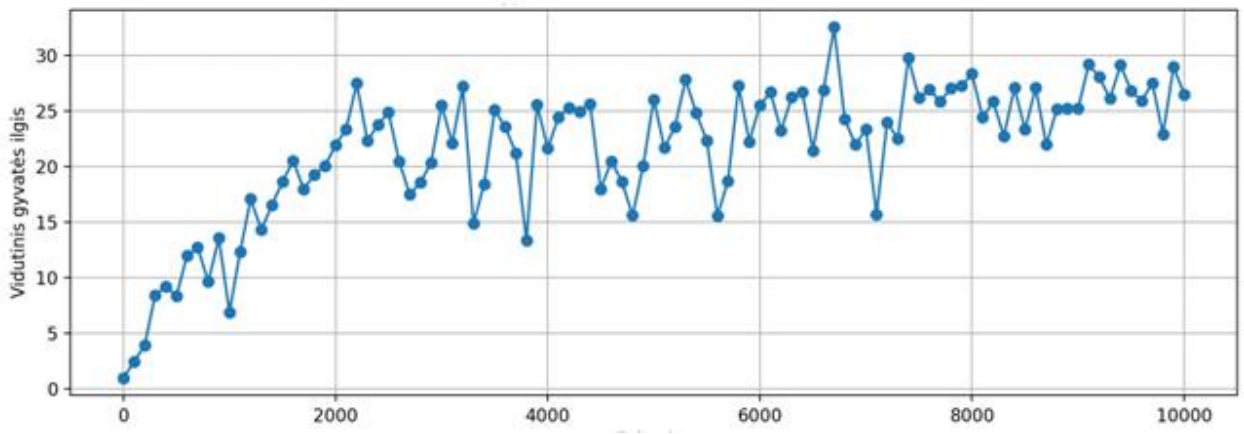
## 3. Tyrimo rezultatų analizė

### 3.1. Q-mokymasis gyvatėlės žaidimo aplinkoje

Šiame etape įgyvendinamas klasikinio žaidimo modeliavimas, kuriame agentas mokosi priimti sprendimus naudodamas Q-mokymosi metodą. Sukurta dvimatė aplinka imituoja žaidimo logiką. Agentas gali judėti viena iš keturių krypčių, o kiekvienas veiksmas lemia žingsnį aplinkoje.

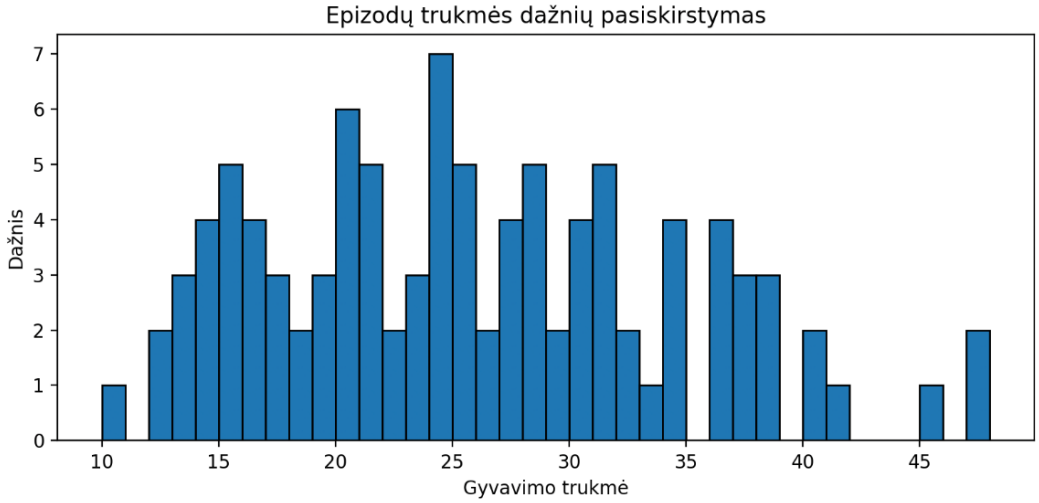

**6 pav.** Gyvatėlės žaidimo aplinkos pavyzdys

Maisto padėtis generuojama atsitiktinai, užtikrinant, kad objektas neatsirastų ten, kur yra gyvatės segmentai. Kiekvienas susidūrimas su sienomis ar gyvatėlės kūno segmentu užbaigia epizodą, o maisto surinkimas lemia gyvatėlės pailgėjimą. Siekiant išmokyti agentą optimalios strategijos, buvo suformuota atlygio funkcija, kurioje teigiamas atlygis skiriamas už maisto surinkimą, o neigiamas už susidūrimą su kliūtimis. Mokymas truko 10 tūkstančių epizodų. Siekiant įvertinti modelio mokymosi pažangą, buvo atlikta rezultatų analizė:




**7 pav.** Vidutinio gyvatės ilgio kitimas mokymo metu

Pateiktame grafike vaizduojama agento surinktų maisto vienetų dinamika, apskaičiuojant vidurkį kas 100 mokymosi epizodų. Rezultatai rodo aiškią tendenciją pirmaisiais 3000 epizodų, per kuriuos agentas palaipsniui formuoja efektyvią elgseną. Aukštesnės vidutinio ilgio reikšmės vėlesniuose epizoduose rodo pasiektą politikos stabilumą ir priartėjimą prie optimalaus sprendimų priėmimo. Nepaisant epizodinių svyravimų, bendras našumo augimas rodo nuoseklų mokymosi progresą.




**8 pav.** Epizodų trukmės dažnių pasiskirstymas

Epizodų trukmės dažnių histograma rodo, kad dauguma epizodų trunka nuo 20 iki 35 žingsnių. Toks pasiskirstymas leidžia daryti prielaidą dėl iš dalies susiformavusios strategijos, kurioje vis dar išlieka tyrinėjimo elgsenos požymių. Epizodų trukmės dispersija rodo, kad mokymosi procesas dar nėra visiškai stabilus, o strategijos tobulinimas galėtų būti siejamas su tolimesniu tyrinėjimo kontrolės politika.

Atsižvelgiant į tai, tolesniame etape numatomas šio žaidimo aplinkos integravimas į verslo procesų imitacinio modeliavimo struktūrą, taikant skatinamojo mokymosi metodus.

### 3.2. Verslo procesų imitacinis modeliavimas

Siekiant modeliuoti verslo sprendimų priėmimo procesą dinamiškoje ir neapibrėžtoje aplinkoje, šiame tyrime klasikinė žaidimo aplinka perstruktūruojama į verslo procesų imitacinį modelį. Pritaikius imitacinio modeliavimo principus, žaidimas interpretuojamas kaip verslo veikimo procesas. Skatinamojo mokymosi agentas veikia kaip autonominė įmonė, kuriai reikia efektyviai reaguoti į besikeičiančią aplinką – pasiekti klientus, išvengti nuostolingų sprendimų ir valdyti turimus resursus.

Kiekvienas klasikinio žaidimo komponentas turi aiškų verslo analogą. Toliau pateikiama lentelė, iliustruojanti, kaip šie žaidimo elementai interpretuojami trijuose skirtinguose verslo kontekstuose:

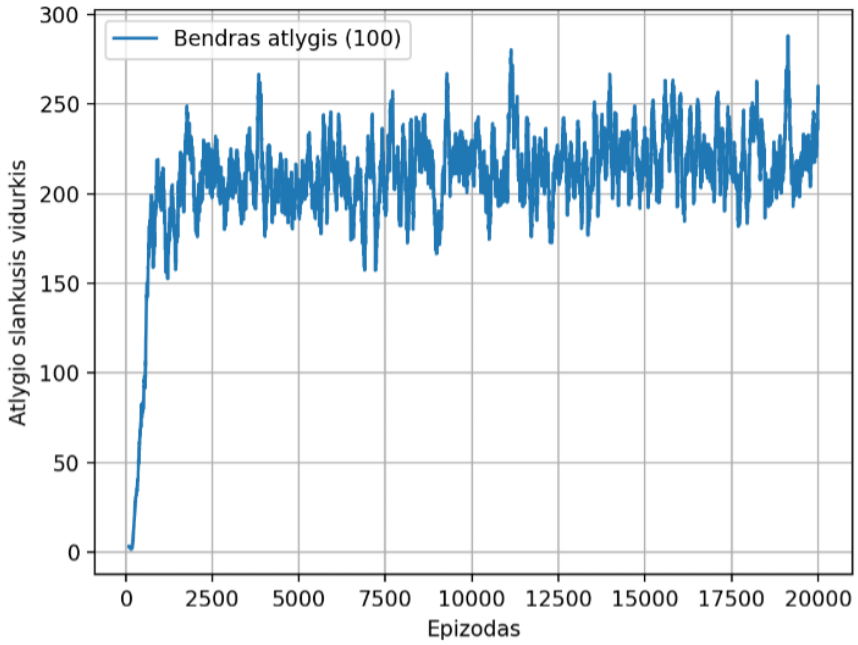
**5 lentelė.** Verslo logikos integracija į imitacinio modeliavimo aplinką

| **Gyvatėlės žaidimo elementas** | **NT pastato užpildymas** | **Startuolio vartotojų bazės plėtra** | **Skolų išieškojimas** |
|---|---|---|---|
| Gyvatės judėjimas | NT agento aktyvi klientų paieška, derybos, sutarčių pasirašymas | Aktyvi marketingo, reklamos, partnerystės iniciatyvų vykdymas | Aktyvus skolininkų kontaktavimas (skambučiai, laiškai, susitikimai) |
| Atsirandančios kliūtys | Rinkos pokyčiai, konkurencija, pandemijos poveikis | Rinkos pokyčiai, konkurentų atakos, reklamos efektyvumo sumažėjimas | Klientų bankrotai, bylinėjimasis, teisiniai ribojimai |
| Atsitrenkimas į save/sieną | Netinkami sprendimai, rizikingi klientai, prarasti sandoriai | Vartotojų pasitikėjimo praradimas, augimo be infrastruktūros pavojai | Netinkamas išieškojimas, reputacijos žala, teisinių reikalavimų pažeidimai |
| Maisto suvalgymas | Sėkmingas nuomininko pritraukimas ir sutarties pasirašymas | Naujas užregistruotas aktyvus vartotojas ar klientas | Skolos grąžinimo sėkmė – pinigų pervedimas į įmonės sąskaitą |
| Skirtingos vertės maistas | Skirtingi nuomininkų dydžiai (dideli plotai, mažos patalpos) | Individualūs vartotojai arba partneriai, kurie atveda šimtus klientų | Mažos ir didelės skolos |
| Keli maisto vienetai vienu metu | Keli klientai vienu metu – reikia pasirinkti prioritetus | Vienu metu kelios kampanijos (reklama, socialiniai tinklai, partnerystės) | Keli aktyvūs išieškojimo procesai vienu metu |
| Laiko spaudimas | Pastato atidarymo data riboja laiką | Laikas iki investuotojų vertinimo datos | Ketvirčio pabaiga ir išieškojimo plano vykdymas |
| Premijos | Skirtingų sektorių klientų kombinacijos didina vertę | Svarbus ne tik vartotojų skaičius, bet ir jų aktyvumas | Prioritetų nustatymas pagal skolų dydį ir grąžinimo tikimybę |
| Kintančios sąlygos | Atsiranda nauji konkurentai ar keičiasi klientų poreikiai | Marketingo kanalų efektyvumo kitimas, rinkos prisotinimas | Skolininkų elgesio pasikeitimai, nauji teisiniai reikalavimai |

Šis struktūrinis aplinkos pritaikymas sudaro prielaidas kurti imitacinius scenarijus, leidžiančius sistemingai tirti agento elgseną, strateginio prisitaikymo gebėjimus ir ilgalaikį finansinį atlygį.

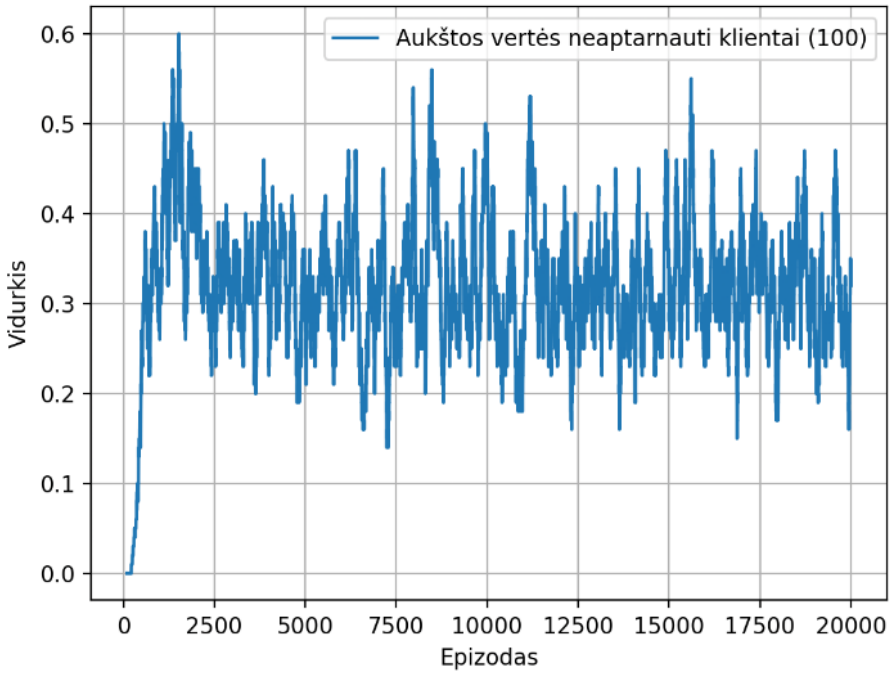
### 3.2.1. Aukštos vertės klientų aptarnavimo scenarijus ir parametrų kaita

Šiame skyriuje pateikiamas DQN ir PPO skatinamojo mokymosi metodų įvertinimas modeliuojant aukštos vertės klientų aptarnavimo imitacinę aplinką. DQN metodo mokymosi etapą įvertinti atvaizduojami pirmieji 20000 epizodų.




**9 pav.** Atlygio funkcijos slankusis vidurkis mokymo etape naudojant DQN metodą

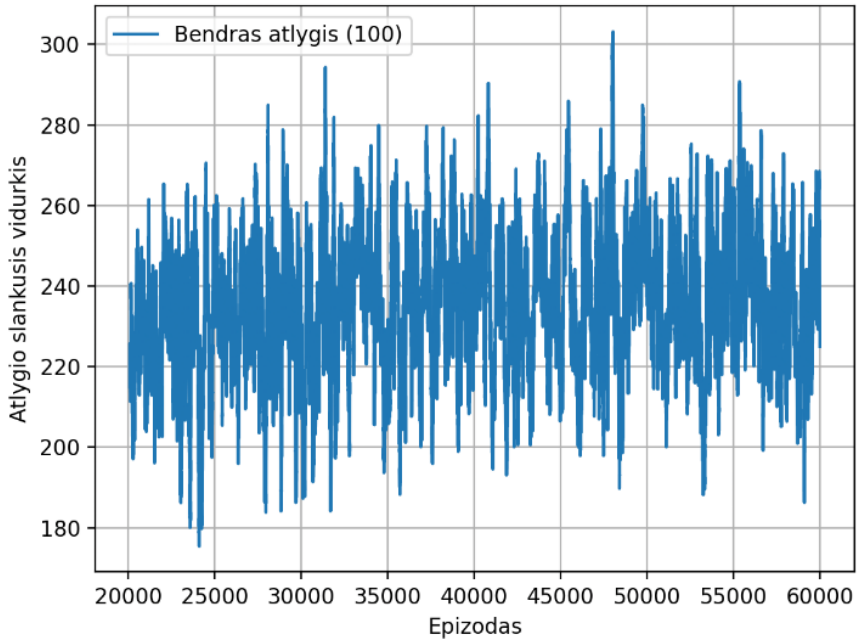
Pirmuose 2000 epizodų stebimas greitas atlygio augimas rodo, kad DQN metodas greitai konverguoja į bazinę politiką, leidžiančią efektyviai spręsti pirminį kliento aptarnavimo uždavinį. Vėlesniuose mokymo etapo epizoduose atlygis stabilizuojasi, reikšmės svyruoja be aiškios augimo tendencijos. Tai rodo, kad įmonė išlaiko išmoktą politiką, tačiau aiškios augimo tendencijos nėra.




**10 pav.** Aptarnautų klientų vidurkis mokymo etape naudojant DQN metodą

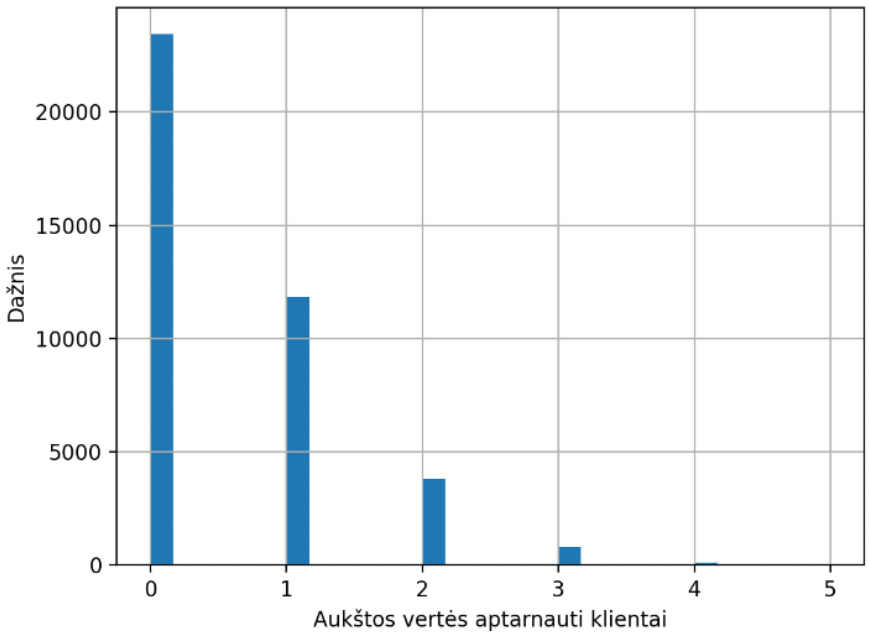
Grafike matomas svyruojantis aukštos vertės neaptarnautų klientų vidurkis rodo, kad įmonės politika nėra stabili šio rodiklio atžvilgiu. Momentiniai vidurkio sumažėjimai leidžia daryti prielaidą, kad

įmonė atsižvelgia į aukštos vertės klientų buvimą aplinkoje ir įtraukia šio tipo klientus į veikimo strategiją. Toliau bus apžvelgiami tyrinėjimo etapo rezultatai.



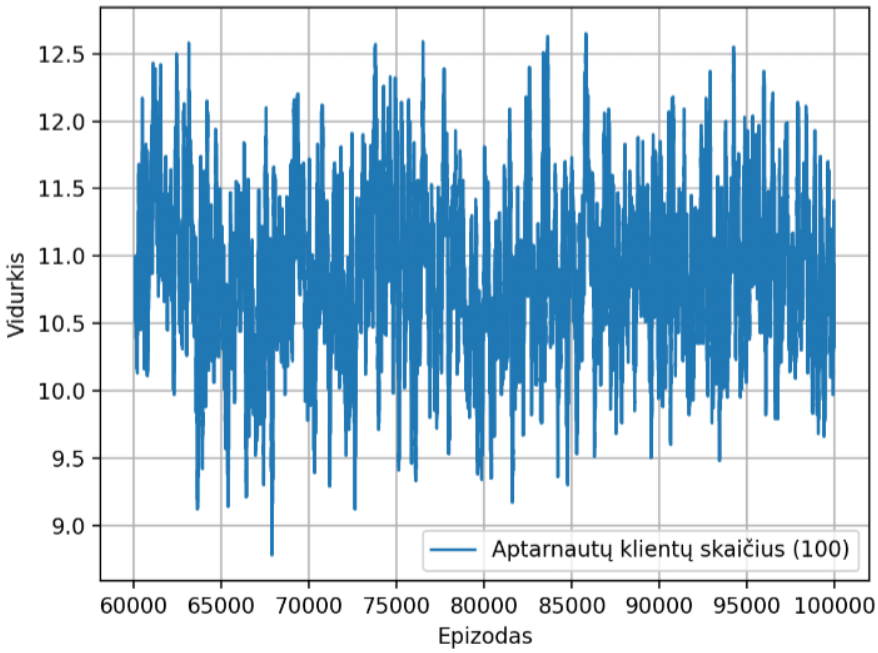

**11 pav.** Atlygio funkcijos slankusis vidurkis tyrinėjimo etape naudojant DQN metodą

Tyrinėjimo etape atlygio funkcijos slankusis vidurkis svyruoja, su momentiniais šuoliais virš 280 bendrojo atlygio vidurkio reikšmės, tačiau be ilgalaikės augimo tendencijos. Šio etapo bendras vidurkis 217,75 rodo, kad Q-funkcijos aproksimacija priartėjo prie lokalių ekstremumų, tačiau nepasiekė globaliai stabilaus sprendinio. Rezultatai atskleidžia didelę politikos dispersiją, kas gali indikuoti nepakankamą optimalios veikimo strategijos išmokimą arba per didelį jautrumą epizodinėms aplinkos sąlygoms.




**12 pav.** Aptarnautų VIP klientų dažnių lentelė tyrinėjimo etape naudojant DQN metodą

Tyrinėjimo etapo analizė rodo, kad daugiau nei 50 % epizodų įmonė neaptarnauja nė vieno aukštos vertės kliento, o daugiau nei vieną tokį klientą aptarnauja tik 6 % epizodų. Toks dažnių pasiskirstymas rodo, kad taikomoje strategijoje VIP klientai nėra prioritetas. Nepaisant ilgesnio epizodų intervalo, įmonės politika išlieka nepakankamai jautri retiems, tačiau reikšmingą atlygį generuojamiems įvykiams.




**13 pav.** Aptarnautų klientų vidurkis taikymo etape naudojant DQN metodą

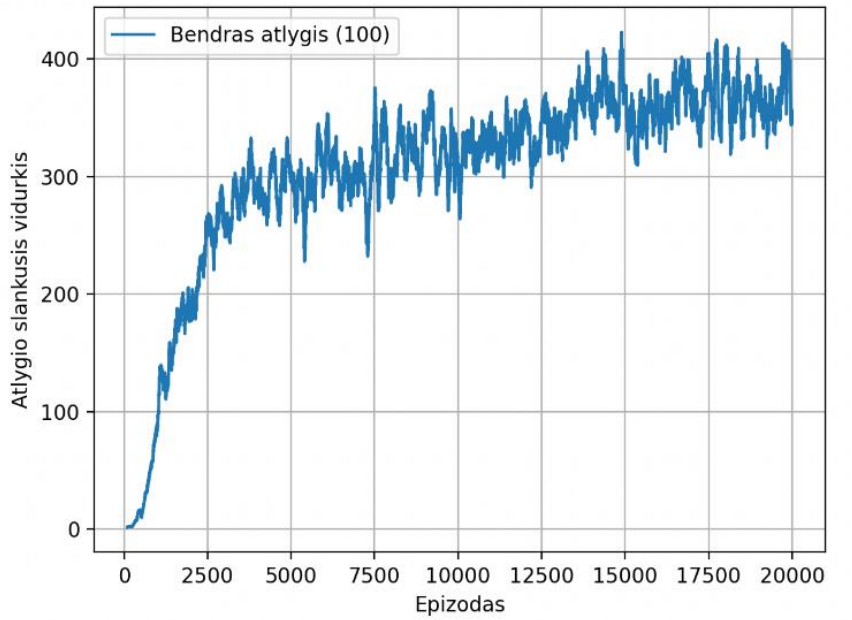
Taikymo etape vidutinis aptarnautų klientų skaičius siekia 10,8, o maksimali epizodinė reikšmė – 32 klientai. Tarp epizodų rezultatų išlieka didelė dispersija, patvirtinanti, kad įmonė atlieka tyrinėjimą, bet nesugeba stabilizuoti efektyvesnės strategijos.

**6 lentelė.** Aukštos vertės klientų aplinkos DQN metodo rodiklių statistika taikymo etape

| **Rodiklis** | **Vidurkis** | **Dispersija** | **75% kvartilis** | **Maksimumas** |
|---|---|---|---|---|
| Aptarnauti klientai | 10,8 | 5.9 | 15 | 32 |
| VIP aptarnauti klientai | 0,57 | 0.78 | 1 | 5 |
| VIP neaptarnauti klientai | 0,32 | 0.62 | 0 | 5 |
| Atlygis už efektyvų judėjimą | 30 | 1.72 | 42 | 114,5 |
| Atlygis už neefektyvų judėjimą | -1,5 | 0.9 | -2,1 | -6,94 |

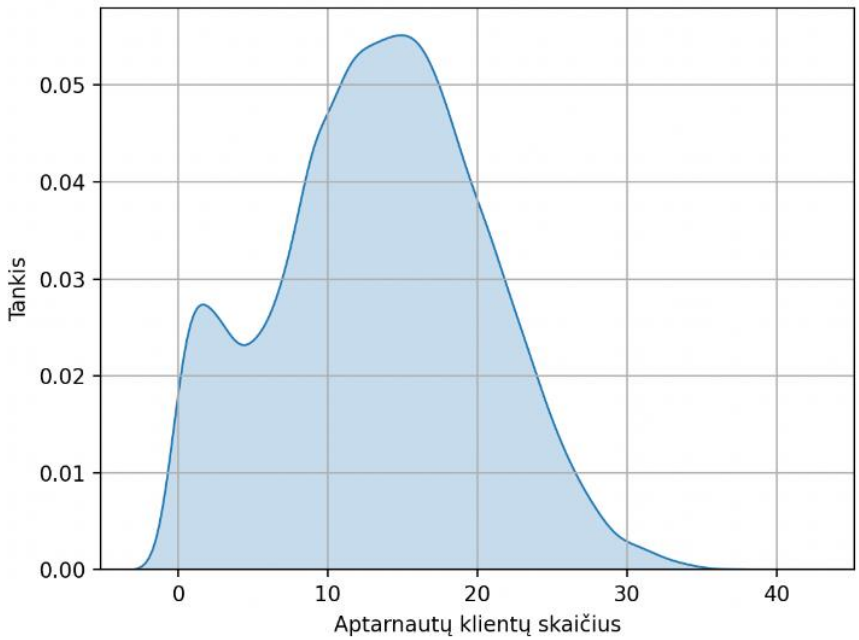
Statistinė rodiklių analizė taikymo etape rodo, kad VIP klientų aptarnavimas išlieka retas reiškinys (vidurkis = 0,57), o neaptarnautų VIP klientų vidurkis (0,32) rodo, nepakankamą įmonės prisitaikymą prie atlygio vertės maksimizavimo sąlygų imituojamoje verslo procesų aplinkoje. Šie rezultatai kartu su žemu efektyvaus judėjimo atlygio vidurkiu ir nedidelėmis baudomis už neefektyvų judėjimą leidžia daryti prielaidą, kad dauguma epizodų trunka trumpai ir generuoja ribotą naudą. Nors

epizodinė maksimali aptarnautų klientų reikšmė (32 klientai) rodo potencialą pasiekti pelningus rezultatus, strategija nukreipimas į aukštos vertės klientų aptarnavimą išlieka neoptimizuotas.



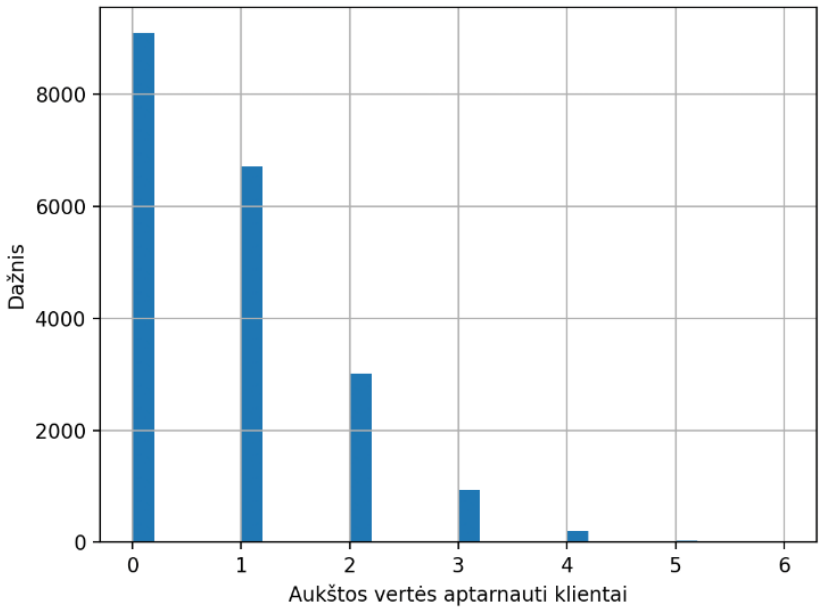

**14 pav.** Atlygio funkcijos slankusis vidurkis mokymo etape naudojant PPO metodą

PPO metodas demonstruoja greitesnį ir tolygesnį atlygio funkcijos augimą nei DQN metodas. Jau iki 5000 epizodo, atlygio slankusis vidurkis viršija 300 reikšmę, o augimo dinamika išlieka teigiama viso mokymo etapu. Šis rezultatas rodo efektyvesnį politikos suformavimą ir stabilesnį atlygio funkcijos optimizavimą.



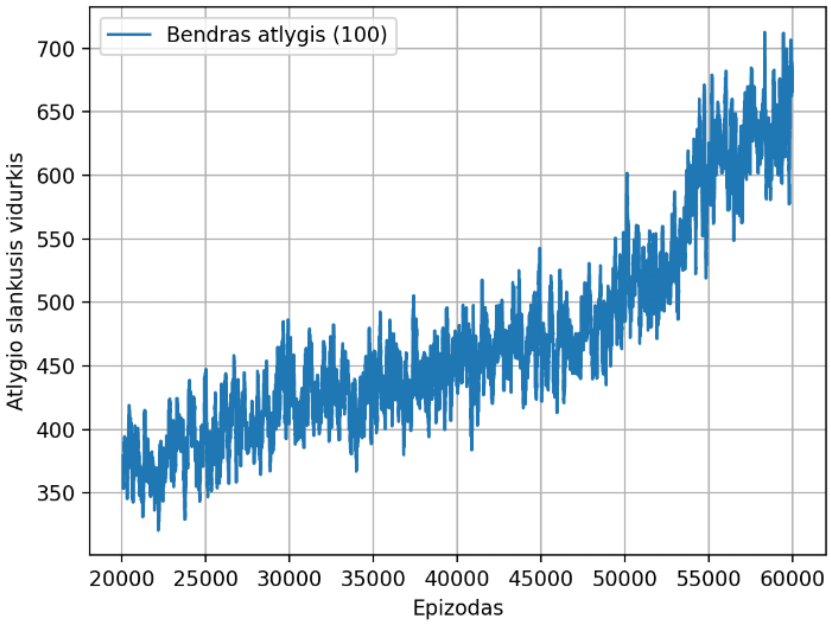

**15 pav.** Aptarnautų klientų tankio funkcija mokymo etape naudojant PPO metodą

Aptarnautų klientų tankio funkcija rodo, kad PPO metodo išmokta politika leidžia ne tik stabiliai pasiekti 15 klientų vidurkį, bet ir epizodiškai taikyti optimalias strategijas, kurios leidžia aptarnauti 30 ir daugiau klientų vieno epizodo metu. Pasiskirstymo asimetrija rodo, kad įmonė geba išnaudoti palankias aplinkos sąlygas generuojant aukštesnį atlygį.



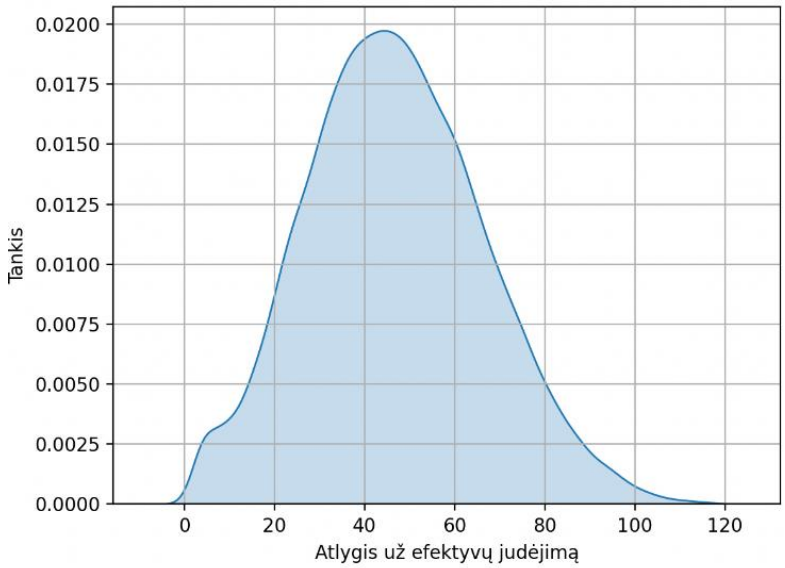

**16 pav.** VIP aptarnautų klientų dažniai mokymo etape naudojant PPO metodą

Dažnių histogramoje matyti, kad daugiau nei pusėje epizodų buvo aptarnautas bent vienas VIP klientas, o reikšminga dalis epizodų pasiekė du ar daugiau VIP klientų aptarnavimo atvejų. Pasiskirstymas rodo didesnį nuoseklumą įmonės elgsenoje VIP klientų atžvilgiu, lyginant su DQN metodu. Tai reiškia, kad PPO politika struktūriškai geriau atpažįsta ir išnaudoja aukštos vertės klientų teikiamą potencialą. Toks strateginis prisitaikymas rodo ne tik efektyvesnę mokymosi eigą, bet ir tikslesnį politikos suformavimą, orientuotą į atlygio maksimizavimą.



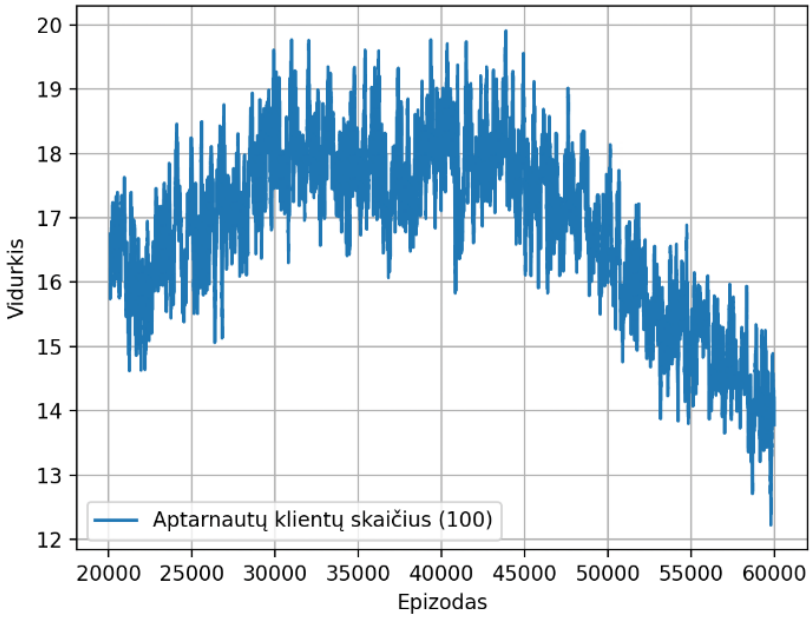

**17 pav.** Atlygio funkcijos slankusis vidurkis tyrinėjimo etape naudojant PPO metodą

Tyrinėjimo etape stebimas nuoseklus bendro atlygio didėjimas, kuris nuo 20000 iki 60000 epizodo pasiekia virš 700 atlygio vidurkio reikšmę. Ši augimo dinamika rodo stabilų politikos tobulinimą ir didėjantį aplinkos supratimą, leidžiantį efektyviau maksimizuoti atlygį. Nuo 45000 epizodo fiksuojamas spartesnis augimo tempas, kuris leidžia daryti prielaidą apie perėjimą prie aukšto efektyvumo strategijos.



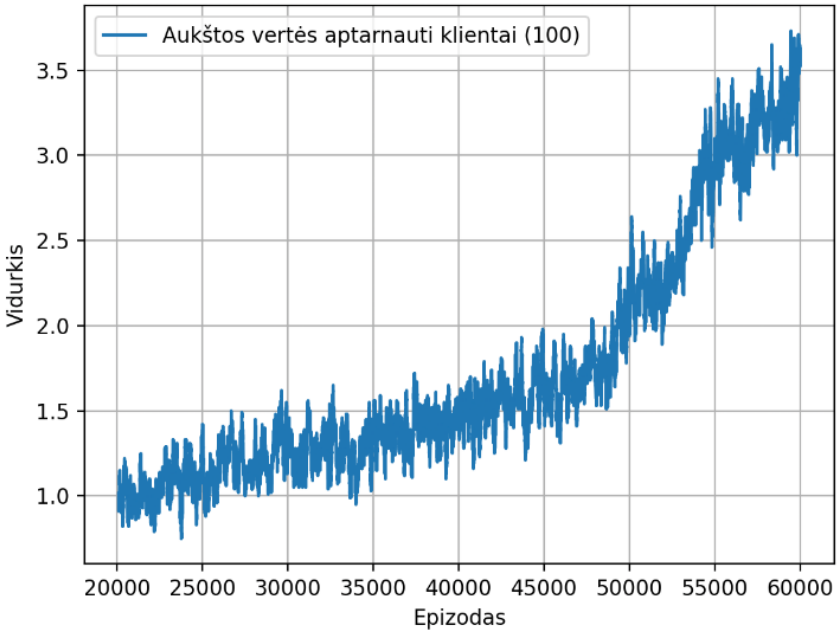

**18 pav.** Atlygio už efektyvų judėjimą tankio funkcija tyrinėjimo etape naudojant PPO metodą

Tankio funkcijos maksimumas stebimas ties 46 atlygio verte, kas rodo, jog įmonės priimami sprendimai dalinai atitinka optimizuotą strategiją. Toks pasiskirstymas patvirtina, kad PPO metodas leidžia įmonei išmokti ne tik stabiliai efektyvią, bet ir galimai optimalią veikimo strategiją aukštos vertės klientų imitacinėje aplinkoje.




**19 pav**. Aptarnautų klientų vidurkis tyrinėjimo etape naudojant PPO metodą

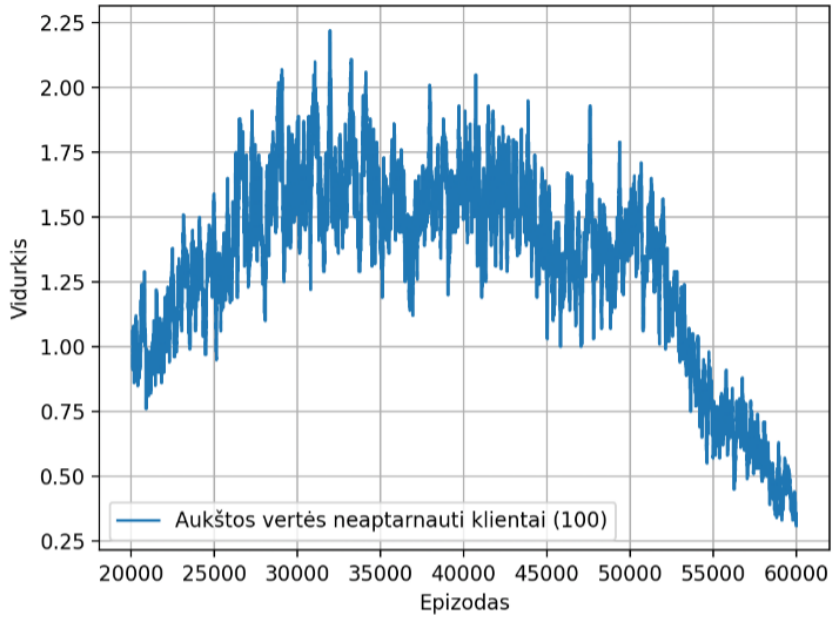
Aptarnautų klientų vidurkio sumažėjimas nuo 18 iki 14 gali būti klaidingai interpretuojamas kaip strateginės regresijos požymis. Vis dėlto, šis aptarnautų klientų sumažėjimas koreliuoja su tuo pačiu laikotarpiu fiksuojamu VIP klientų aptarnavimo augimu ir bendro atlygio reikšmių šuoliais. Tokia dinamika rodo, kad PPO keičia savo taikomą politiką nuo kiekybinio klientų skaičiaus maksimizavimo link aukštesnio individualaus atlygio, optimizuodamas sprendimų kokybę vietoje jų kiekio.




**20 pav.** VIP aptarnautų klientų vidurkis tyrinėjimo etape naudojant PPO metodą

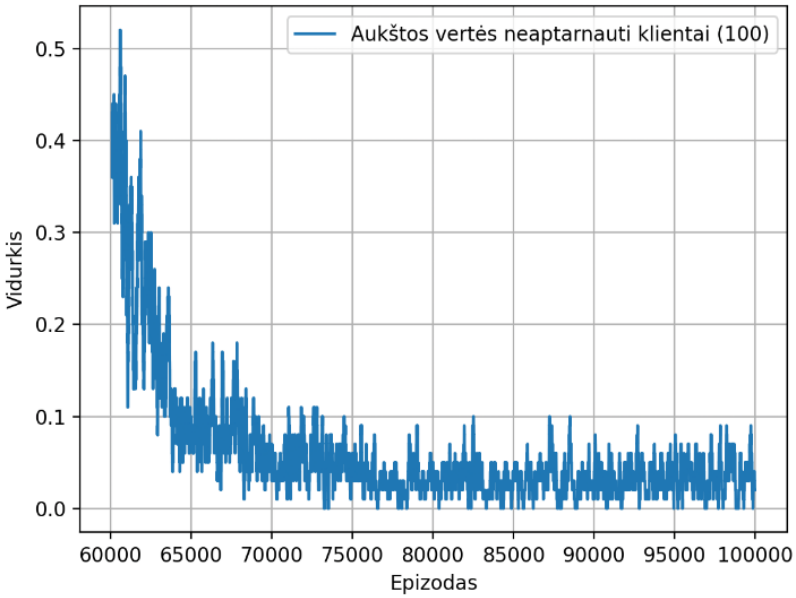
Grafikas rodo nuoseklų VIP klientų aptarnavimo vidurkio augimą iki 3,5 aukštos vertės aptarnautų klientų, kas parodo reikšmingą PPO politikos prisitaikymą. Ši dinamika rodo, kad perėjus iš mokymo ir tyrinėjimo etapą įmonė perorientuoja sprendimų priėmimą prioritetu laikant VIP klientų

aptarnavimą kaip reikšmingą optimizavimo tikslą. Augimo trajektorija koreliuoja su bendro atlygio didėjimu, patvirtindama,, kad VIP klientų integracija tiesiogiai prisideda prie atlygio maksimizavimo.



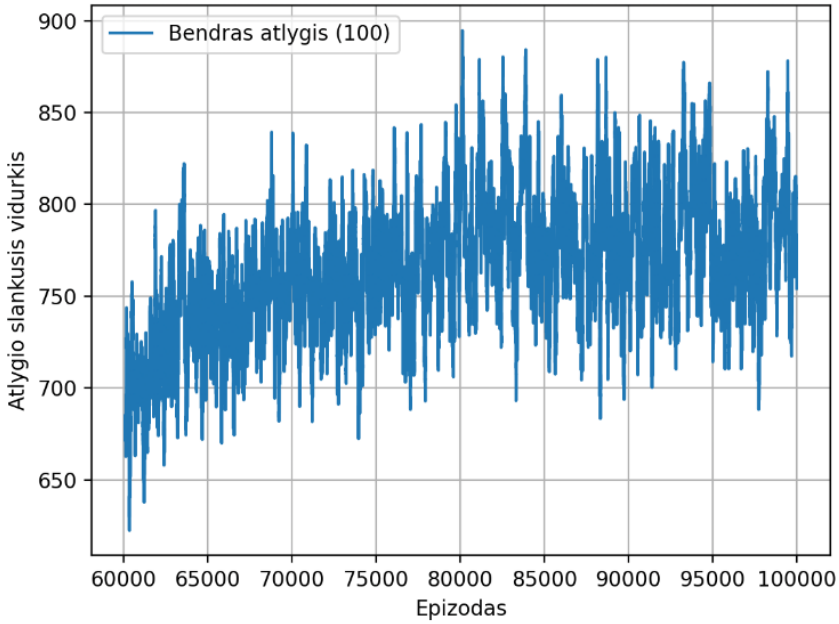

**21 pav.** VIP neaptarnautų klientų vidurkis tyrinėjimo etape naudojant PPO metodą

Neaptarnautų VIP klientų vidurkio kitimas nuo 2 iki mažiau nei 0,5 rodo, kad PPO ne tik siekia VIP klientų, bet ir vis rečiau juos praleidžia. Tai reiškia, kad įmonės sprendimų kokybė pagerėja dėl tikslesnio veiksmų plano vykdymo. Neaptarnautų VIP klientų vidurkio sumažėjimas nuo 2 iki 0,5 per epizodą rodo, kad PPO politika ne tik apima VIP klientų identifikavimą, bet ir sistemingai mažina jų praleidimo tikimybę. Tai reiškia, kad įmonės sprendimų kokybė pagerėja dėl tikslesnio veiksmų plano vykdymo.




**22 pav.** VIP neaptarnautų klientų vidurkis taikymo etape naudojant PPO metodą

Per pirmuosius 5000 taikymo etapo epizodų vidutinis neaptarnautų VIP klientų skaičius sumažėja iki 0,1 ir vėliau stabilizuojasi artimoje nuliui reikšmėje. Ši dinamika rodo, kad PPO politika pasiekia beveik pilną aukštos vertės klientų pasiekiamumą, o sprendimų priėmimo nuoseklumas išlieka aukštas ir vėlesniuose etapo intervaluose. Tokie rezultatai rodo maksimalų įmonės prisitaikymą prie optimizuoto elgesio aukštos vertės klientų atžvilgiu.




**23 pav.** Atlygio funkcijos slankusis vidurkis taikymo etape naudojant PPO metodą

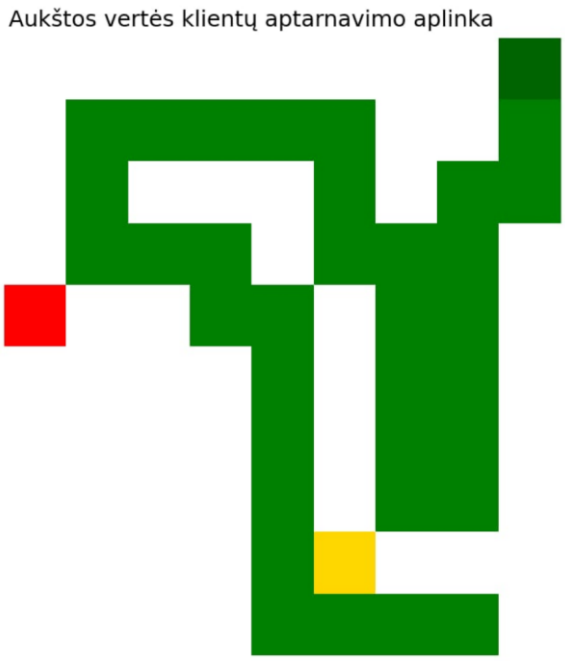
Bendras atlygio slankusis vidurkis toliau nuosekliai didėja, artėdamas prie 850 reikšmės, o tai rodo, kad įmonė toliau tobulina sprendimų priėmimo politiką. Nors fiksuojami didesni svyravimai, tai laikoma natūraliu šalutiniu reiškiniu taikant sudėtingesnes ir labiau pelningumą maksimizuojančias strategijas.

**7 lentelė.** Aukštos vertės klientų aplinkos PPO metodo rodiklių statistika taikymo etape

| **Rodiklis** | **Vidurkis** | **Dispersija** | **75% kvartilis** | **Maksimumas** |
|---|---|---|---|---|
| Aptarnauti klientai | 14 | 5,11 | 18 | 34 |
| VIP aptarnauti klientai | 4,1 | 1,7 | 5 | 11 |
| VIP neaptarnauti klientai | 0,06 | 0,27 | 0 | 4 |
| Atlygis už efektyvų judėjimą | 39 | 16 | 50 | 106,5 |
| Atlygis už neefektyvų judėjimą | -2,7 | -1,3 | -3,5 | -7,5 |

Pateikti statistiniai rodikliai rodo, kad PPO išmokta politika pasižymi aukštu strateginiu efektyvumu. Vieno epizodo metu vidutiniškai aptarnaujami 14 paprastų ir 4,1 aukštos vertės klientai. Taip pat 25 % epizodų pasiekiamas bent 5 VIP klientų aptarnavimas, o maksimali aukštos vertės aptarnautų klientų reikšmė lygi 11. Neaptarnautų VIP klientų rodikliai yra itin žemi, vidurkis siekia vos 0,06, o reikšmių pasiskirstymas rodo, kad daugumoje epizodų visi VIP klientai yra sėkmingai aptarnaujami.

Vis dėlto, kai kuriais atvejais fiksuojamas neoptimalaus veikimo epizodas, galimai dėl padidėjusio aplinkos sudėtingumo ar netipinės aplinkos dinamikos.




**24 pav.** Aukštos vertės klientų aptarnavimo aplinkos sužaidybinimas

Pateikiamas vieno epizodo sužaidybinimo vaizdas VIP aplinkoje, taikant PPO metodą, kuriame atvaizduojama įmonės elgsena ir aplinkos būsena. Aplinkoje tamsiai žalia spalva žymima įmonės dalis atsakinga už sprendimų priėmimą, žalia spalva simbolizuoja įmonės sukauptus resursus, geltona spalva vaizduoja aukštos vertės kliento padėtį aplinkoje, o raudona spalva įprasto tipo klientą.

**8 lentelė.** Bendrojo atlygio statistika aukštos vertės klientų aptarnavimo aplinkoje

| **Rodiklis** | **Mokymo etapas** | | **Tyrinėjimo etapas** | | **Taikymo etapas** | |
|---|---|---|---|---|---|---|
| | **DQN** | **PPO** | **DQN** | **PPO** | **DQN** | **PPO** |
| Vidurkis | 208,7 | 305 | 235,1 | 471,4 | 239,5 | 765,8 |
| Dispersija | 161,8 | 195,13 | 163,5 | 235 | 165 | 302,1 |
| Mediana | 174,5 | 291,5 | 211,75 | 447,5 | 215,9 | 758,9 |
| 75% kvartilis | 307,6 | 429.8 | 337,2 | 615,4 | 343,8 | 951 |
| Maksimumas | 1224,7 | 1259,3 | 1049,4 | 1641,1 | 1076,4 | 2011,2 |

Pateikta bendrojo atlygio statistika aiškiai parodo reikšmingą našumo skirtumą tarp DQN ir PPO metodų visais etapais. Jau mokymo etape PPO pasiekia ženkliai aukštesnį vidutinį atlygį (305) nei DQN (208,7). Šis skirtumas dar labiau išryškėja tyrinėjimo etape ir ypač taikymo etape, kur PPO rezultatas (765,8) yra daugiau nei tris kartus didesnis už DQN (239,5).

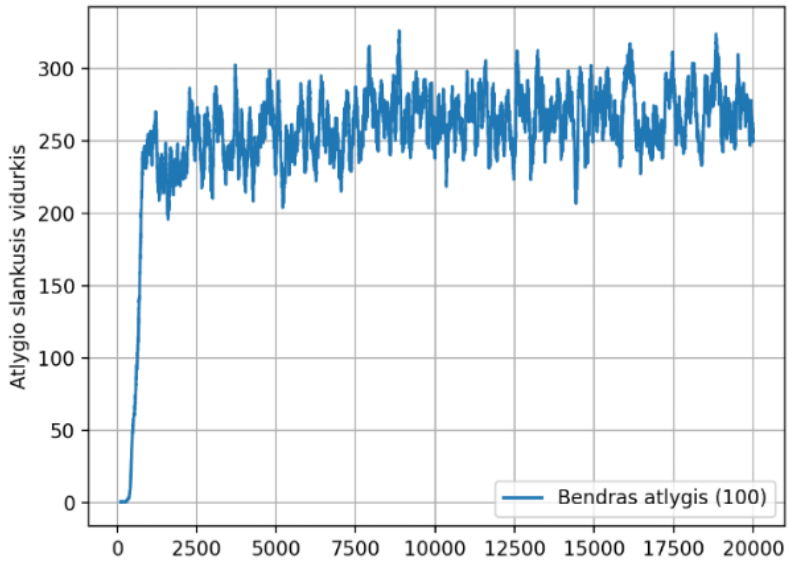
Nors PPO metodo dispersija taip pat didesnė, tai siejasi su aukštesnėmis reikšmėmis viršutinėje pasiskirstymo dalyje. PPO metodo efektyvumo tankis VIP klientų aptarnavime siekia 2,41, tuo tarpu DQN vos 0,73. PPO metodas ne tik sumažino aukštos vertės klientų neaptarnavimo atvejus, bet ir optimizavo judėjimo logiką, parodydamas aukštesnę 75 % kvartilio reikšmę už DQN metodą daugiau nei 2,7 karto. Šis skirtumas rodo, kad PPO ne tik vidutiniškai aptarnauja ženkliai daugiau VIP klientų, bet daro tai kur kas nuosekliau, su mažesniu rezultatų svyravimu. PPO medianos ir 75 % kvartilio

reikšmės kiekviename etape viršija analogiškas DQN reikšmes, kas rodo, kad didžioji PPO epizodų dalis generuoja santykinai aukštą atlygį. Taip pat PPO pasiekia žymiai aukštesnes maksimalias reikšmes visuose etapuose, ypač taikymo metu – 2011,2, palyginti su 954,1 DQN atveju.

Šie rezultatai patvirtina, kad PPO ne tik greičiau optimizuoja veikimo politiką, bet ir efektyviau išnaudoja mokymosi potencialą strategijos tobulinimui. Tuo tarpu DQN rezultatai rodo elgesio konvergavimą į lokalų optimumą, su ribotu gebėjimu pereiti prie globaliai pelningesnių sprendimų.

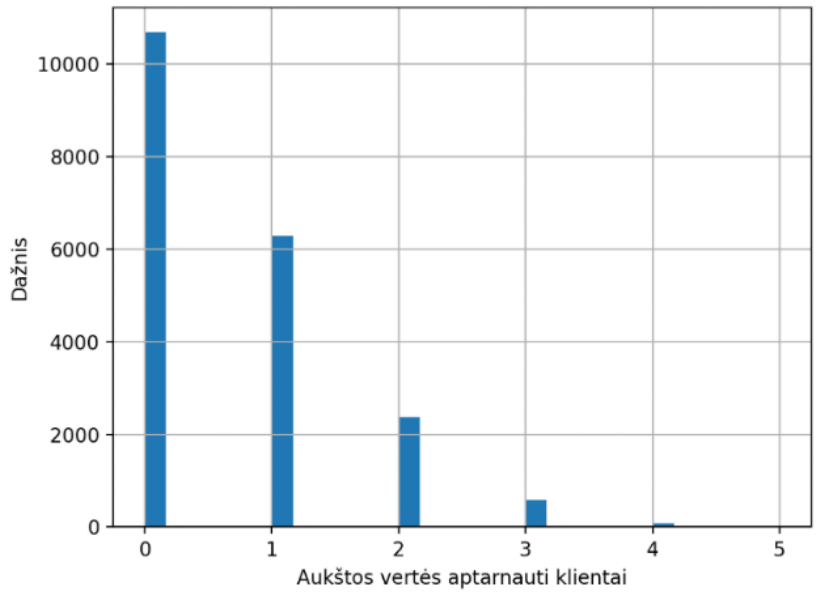
### 3.2.2. Rizikingų užsakymų aptarnavimo scenarijus esant rinkos kliūtims ir parametrų kaita

Vertinant DQN metodo taikymą sudėtingesnėje verslo procesų imitacinėje aplinkoje, kurioje egzistuoja rinkos kliūtys ir rizikingi klientai, siekiama įvertinti įmonės gebėjimą prisitaikyti prie neapibrėžtų, kintančių rizikos veiksnių. Taikant DQN metodą mokymo etape gaunami tokie rezultatai:



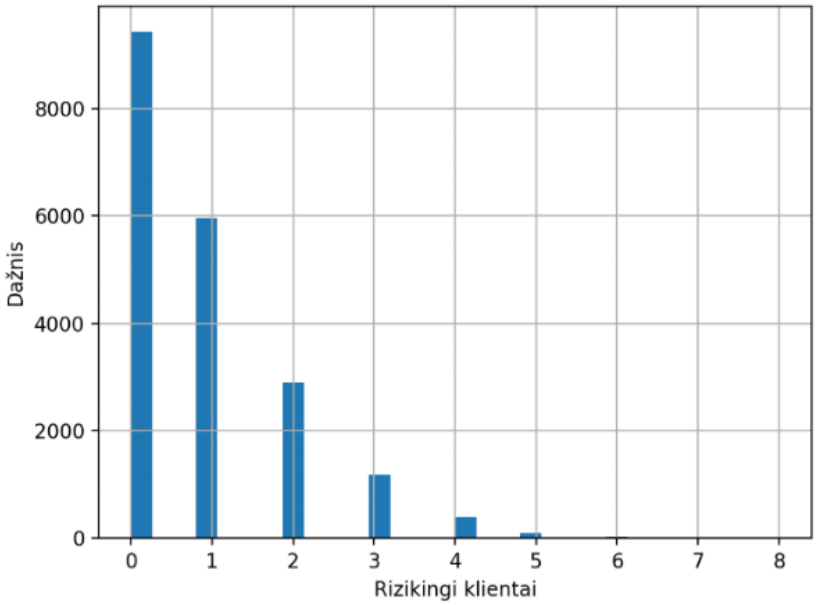

**25 pav.** Atlygio funkcijos slankusis vidurkis mokymo etape naudojant DQN metodą

Per pirmuosius 2000 epizodų stebimas reikšmingas bendro atlygio didėjimas iki 250 vertės. Mokymo etapas atspindi modelio gebėjimo išmokti vengti nuostolingiausių veiksmų, tokių kaip rizikingų užsakymų vykdymas ar netinkamas reagavimas į rinkos kliūtis. Ilgalaikis atlygis išlieka stabilus be aiškios tolimesnės augimo tendencijos mokymo etapo epizoduose.



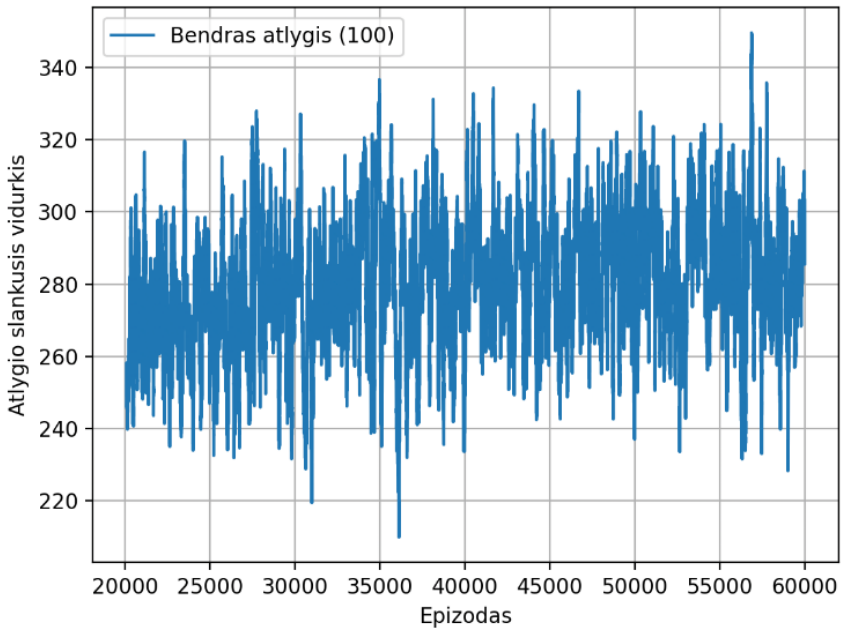

**26 pav.** Aptarnautų VIP klientų dažnių lentelė mokymo etape naudojant DQN metodą

Analizuojat mokymo etapo metu gautus rezultatus, nustatyta, kad daugiau nei 50 % epizodų įmonė neaptarnauja nė vieno aukštos vertės kliento. Vis dėlto pasitaiko tokių epizodų kai aptarnaujami net keturi aukštos vertės klientai, nors optimali veikimo politika dar nėra suformuota. Atsižvelgiant į tai, kad DQN metodas optimizuoja sprendimų priėmimą dažnai pasikartojančia patirtimi ir siekia sumažinti tikėtiną nuostolį, rečiau pasitaikantys įvykiai, turi ribotą įtaką Q-funkcijos parametrų atnaujinimui. Šis metodas formuoja strategiją, kuri prioritetą skiria baudų vengimui, o ne rizikingiems, bet potencialiai didesnį atlygį generuojantiems sprendimams.



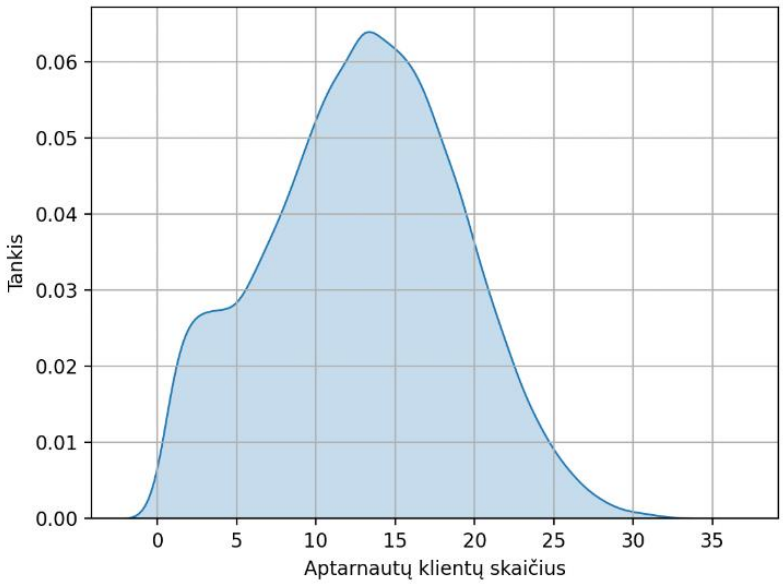

**27 pav.** Aptarnautų rizikingų klientų dažnių lentelė mokymo etape naudojant DQN metodą

Epizodų, kuriuose įvykdomi trys ar daugiau rizikingų užsakymų, dalis sudaro tik 10 % visų mokymo etapo atvejų. Toks aptarnautų rizikingų klientų pasiskirstymas rodo, kad suformuota veikimo politika orientuota į rizikos vengimą ir teikia pirmenybę veiksmams, kurie sumažina baudų tikimybę bei rezultatų dispersiją.



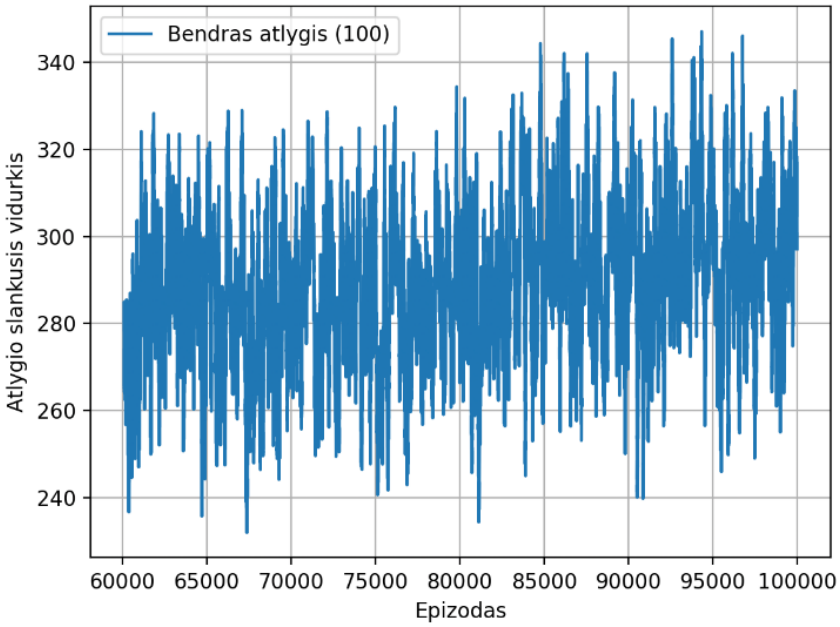

**28 pav.** Atlygio funkcijos slankusis vidurkis tyrinėjimo etape naudojant DQN metodą

Tyrinėjimo etape analizuojant atlygio funkciją, fiksuojami periodiškai pasikartojantys šuoliai, kai atlygio slankiojo vidurkio reikšmė viršija 320 vertę. Tokie epizodiniai kilimai indikuoja, kad įmonė geba atrasti strateginius veiksmų sekos sprendimus, kurie užtikrina didesnį atlygį. Nors dauguma stebimų atlygio reikšmių yra tarp 270 ir 300, ekstremumų reikšmės rodo, jog įmonė išlaiko gebėjimą prisitaikyti prie aplinkos kaitos ir periodiškai pasiekia aukštą naudą generuojančias strategijas.




**29 pav.** Aptarnautų klientų tankio funkcija tyrinėjimo etape naudojant DQN metodą

Aptarnautų klientų tankio funkcija rodo, kad įmonės suformuota veikimo politika pasižymi efektyvumu ir nuoseklumu. Didžiausia tikimybė, kad tyrinėjimo etape bus aptarnaujama 13 klientų, o tai rodo, kad sprendimų priėmimas yra stabilus. Tankio funkcija leidžia daryti prielaidą, kad dauguma epizodų generuoja vienodą sprendimų kokybę, o pasirinkta strategija sugeba palaikyti aptarnavimo lygį esant atsitiktinei veiksmų atrankai, neprarandant efektyvumo.




**30 pav.** Atlygio funkcijos slankusis vidurkis taikymo etape naudojant DQN metodą

Bendrojo atlygio reikšmės viso taikymo etapo metu išlieka reikšmingai kintančios. Nors vidutinė atlygio vertė yra aukštesnė nei tyrinėjimo etape, nepastebima stabilumo požymių, kurie leistų daryti išvadą apie galutinai suformuotą veikimo politiką. Tai reiškia, kad įmonės sprendimų strategija išlieka jautri aplinkos pokyčiams. Staigūs atlygio šuoliai susiję su epizodais, kuriuose efektyviai aptarnaujami VIP klientai arba įvykdomi rizikingi užsakymai. Tačiau šie atvejai nėra dažni ir nesudaro ilgalaikės, optimizuotos veikimo politikos pagrindo.

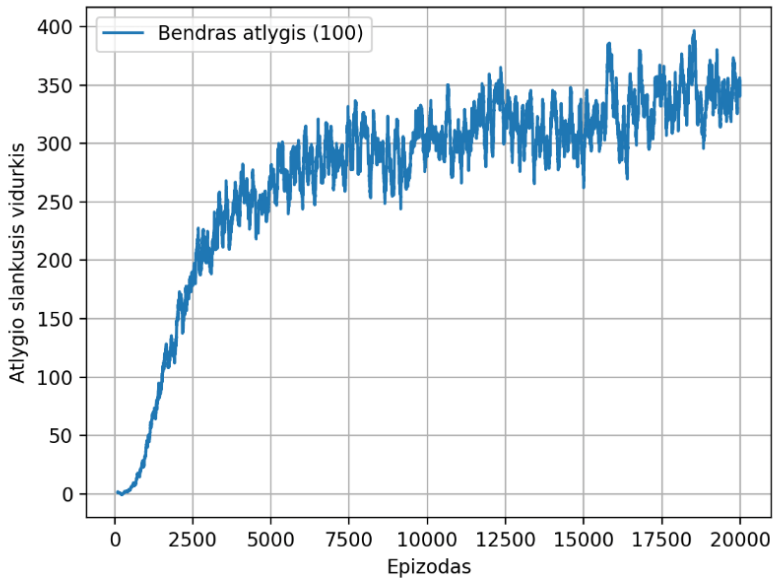
**9 lentelė.** Rizikingų užsakymų aptarnavimo aplinkos DQN rodiklių statistika taikymo etape

| **Rodiklis** | **Vidurkis** | **Dispersija** | **75% kvartilis** | **Maksimumas** |
|---|---|---|---|---|
| Aptarnauti klientai | 13,3 | 6 | 18 | 34 |
| VIP aptarnauti klientai | 0,74 | 0,87 | 1 | 6 |
| VIP neaptarnauti klientai | 0,5 | 0,76 | 1 | 5 |
| Atlygis už efektyvų judėjimą | 36,3 | 17,7 | 48,5 | 110 |
| Atlygis už neefektyvų judėjimą | -1,89 | -1 | -2,5 | -7.32 |
| Aptarnauti rizikingi klientai | 1 | 1,12 | 2 | 8 |
| Susidūrimas su kliūtimis | 0,14 | 0,12 | 0 | 1 |
| Išnykusios kliūtys | 0,6 | 1,2 | 0 | 10 |

Taikymo etape vidutinis aptarnautų klientų skaičius siekia 13,3, tačiau dispersijos reikšmė rodo didelį rezultatų svyravimą tarp epizodų. Toks nuokrypis leidžia daryti išvadą, kad įmonės veikimo politika nėra optimizuota visame taikymo etape. Nors maksimalus epizodinis klientų skaičius pasiekia 34, tai rodo aukštą maksimumo reikšmę, bet neatspindi taikomos strategijos vidutinių rezultatų. VIP klientų aptarnavimo vidurkis siekia vos 0,74, o tai reikšmingai artima neaptarnautų VIP klientų vidurkiui

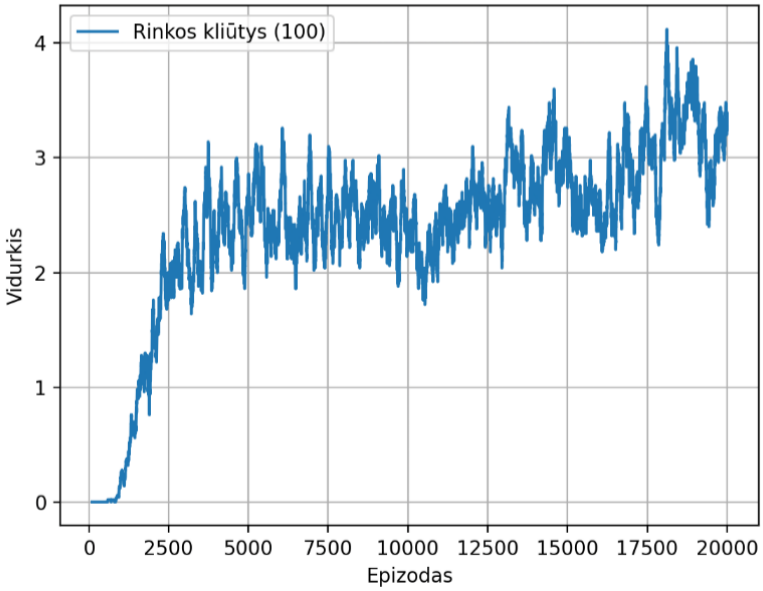
(0,5). Tai rodo, kad įmonė neįgyvendina aiškios strategijos aukštos vertės klientų aptarnavimui. Išskirtiniais atvejais įmonei pavyksta aptarnauti iki 6 VIP klientų per epizodą, tačiau tai nėra dominuojanti tendencija. Atlygio už efektyvų judėjimą vidurkis siekia 36,3, tačiau dispersija patvirtina epizodinį veikimo kokybės nepastovumą. Tai rodo, kad optimalus kliento pasiekimas nėra išlaikomas viso taikymo etapo metu. Vidutiniškai taikymo etape aptarnaujamas vienas rizikingas klientas, o daugiausiai per vieną epizodą pasiekiamas 8 rizikingų užsakymų priėmimas. Tai reiškia, kad įmonės strategijoje įtraukiami ir rizikingi sprendimai, kurių pavyksta išvengti. Tai patvirtina maža susidūrimo su kliūtimis vidurkio reikšmė, įmonė geba valdyti paprastus išteklių ir aplinkos pokyčių trikdžius, o klaidų dažnis, susijęs su netinkamu aplinkos vertinimu, lieka žemas (0,14).

Toliau bus atliekama PPO metodo rezultatų analizė išplėstinėje verslo procesų imitacinėje aplinkoje.



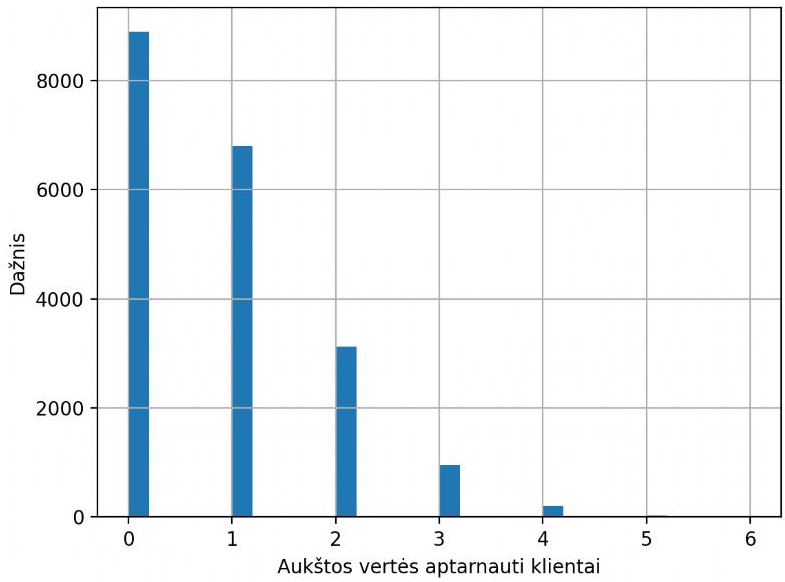

**31 pav.** Atlygio funkcijos slankusis vidurkis mokymo etape naudojant PPO metodą

Bendrojo atlygio funkcija rodo tolygią augimo tendenciją, prasidedančią nuo pat mokymo etapo pradžios. Tolimesnėje mokymo eigoje atlygis nuosekliai didėja ir artėja iki 350 atlygio vidurkio reikšmės. Toks augimo pobūdis leidžia daryti išvadą, kad PPO metodas pasižymi geresniu konvergavimu nei DQN, efektyviau identifikuoja naudą generuojančius sprendimus ir greičiau formuoja stabilią politiką.



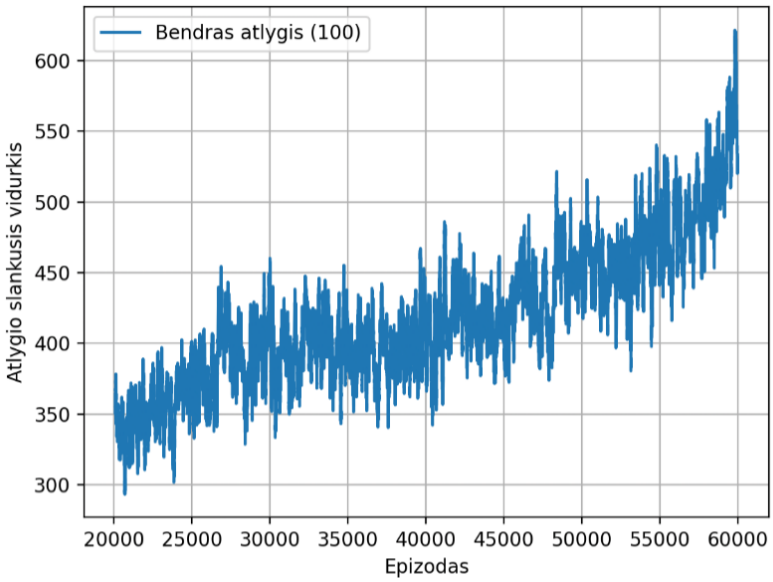

**32 pav.** Rinkos kliūčių aplinkoje vidurkis mokymo etape

Per pirmuosius 5000 epizodų rinkos kliūčių vidurkis pakyla iki 2,5, o vėlesniuose mokymo etapo epizoduose kreivė išlaiko augimo tendenciją. Šie rezultatai rodo, kad įmonė mokymosi pradžioje dar nėra pakankamai efektyvi valdant skirtingus scenarijus, susijusius su VIP klientų aptarnavimo vėlavimu ar netinkamų sprendimų pasekmėmis, dėl to aplinkoje susidaro naujos rinkos kliūtys.



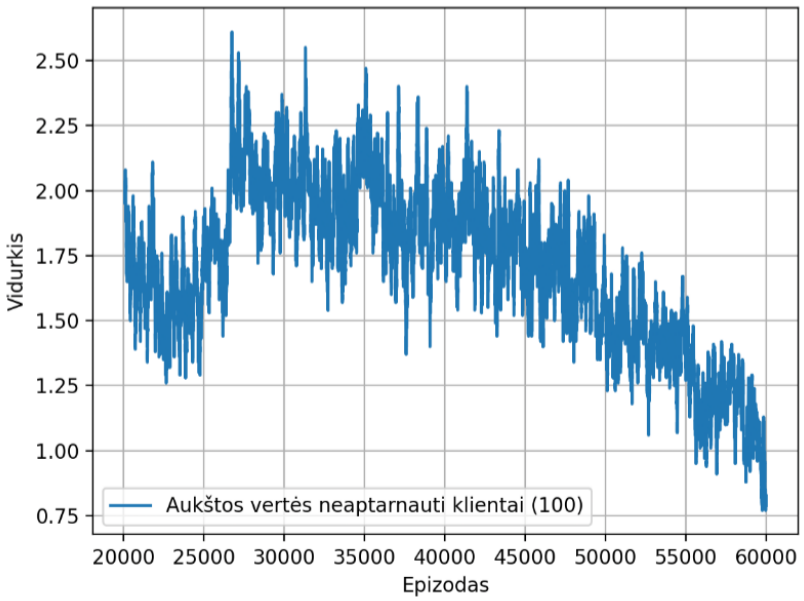

**33 pav.** VIP aptarnautų klientų dažniai mokymo etape naudojant PPO metodą

Daugiau nei 40 % epizodų pasibaigia neaptarnavus nė vieno VIP kliento, tačiau, lyginant su DQN, PPO metodas fiksuoja žymiai didesnį epizodų skaičių, kuriuose aptarnaujami du ar daugiau aukštos vertės klientų. Toks pasiskirstymas leidžia daryti išvadą, kad nors ir VIP klientų aptarnavimas dar nėra dominuojantis esamoje politikoje, įmonė pradeda identifikuoti šių klientų teikiamą naudą ir integruoja tai į sprendimų priėmimą dažniau nei DQN.




**34 pav.** Atlygio funkcijos slankusis vidurkis tyrinėjimo etape naudojant PPO metodą

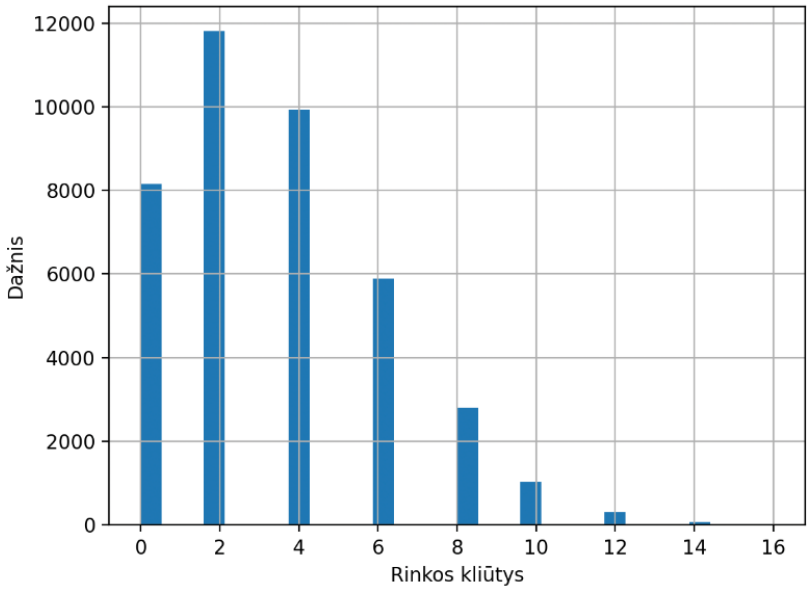
Tyrinėjimo etape atlygio funkcija rodo nuoseklų augimą. Tokia dinamika indikuoja, kad PPO tyrinėjimo etape ne tik išlaiko stabilų sprendimų efektyvumą, bet ir sugeba toliau optimizuoti veiksmų politiką didinant gaunamą atlygį. Skirtingai nei DQN metodas, kuriam būdinga didelė atlygio reikšmių dispersija ir epizodinis nepastovumas, PPO demonstruoja geresnį ilgalaikio atlygio kaupimą net ir veikiant neapibrėžtumo sąlygomis. Tyrinėjimo etape įmonė efektyviai sujungia tyrinėjimo ir išnaudojimo komponentus, palaipsniui formuodama didesnį atlygį generuojančia veikimo politiką.




**35 pav**. Aptarnautų klientų vidurkis tyrinėjimo etape naudojant PPO metodą

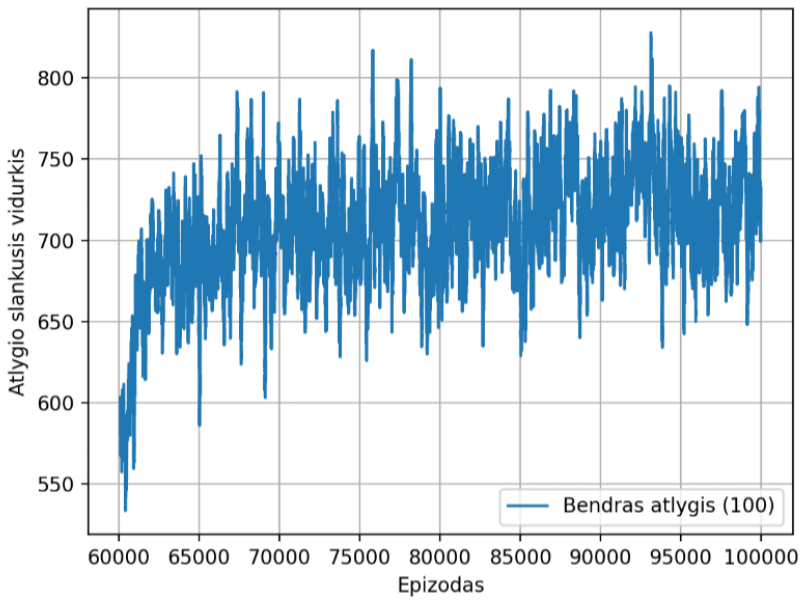
Nuo 27500 epizodo tyrinėjimo etape VIP klientų neaptarnavimo vidurkis sumažėja nuo 2,5 iki mažiau nei 1, tai rodo nuoseklų PPO politikos prisitaikymą įtraukiant VIP klientų aptarnavimą kaip strateginį prioritetą. Nuo 40000 epizodų pastebimas reikšmingas tendencijos pokytis, po kurio neaptarnavimo vidurkis pradeda sistemingai mažėti. Kadangi VIP klientų ignoravimas ankstesniame etape buvo

vienas pagrindinių atlygio funkcijos augimą ribojančių veiksnių, jų įtraukimas tiesiogiai prisideda prie bendro atlygio didėjimo.




**36 pav**. Rinkos kliūčių aplinkoje dažnių pasiskirstymas tyrinėjimo etape naudojant PPO metodą

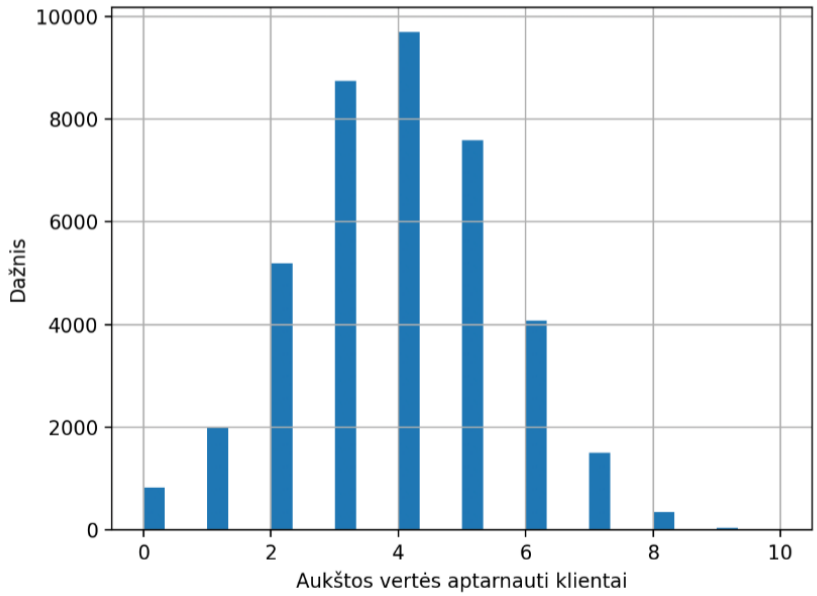
Rinkos kliūčių dažnių pasiskirstymo forma rodo, kad tyrinėjimo etape sumažėja epizodų, kuriuose nepasirodo kliūčių, tai tiesiogiai susiję su tuo, jog įmonė aktyviau įtraukia VIP klientų aptarnavimą į savo strategiją. Tačiau tuo pačiu atsiranda daugiau epizodų su didesniu nei vidutiniu kliūčių skaičiumi, o tai rodo, kad didėjant VIP klientų skaičiui, ne visais atvejais įmonė sugeba juos aptarnauti optimaliai.




**37 pav.** Atlygio funkcijos slankusis vidurkis taikymo etape naudojant PPO metodą

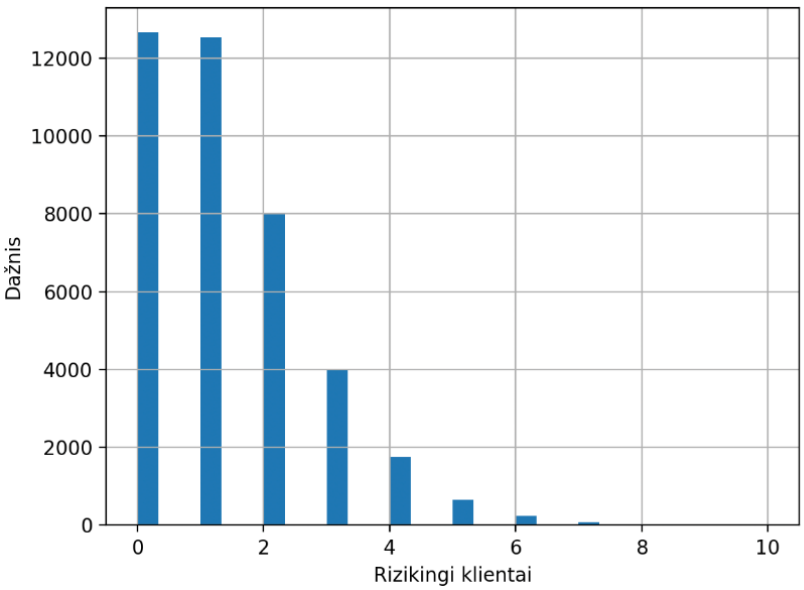
Taikymo etape epizodiniai šuoliai išlieka, tačiau bendra atlygio augimo tendencija sumažėja. Vidutinė atlygio reikšmė padidėja daugiau nei 300 atlygio reikšme, lyginant su tyrinėjimo etapu, kas rodo, jog PPO sėkmingai perkelia išmoktą strategiją iš tyrinėjimo į taikymo aplinką. Įmonė geba

išlaikyti efektyvią politiką veikiant dinamiškoje aplinkoje, kartu užtikrinant nuoseklią sprendimų kokybę.




**38 pav.** VIP klientų aptarnavimo dažnių pasiskirstymas taikymo etape naudojant PPO metodą

Taikymo etape VIP klientų aptarnavimas tampa dominuojančiu įmonės veikimo politikos komponentu. Dažniausiai fiksuojama 4 aukštos vertės klientų aptarnavimo reikšmė rodo, kad įmonės strategija sistemingai orientuota į šio klientų kategorijos užsakymų vykdymą. Toks pasiskirstymas aiškiai skiriasi nuo mokymo ir tyrinėjimo etapų rezultatų, kur VIP klientų aptarnavimas buvo epizodinis ir nepastovus.




**39 pav.** Aptarnautų rizikingų klientų dažnių lentelė taikymo etape naudojant PPO metodą

Rizikingų klientų aptarnavimo pasiskirstymas taikymo etape rodo aiškų dažnio mažėjimą, viršijus 4 aptarnaujamų klientų ribą per epizodą. Dauguma epizodų pasižymi mažiau nei 2 rizikingų užsakymų vykdymu, kas rodo, kad įmonė taiko strategiją, orientuotą į tikėtinos naudos ir rizikos santykį.

**10 lentelė.** Rizikingų užsakymų aptarnavimo aplinkos PPO rodiklių statistika taikymo etape

| **Rodiklis** | **Vidurkis** | **Dispersija** | **75% kvartilis** | **Maksimumas** |
|---|---|---|---|---|
| Aptarnauti klientai | 13,4 | 4,87 | 17 | 34 |
| VIP aptarnauti klientai | 3,8 | 1,6 | 5 | 10 |
| VIP neaptarnauti klientai | 0,07 | 0,3 | 0 | 5 |
| Atlygis už efektyvų judėjimą | 36,9 | 15,4 | 47 | 99 |
| Atlygis už neefektyvų judėjimą | -2,68 | -1,28 | -3,46 | -7,4 |
| Aptarnauti rizikingi klientai | 1,33 | 1,32 | 2 | 10 |
| Susidūrimas su kliūtimis | 0,002 | 0,05 | 0 | 1 |
| Išnykusios kliūtys | 0,1 | 0,49 | 0 | 10 |

Vidutiniškai vieno epizodo metu aptarnaujama 13,4 klientai, o papildomai 3,8 VIP kategorijos klientai. Tai rodo, kad įmonė sistemingai įtraukia VIP klientų aptarnavimą į savo veikimo strategiją. Vidutinis atlygis už efektyvų judėjimą (36,9) ir minimalios baudos už neefektyvius veiksmus (-2,68) patvirtina, jog įmonė ne tik optimizuoja judėjimą aplinkoje, bet ir efektyviai mažina veiklos nuostolingumą.

Rizikingų klientų aptarnavimo vidurkis (1,33) leidžia daryti išvadą, kad įmonė geba įtraukti riziką keliančius veiksnius į sprendimų priėmimą, tačiau tai daro remiantis apskaičiuota tikėtina nauda. Žemos susidūrimų su kliūtimis ir išnykusių kliūčių reikšmės rodo, kad įmonė sėkmingai kontroliuoja aplinkos trikdžius ir veikia pagal politiką, užtikrinančią veikimo stabilumą.


**40 pav.** Rizikingų užsakymų aptarnavimo aplinkos sužaidybinimas

Pateikiamas vieno epizodo sužaidybinimo vaizdas rizikingų užsakymų aptarnavimo aplinkoje esant rinkos kliūtims, taikant PPO metodą. Skirtingai nei VIP aplinkoje čia papildomai aplinkoje egzistuoja pilka spalva pažymėti rizikingi užsakymai.

**11 lentelė.** Bendrojo atlygio statistika rizikingų užsakymų aptarnavimo aplinkoje

| **Rodiklis** | **Mokymo etapas** | | **Tyrinėjimo etapas** | | **Taikymo etapas** | |
|---|---|---|---|---|---|---|
| | **DQN** | **PPO** | **DQN** | **PPO** | **DQN** | **PPO** |
| Vidurkis | 253,5 | 273,1 | 280,2 | 418,7 | 288,9 | 708,9 |
| Dispersija | 172,7 | 184,1 | 172,4 | 215,6 | 173 | 285,1 |
| Mediana | 235,15 | 257,6 | 264,7 | 404,6 | 275,45 | 725,8 |
| 75% kvartilis | 363,3 | 390 | 389 | 556,9 | 397,7 | 912,6 |
| Maksimumas | 1081,6 | 1136,1 | 1235,5 | 1585,5 | 1280,9 | 1715,5 |

Bendrojo atlygio statistika rodo, kad PPO metodas pranoksta DQN visuose rizikingų užsakymų aptarnavimo aplinkos etapuose. Mokymo etape PPO pasiekia didesnį vidutinį atlygį 273,1 ir aukštesnė maksimalią reikšmę 1136,1, kas rodo geresnį ankstyvą pelningų sprendimų identifikavimą. Tyrinėjimo etape šis skirtumas išryškėja dar labiau, PPO metodo 75 % kvartilio reikšmė siekia 556,9 bendrojo atlygio, kas yra 42,79 % padidėjusi reikšmė, tuo tarpu DQN metodo reikšmė nuo mokymo etapo pasikeičia tik 7,07 %. Taikymo etape PPO metodas pasiekia vidutinį atlygį 2,45 karto viršijantį DQN rezultatą, o mediana (725,8) rodo, kad net tipiškas PPO epizodas yra aukštesnės vertės nei DQN 75 % kvartilio reikšmė (397,7). Šie rezultatai leidžia daryti išvadą, kad PPO metodas pasižymi geresniu mokymo stabilumu ir sprendimų kokybe, sudėtingoje verslo procesų aplinkoje.

## Išvados

1. Literatūros analizė parodė, kad skatinamojo mokymosi metodai, sudaro teorinį pagrindą optimalių sprendimų paieškai kintančiose ir neapibrėžtose verslo aplinkose. Sintetinių duomenų generavimas, naudojant imitacines sistemas, leidžia efektyviai spręsti empirinių duomenų trūkumo ir eksperimentavimo rizikos problemas. o žaidybinimas padeda įtraukti dinamiškos aplinkos elementus, būdingus realioms verslo situacijoms.
2. Tyrimo metu sukurtas verslo procesų imitavimo modelis, grįstas skatinamojo mokymosi metodais, kuriame realizuotos dvi imitacinės aplinkos, atitinkančios aukštos vertės klientų, rizikingų užsakymų ir išorinių trukdžių scenarijų valdymu. Taikant DQN ir PPO metodus, nustatyta, kad PPO metodas, grindžiamas politikos gradientų optimizavimu, užtikrino aukštesnį bendrą atlygį, stabilesnį klientų aptarnavimo procesą ir mažesnį klaidų skaičių, palyginti su DQN metodu.
3. PPO metodas imituojamoje aukštos vertės klientų aplinkoje pasiekė 2,41 efektyvumo tankį VIP klientų aptarnavime, kai DQN metodo rezultatas siekė 0,73, o rizikingų užsakymų aplinkoje vidutinis PPO taikymo etapo vidutinis atlygis viršijo DQN rezultatą 2,45 kartų, todėl PPO metodas laikomas efektyvesniu sprendžiant sužaidybintų verslo procesų optimizavimo uždavinį.
4. Eksperimentinių rezultatų tikslumas užtikrintas vykdant ilgalaikį mokymą iki 100000 epizodų, kurio metu fiksuotas stabilus atlygio vidurkio konvergavimas ir mažėjantis įmonės politikos jautrumas aplinkos kliūtims. PPO metodas pasižymėjo nuosekliu veikimo stabilumu, tuo tarpu DQN metodas išliko jautrus lokaliems ekstremumams.

## Rekomendacijos

1. Tyrimas parodė, kad agentas geba prisitaikyti prie rinkos svyravimų, tačiau dabartinis modelis pagrįstas fiksuotu veiksmų ir būsenų apibrėžimu, kuris neatspindi realaus verslo ribojimų, tokių kaip išteklių kiekiai. Rekomenduojama į modelį integruoti papildomas dimensijas, kurios apibrėžtų išteklių ribotumą ir jų paskirstymo strategijas, tai leistų imituoti realistiškesnius sprendimų scenarijus ir didinti modelio praktinį taikomumą verslo strategijų planavime
2. PPO agento architektūroje *MlpPolicy* pakeisti į *CnnPolicy*, kad agentas galėtų mokytis iš dvimačio aplinkos vaizdo ir efektyviau atpažintų aplinkos dėsningumus, būdingus sudėtingiems verslo scenarijams.

## Literatūros sąrašas


1. Cassidy, R., Singh, N.S., Schiratti, P.R. et al. (2019). Mathematical modelling for health systems research: a systematic review of system dynamics and agent–based models. BMC Health Services Research, 19, 845. Retrieved from https://doi.org/10.1186/s12913-019-4627-7.
2. Pourghahreman, N., Rajabzadeh Ghatari, A., Moosivand, A. (2018). Agent based simulation of sale and manufacturing agents acting across a pharmaceutical supply chain. Iranian Journal of Pharmaceutical Research, 17(4), 1581–1592. Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC6269582/.
3. Terano, T., Kishimoto, A., Takahashi, T., Yamada, T. (2009). Agent-Based In-Store Simulator for Analyzing Customer Behaviors in a Super-Market. Retrieved from https://doi.org/10.1007/978-3-642-04592-9_31.
4. Liu, S., Li, Y., Triantis, K. P., Xue, H., Wang, Y. (2020). The diffusion of discrete event simulation approaches in health care management in the past four decades: A comprehensive review. MDM Policy & Practice, 5(1). Retrieved from https://doi.org/10.1177/2381468320915242.
5. Williams, E., Szakmany, T., Spernaes, I., Muthuswamy, B., Holborn, P. (2020). Discrete-event simulation modeling of critical care flow: new hospital, old challenges. Critical Care Explorations, 2(9), e0174. Retrieved from 10.1097/CCE.0000000000000174.
6. Zupan, H., Herakovic, N. (2015). Production line balancing with discrete event simulation: a case study. IFAC-PapersOnLine, 48(3), p. 2305-3211. Retrieved from https://doi.org/10.1016/j.ifacol.2015.06.431.
7. Bottani, E., Montanari, R. (2010). Supply chain design and cost analysis through simulation. International Journal of Production Research, 48(10), p. 2859-2886. Retrieved from 10.1080/00207540902960299.
8. Moshood, T. D., Rotimi, J., Shahzad, W., Bamgbade, J. A. (2024). Infrastructure digital twin technology: A new paradigm for future construction industry. Technology in Society, 79, 102519. Retrieved from https://doi.org/10.1016/j.techsoc.2024.102519.
9. Drechsler, J. (2011). Synthetic Datasets for Statistical Disclosure Control: Theory and Implementations. Retrieved from: https://doi.org/10.1007/978-1-4614-0326-5.
10. Deng, J., Sierla, S., Sun, J., Vyatkin, V. (2022). Reinforcement learning for industrial process control: A case study in flatness control in steel industry. Computers in Industry, 143, 103748. Retrieved from https://doi.org/10.1016/j.compind.2022.103748.
11. Koroteev, M., Romanova, E., Korovin, D., et al. (2022). Optimization of Food Industry Production Using the Monte Carlo Simulation Method: A Case Study of a Meat Processing Plant. Retrieved from https://doi.org/10.3390/informatics9010005.
12. Wang, Y., He, H., Sun, C. (2018). Learning to Navigate Through Complex Dynamic Environment With Modular Deep Reinforcement Learning. Retrieved from 10.1109/TG.2018.2849942.
13. Kevin P. Murphy (2024). Reinforcement Learning: An Overview. Retrieved from: https://doi.org/10.48550/arXiv.2412.05265.
14. Feng, G., Zhong, H. (2023). Rethinking Model-based, Policy-based, and Value-based Reinforcement Learning via the Lens of Representation Complexity. Retrieved from https://doi.org/10.48550/arXiv.2312.17248.

15. Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction. Retrieved from https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf.
16. Kastius, A., Schlosser, R. (2022). Dynamic Pricing under Competition using Reinfrocement Learning. Retrieved from: https://doi.org/10.1057/s41272-021-00285-3.
17. Plaat, A., Kosters, W., Preuss, M. (2023). High-accuracy model-based reinforcement learning. Retrieved from https://doi.org/10.1007/s10462-022-10335-w.
18. Oroojlooyjadid, A., Nazari, M. R., Snyder, L., Takáč. M. (2020). A Deep Q-Network for the Beer Game: A Deep Q–Learning algorithm to Solve Inventory Optimization Problems. Retrieved from https://doi.org/10.48550/arXiv.1708.05924
19. Deng, S., Schiffer, M., Bichler, M. (2025). Exploring competitive and collusive behaviors in algorithmic pricing with deep reinforcement learning. Retrieved from 10.48550/arXiv.2503.11270.
20. Nomura, Y., Liu, Z., Nishi, T. (2024). Deep Reinforcement Learning for Dynamic Pricing and Ordering Policies in Perishable Inventory Management. Retrieved from: https://doi.org/10.3390/app15052421.
21. Rozhkov, M., Alyamovskaya, N., Zakhodiakin, G. (2024). The beer game bullwhip effect mitigation: a deep reinforcement learning approach. Retrieved from https://doi.org/10.1080/00207543.2025.2479831.
22. Sun, J., Huang, G., Sun, G., Yu, H., et al. (2018). A Q-Learning-Based Approach for Deploying Dynamic Service Function Chains. Retrieved from https://doi.org/10.3390/sym10110646.
23. Waschneck, B., Reichstaller, A., Belzner, L. et al. (2018). Optimization of global production scheduling with deep reinforcement learning. Retrieved from https://doi.org/10.1016/j.procir.2018.03.212.
24. Liu, X. Y., Xiong, Z., Zhong, S., Yang, H., Walid, A. (2022). Practical Deep Reinforcement Learning Approach for Stock Trading. Retrieved from https://doi.org/10.48550/arXiv.1811.07522.
25. Pathare, A., Mangrulkar, R., Suvarna, K., Parekh, A., Thakur, G., Gawade, A. (2023). Comparison of tabular synthetic data generation techniques using propensity and cluster log metric. Retrieved from https://doi.org/10.1016/j.jjimei.2023.100177.
26. Cheng-Hsin, Y. P., Eaves, B., Schmidt, M., Swanson, K. (2024). Structured Evaluation of Synthetic Tabular Data. Retrieved from https://doi.org/10.48550/arXiv.2403.10424.
27. Jordon, J., Szpruch, L., Houssiau, M., Bottarelli, et al (2022). Sythetic Data – what, why and how? Retrieved from https://doi.org/10.48550/arXiv.2205.03257.
28. Figueira, A., Vaz, B. (2022). Survey on Synthetic Data Generation, Evaluation Methods, and GANs. Retrieved from https://doi.org/10.3390/math10152733.
29. Giuffre, M., Shung, D. (2023). Harnessing the power of synthetic data in healthcare: innovation, application, and privacy. Retrieved from https://doi.org/10.1038/s41746-023-00927-3.
30. Liu, Q., Shakya, R., Jovanovic, J., Khalil, M. (2025). Ensuring privacy through synthetic data generation in education. Retrieved from https://doi.org/10.1111/bjet.13576.
31. Song, Z., He, Z., Li, X., et al (2023). Synthetic Datasets for Autonomous Driving: A Survey. Retrieved from https://doi.org/10.48550/arXiv.2304.12205.
32. Ammara, D. A., Ding, J., Tutschku, K. (2024). Synthetic Data Generation in Cybersecurity: A Comparative Analysis. Retrieved from https://arxiv.org/html/2410.16326v1.

33. Xia, Y., Arian, A., Narayanamoorthy, S., Mabry, J. (2023). RetailSynth: Synthetic Data Generation for Retail AI Systems Evaluation. Retrieved from https://doi.org/10.48550/arXiv.2312.14095.
34. Assefa, S., Dervovic, D., Mahfouz, M. et al (2020). Generating Synthetic Data in Finance: Opportunities, Challenges and Pitfalls. Retrieved from https://dx.doi.org/10.2139/ssrn.3634235.
35. Buggineni, V., Chen, C., Camelio, J. (2024). Enhancing manufacturing operations with synthetic data: a systematic framework for data generation, accuracy, and utility. Retrieved from https://doi.org/10.3389/fmtec.2024.1320166.
36. Dichev, C., Dicheva, D. (2017). Gamifying education: what is known, what is believed and what remains uncertain: a critical review. Retrieved from https://doi.org/10.1186/s41239-017-0042-5.
37. Schell, J. (2008). The Art of Game Design: A Book of Lenses.
38. Silver, D., Hubert, T., Schrittwieser, J., et al (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. Retrieved from https://doi.org/10.48550/arXiv.1712.01815.
39. Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. Retrieved from https://doi.org/10.1038/nature14236
40. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal Policy Optimization Algorithms. Retrieved from https://doi.org/10.48550/arXiv.1707.06347.
41. Schulman, J., Moritz, P., Levine, S., Jordan, M. I., Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. Retrieved from 10.48550/arXiv.1506.02438
42. Raffin, A., Hill, A., Gleave, A., et al (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. Retrieved from https://jmlr.org/papers/v22/20-1364.html.
43. Liang, E., Liaw, R., Moritz, P., et al (2018). Rlib: Abstractions for Distributed Reinforcement Learning. Retrieved from https://doi.org/10.48550/arXiv.1712.09381.
44. Wikipedia. (2025). Snake (1998 video game). Retrieved from https://en.wikipedia.org/wiki/Snake_(1998_video_game) [žiūrėta 2025-05-21].

## Priedai

### 1 priedas. Gyvatėlės žaidimo aplinkos kodo fragmentas

```
import numpy as np
import random
class Segment():
    def __init__(self, parent, x, y):
        self.parent = parent
        self.x = x
        self.y = y
    def setX(self, x):
        self.x = x
    def setY(self, y):
        self.y = y
    def setParent(self, parent):
        self.parent = parent
    def getCoords(self):
        return (self.x, self.y)
    def getGridIndex(self):
        return (self.y, self.x)
class SnakeAgent():
    def __init__(self, start_x, start_y):
        self.head = Segment(None, start_x, start_y)
        self.tail = self.head
    def advanceBody(self):
        current = self.tail
        while current.parent is not None:
            parent_pos = current.parent.getCoords()
            current.setX(parent_pos[0])
            current.setY(parent_pos[1])
            current = current.parent
    def advance(self, direction):
        old_tail = self.tail.getCoords()
        self.advanceBody()
        hx, hy = self.head.getCoords()
        if direction == 0:
            self.head.setY(hy - 1)
        elif direction == 1:
            self.head.setX(hx + 1)
        elif direction == 2:
            self.head.setY(hy + 1)
        elif direction == 3:
            self.head.setX(hx - 1)
        return (*old_tail, *self.head.getCoords())
    def grow(self, x, y):
```

```
        new_head = Segment(None, x, y)
        self.head.setParent(new_head)
        self.head = new_head
    def getHead(self):
        return self.head
    def getTail(self):
        return self.tail
class GridGame():
    def __init__(self, width, height):
        self.head_marker = 2
        self.body_marker = 1
        self.food_marker = 7
        self.width = width
        self.height = height
        self.grid = np.zeros([height, width], dtype=int)
        self.score = 1
        center_x = width // 2
        center_y = height // 2
        self.grid[center_y, center_x] = self.head_marker
        self.snake = SnakeAgent(center_x, center_y)
        self.spawnFood()
        self.updateState()
    def spawnFood(self):
        available = [i for i, val in np.ndenumerate(self.grid) if val == 0]
        self.food_loc = random.choice(available)
        self.grid[self.food_loc] = self.food_marker
    def isValidMove(self, direction):
        new_x, new_y = self.predictMove(direction)
        if not (0 <= new_x < self.width and 0 <= new_y < self.height):
            return False
        if self.grid[new_y, new_x] == self.body_marker:
            return False
        return True
    def predictMove(self, direction):
        x, y = self.snake.getHead().getCoords()
        if direction == 0:
            y -= 1
        elif direction == 1:
            x += 1
        elif direction == 2:
            y += 1
        elif direction == 3:
            x -= 1
        return (x, y)
    def updateState(self):
        self.state = np.zeros(8, dtype=int)
```

```
    for i in range(4):
        self.state[i] = not self.isValidMove(i)
    self.state[4:] = self.foodDirection()
def encodeState(self):
    return sum(2**i * val for i, val in enumerate(self.state))
def foodDirection(self):
    direction_flags = np.zeros(4, dtype=int)
    delta = np.array(self.food_loc) - np.array(self.snake.getHead().getGridIndex())
    if delta[0] < 0:
        direction_flags[0] = 1
    elif delta[0] > 0:
        direction_flags[2] = 1
    if delta[1] > 0:
        direction_flags[1] = 1
    elif delta[1] < 0:
        direction_flags[3] = 1
    return direction_flags
def getRenderMatrix(self):
    canvas = np.zeros([self.width, self.height])
    node = self.snake.getTail()
    fade = 0
    while node:
        fade += 1
        canvas[node.getGridIndex()] = 0.2 + 0.8 * fade / self.score
        node = node.parent
    canvas[self.food_loc] = -1
    return canvas
def render(self):
    horizontal_border = '+' + '-' * self.width + '+'
    print(horizontal_border)
    for i in range(self.height):
        row = '|'
        for j in range(self.width):
            val = self.grid[i, j]
            if val == 0:
                row += '.'
            elif val == self.head_marker:
                row += 'H'
            elif val == self.body_marker:
                row += 'B'
            elif val == self.food_marker:
                row += 'F'
        row += '|'
        print(row)
    print(horizontal_border)
def step(self, action):
```

```
        game_over = False
        if self.isValidMove(action):
            reward = 1 if self.foodDirection()[action] else 0
            head_x, head_y = self.snake.getHead().getCoords()
            self.grid[head_y, head_x] = self.body_marker
            new_x, new_y = self.predictMove(action)
            if self.grid[new_y, new_x] == self.food_marker:
                self.snake.grow(new_x, new_y)
                self.grid[new_y, new_x] = self.head_marker
                self.spawnFood()
                self.score += 1
                reward = 2
            else:
                old_tx, old_ty, new_hx, new_hy = self.snake.advance(action)
                self.grid[old_ty, old_tx] = 0
                self.grid[new_hy, new_hx] = self.head_marker
        else:
            reward = -2
            game_over = True
        self.updateState()
        return (self.encodeState(), reward, game_over, self.score)
if __name__ == "__main__":
    game = GridGame(8, 8)
    game.render()
    print("Score: 1")
    while True:
        move = input("Move (w/a/s/d or q): ")
        if move == 'w':
            state, reward, over, score = game.step(0)
        elif move == 'a':
            state, reward, over, score = game.step(3)
        elif move == 's':
            state, reward, over, score = game.step(2)
        elif move == 'd':
            state, reward, over, score = game.step(1)
        elif move == 'q':
            break
        else:
            continue
        if over:
            print("Game Over. Final Score:", score)
            break
        else:
            game.render()
            print("Reward:", reward, "Score:", score)
```

**2 priedas. Q-mokymosi agento kodo fragmentas**

```
import random
from qlearningaplinka import GridGame
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
def evaluateScore(Q, boardDim, numRuns, displayGame=False):
    cutoff = 100
    scores = []
    for _ in range(numRuns):
        game = GridGame(boardDim, boardDim)
        state = game.encodeState()
        score = 0
        oldScore = 0
        gameOver = False
        moveCounter = 0
        while not gameOver:
            action = np.argmax(Q[state, :])
            state, reward, gameOver, score = game.step(action)
            if score == oldScore:
                moveCounter += 1
            else:
                oldScore = score
                moveCounter = 0
            if moveCounter >= cutoff:
                break
        scores.append(score)
    return np.average(scores), scores
boardDim = 16
numStates = 2 ** 8
numActions = 4
Q = np.zeros((numStates, numActions))
gamma = 0.8
epsilon = 0.2
numEpisodes = 10001
Qs = np.zeros([numEpisodes, numStates, numActions])
bestLength = 0
print("Training for", numEpisodes, "games...")
for episode in range(numEpisodes):
    game = GridGame(boardDim, boardDim)
    state = game.encodeState()
    gameOver = False
    score = 0
    while not gameOver:
        if random.uniform(0, 1) < epsilon:
```

```
            action = random.randint(0, 3)
        else:
            action = np.argmax(Q[state, :])
        new_state, reward, gameOver, score = game.step(action)
        Q[state, action] = reward + gamma * np.max(Q[new_state, :])
        state = new_state
    Qs[episode, :, :] = np.copy(Q)
    if episode % 100 == 0:
        averageLength, _ = evaluateScore(Q, boardDim, 25)
        if averageLength > bestLength:
            bestLength = averageLength
            bestQ = np.copy(Q)
        print("Episode", episode, "Average snake length :", averageLength)
print("")
plotEpisodes = [0, 200, 400, 600, 800, 1000, 2500, 5000, 10000]
fig, axes = plt.subplots(3, 3, figsize=(9, 9))
axList, ims, dataArrays, scores, labels = [], [], [], [], []
for i, row in enumerate(axes):
    for j, ax in enumerate(row):
        episode_idx = i * len(row) + j
        ax.set_title(f"Episode {plotEpisodes[episode_idx]}")
        ax.axis('off')
        axList.append(ax)
        ims.append(ax.imshow(np.zeros([boardDim, boardDim]), vmin=-1, vmax=1, cmap='RdGy'))
        labels.append(ax.text(0, 15, "Length: 0", bbox={'facecolor': 'w', 'alpha': 0.75, 'pad': 1,
'edgecolor': 'white'}))
        dataArrays.append([])
        scores.append([])
cutoff = 100
numGames = 10
maxFrames = 1000
for k in range(numGames):
    games, states, gameOvers, moveCounters, oldScores = [], [], [], [], []
    for l in range(len(plotEpisodes)):
        g = GridGame(boardDim, boardDim)
        games.append(g)
        states.append(g.encodeState())
        gameOvers.append(False)
        moveCounters.append(0)
        oldScores.append(0)
    for j in range(maxFrames):
        for i in range(len(plotEpisodes)):
            if gameOvers[i]:
                continue
            Qsnapshot = Qs[plotEpisodes[i], :, :]
            action = np.argmax(Qsnapshot[states[i], :])
```

```
            states[i], reward, gameOver, score = games[i].step(action)
            dataArrays[i].append(games[i].getRenderMatrix())
            scores[i].append(score)
            if score == oldScores[i]:
                moveCounters[i] += 1
            else:
                oldScores[i] = score
                moveCounters[i] = 0
            if moveCounters[i] >= cutoff or gameOver:
                gameOvers[i] = True
        if not any(not over for over in gameOvers):
            print("Game", k, "finished, total moves:", len(dataArrays[0]))
            break
def animate(frameNum):
    for i, im in enumerate(ims):
        if frameNum < len(dataArrays[i]):
            labels[i].set_text(f"Length: {scores[i][frameNum]}")
            ims[i].set_data(dataArrays[i][frameNum])
    return ims + labels
print("")
evaluation_interval = 100
evaluation_scores = []
for i in range(0, numEpisodes, evaluation_interval):
    avg_length, _ = evaluateScore(Qs[i], boardDim, 25)
    evaluation_scores.append(avg_length)
plt.figure(figsize=(10, 4))
plt.plot(range(0, numEpisodes, evaluation_interval), evaluation_scores, marker='o')
plt.title('Vidutinis gyvatės ilgio kitimas treniravimo metu')
plt.xlabel('Epizodas')
plt.ylabel('Vidutinis gyvatės ilgis')
plt.grid(True)
plt.tight_layout()
plt.show()
_, final_scores = evaluateScore(bestQ, boardDim, 100)
plt.figure(figsize=(8, 4))
plt.hist(final_scores, bins=range(min(final_scores), max(final_scores) + 1), edgecolor='black')
plt.title('Epizodų trukmės dažnių pasiskirstymas')
plt.xlabel('Gyvavimo trukmė')
plt.ylabel('Dažnis')
plt.tight_layout()
plt.show()
last_Q = Qs[-1]
action_counts = np.zeros(numActions, dtype=int)
game = GridGame(boardDim, boardDim)
state = game.encodeState()
gameOver = False
```

```
cutoff = 100
moveCounter = 0
oldScore = 0
while not gameOver:
    action = np.argmax(last_Q[state, :])
    action_counts[action] += 1
    state, reward, gameOver, score = game.step(action)
    if score == oldScore:
        moveCounter += 1
    else:
        oldScore = score
        moveCounter = 0
    if moveCounter >= cutoff:
        break
plt.figure(figsize=(6, 4))
plt.bar(['↑', '→', '↓', '←'], action_counts, color='steelblue', edgecolor='black')
plt.title('Pasirinktų veiksmų pasiskirstymas')
plt.xlabel('Veiksmas')
plt.ylabel('Dažnis')
plt.tight_layout()
plt.show()
```

**3 priedas. Imitacinės aplinkos kodo fragmentas**

```
import gymnasium as gym
from gymnasium import spaces
import numpy as np
import random
from enum import IntEnum
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
class Direction(IntEnum):
    UP = 0
    RIGHT = 1
    DOWN = 2
    LEFT = 3
    @staticmethod
    def to_vector(direction):
        return {
            Direction.UP: (0, -1),
            Direction.RIGHT: (1, 0),
            Direction.DOWN: (0, 1),
            Direction.LEFT: (-1, 0),
        }[direction]
    @staticmethod
    def is_opposite(dir1, dir2):
        return abs(dir1 - dir2) == 2
```

```
class Snake:
    def __init__(self, x, y):
        self.positions = [(x, y)]
        self.last_direction = Direction.RIGHT
    def get_head(self):
        return self.positions[0]
    def get_positions(self):
        return self.positions
    def move(self, direction):
        if Direction.is_opposite(direction, self.last_direction):
            direction = self.last_direction
        dx, dy = Direction.to_vector(direction)
        head_x, head_y = self.get_head()
        new_head = (head_x + dx, head_y + dy)
        self.positions = [new_head] + self.positions[:-1]
        self.last_direction = direction
        return self.positions[-1]
    def grow(self, x, y):
        self.positions = [(x, y)] + self.positions
class SnakeEnv(gym.Env):
    metadata = {"render_modes": ["human"], "render_fps": 10}
    def __init__(self, width=10, height=10, max_steps_per_episode=400, render_mode=None):
        super().__init__()
        self.width = width
        self.height = height
        self.max_steps = max_steps_per_episode
        self.render_mode = render_mode
        self.action_space = spaces.Discrete(4)
        self.observation_space = spaces.Box(low=-1, high=1, shape=(14,), dtype=np.float32)
        self.vip_food_pos = None
        self.vip_food_timer = 0
        self.vip_spawn_trigger = 3
        self.vip_collected_counter = 0
        self.vip_food_val = 9
        self.head_val = 2
        self.body_val = 1
        self.food_val = 7
        self.reset()
    def reset(self, *, seed=None, options=None):
        super().reset(seed=seed)
        self.board = np.zeros((self.height, self.width), dtype=np.uint8)
        start_x, start_y = self.width // 2, self.height // 2
        self.snake = Snake(start_x, start_y)
        x, y = self.snake.get_head()
        self.board[y, x] = self.head_val
        self.steps = 0
```

```
        self.done = False
        self.steps_since_last_apple = 0
        self.collected_clients = 0
        self.cumulative_distance_reward = 0.0
        self.cumulative_step_penalty = 0.0
        self.total_revenue = 0.0
        self.vip_collected_counter = 0
        self.vip_clients_served = 0
        self.vip_failed = 0
        self._spawn_food()
        self.vip_food_pos = None
        self.vip_food_timer = 0
        return self._get_obs(), {}
    def _spawn_food(self):
        empty = [(x, y) for y in range(self.height) for x in range(self.width) if self.board[y, x] == 0]
        self.food_pos = random.choice(empty)
        self.board[self.food_pos[1], self.food_pos[0]] = self.food_val
    def _spawn_vip_food(self):
        empty = [(x, y) for y in range(self.height) for x in range(self.width) if self.board[y, x] == 0]
        if not empty:
            return
        self.vip_food_pos = random.choice(empty)
        self.board[self.vip_food_pos[1], self.vip_food_pos[0]] = self.vip_food_val
        self.vip_food_timer = 0
    def _valid(self, x, y):
        if not (0 <= x < self.width and 0 <= y < self.height):
            return False
        head = self.snake.get_head()
        body = self.snake.get_positions()[1:]
        target = (x, y)
        if target in body:
            return False
        if target == self.food_pos:
            return True
        if self.vip_food_pos is not None and target == self.vip_food_pos:
            return True
        return True
    def _get_obs(self):
        head_x, head_y = self.snake.get_head()
        food_x, food_y = self.food_pos
        dx = (food_x - head_x) / self.width
        dy = (food_y - head_y) / self.height
        if self.vip_food_pos:
            vip_dx = (self.vip_food_pos[0] - head_x) / self.width
            vip_dy = (self.vip_food_pos[1] - head_y) / self.height
        else:
```

```
            vip_dx, vip_dy = 0.0, 0.0
        blocked = [
            not self._valid(head_x, head_y - 1),
            not self._valid(head_x + 1, head_y),
            not self._valid(head_x, head_y + 1),
            not self._valid(head_x - 1, head_y),
        ]
        direction = [int(self.snake.last_direction == d) for d in Direction]
        length_ratio = len(self.snake.get_positions()) / (self.width * self.height)
        vip_active = 1.0 if self.vip_food_pos else 0.0
        return np.array([dx, dy, vip_dx, vip_dy] + blocked + direction + [length_ratio,
vip_active],dtype=np.float32)
    def step(self, action):
        action = int(action)
        if Direction.is_opposite(action, self.snake.last_direction):
            action = self.snake.last_direction
        self.steps += 1
        self.steps_since_last_apple += 1
        for x, y in self.snake.get_positions():
            self.board[y, x] = 0
        head_x, head_y = self.snake.get_head()
        dx, dy = Direction.to_vector(action)
        new_x, new_y = head_x + dx, head_y + dy
        if not self._valid(new_x, new_y):
            self.done = True
            return self._get_obs(), -10.0, True, False, self._get_info()
        reward = 0.0
        got_food = (new_x, new_y) == self.food_pos
        got_vip = self.vip_food_pos is not None and (new_x, new_y) == self.vip_food_pos
        if got_food:
            self.snake.grow(new_x, new_y)
            self.collected_clients += 1
            self._spawn_food()
            self.steps_since_last_apple = 0
            base_reward = 10.0
            length_bonus = len(self.snake.get_positions()) / (self.width + self.height)
            efficiency_bonus = max(0, (1.0 - self.steps_since_last_apple / 100)) * 5.0
            reward += base_reward + length_bonus + efficiency_bonus
            self.total_revenue += reward
            self.vip_collected_counter += 1
            if self.vip_collected_counter >= self.vip_spawn_trigger:
                self._spawn_vip_food()
                self.vip_collected_counter = 0
        elif got_vip:
            self.snake.grow(new_x, new_y)
            efficiency_bonus = max(0, (1.0 - self.vip_food_timer / 30)) * 50.0
```

```
            reward += 100.0 + efficiency_bonus
            self.total_revenue += reward
            self.vip_clients_served += 1
            self.vip_food_pos = None
            self.vip_food_timer = 0
        else:
            old_tail = self.snake.move(action)
            old_dist = abs(head_x - self.food_pos[0]) + abs(head_y - self.food_pos[1])
            new_dist = abs(new_x - self.food_pos[0]) + abs(new_y - self.food_pos[1])
            distance_delta = old_dist - new_dist
            distance_reward = distance_delta * 0.5
            idle_penalty = -0.02
            reward += distance_reward + idle_penalty
            self.cumulative_distance_reward += distance_reward
            self.cumulative_step_penalty += abs(idle_penalty)
        for i, (x, y) in enumerate(self.snake.get_positions()):
            self.board[y, x] = self.head_val if i == 0 else self.body_val
        if self.vip_food_pos:
            self.vip_food_timer += 1
            if self.vip_food_timer > 30:
                self.board[self.vip_food_pos[1], self.vip_food_pos[0]] = 0
                self.vip_food_pos = None
                self.vip_food_timer = 0
                self.total_revenue -= 10.0
                self.vip_failed += 1
        if self.steps_since_last_apple >= 100:
            self.done = True
        self.done = self.done or self.steps >= self.max_steps
        return self._get_obs(), reward, self.done, self.done, self._get_info()
    def _get_info(self):
        return {
            "clients_served": self.collected_clients,
            "distance_reward_total": self.cumulative_distance_reward,
            "step_penalty_total": self.cumulative_step_penalty,
            "total_revenue": self.total_revenue,
            "vip_clients_served": self.vip_clients_served,
            "vip_clients_failed": self.vip_failed
        }
    def render(self):
        if self.render_mode != "human":
            return
        if not hasattr(self, 'fig'):
            self.fig, self.ax = plt.subplots()
            self.im = self.ax.imshow(self.board, cmap=self._get_cmap(), vmin=0, vmax=9)
            self.ax.set_title("Business Snake Simulation")
            self.ax.axis('off')
```

```
            plt.ion()
            plt.show()
        else:
            self.im.set_data(self.board)
            self.fig.canvas.draw()
            self.fig.canvas.flush_events()
    def _get_cmap(self):
        colors = {
            0: "white",
            1: "green",
            2: "darkgreen",
            7: "red",
            9: "gold"
        }
        color_list = [colors.get(i, "black") for i in range(10)]
        return mcolors.ListedColormap(color_list)
    def close(self):
        pass
```

**4 priedas. DQN metodo kodo fragmentas**

```
import os
import time
import pandas as pd
from stable_baselines3 import DQN
from stable_baselines3.common.vec_env import DummyVecEnv, VecMonitor
from stable_baselines3.common.callbacks import BaseCallback
from torch.utils.tensorboard import SummaryWriter
from VIPkliutysAplinka import SnakeEnv
class EpisodeLoggingCallback(BaseCallback):
    def __init__(self, csv_path, writer, save_every_episodes=100, verbose=0):
        super().__init__(verbose)
        self.csv_path = csv_path
        self.writer = writer
        self.logged_data = []
        self.save_every_episodes = save_every_episodes
        self.episode_counter = 0
    def _on_step(self) -> bool:
        # Ensure compatibility with both vectorized and non-vectorized environments
        infos = [self.locals["infos"]] if isinstance(self.locals["infos"], dict) else self.locals["infos"]
        dones = [self.locals["dones"]] if isinstance(self.locals["dones"], bool) else self.locals["dones"]
        step = self.num_timesteps
        for info, done in zip(infos, dones):
            if done:
                episode_data = {
                    "step": step,
                    "reward_sum": info.get("reward_sum", 0.0),
```

```
                "clients_served": info.get("clients_served", 0),
                "total_revenue": info.get("total_revenue", 0.0),
                "vip_clients_served": info.get("vip_clients_served", 0),
                "vip_clients_failed": info.get("vip_clients_failed", 0),
                "distance_reward_total": info.get("distance_reward_total", 0.0),
                "step_penalty_total": info.get("step_penalty_total", 0.0),
                "bad_food_collected": info.get("bad_food_collected", 0),
                "obstacle_collisions": info.get("obstacle_collisions", 0),
                "obstacles_spawned": info.get("obstacles_spawned", 0),
                "obstacles_expired": info.get("obstacles_expired", 0)
            }
            self.logged_data.append(episode_data)
            self.episode_counter += 1
            for k, v in episode_data.items():
                if k != "step":
                    self.writer.add_scalar(f"ep/{k}", v, self.episode_counter)
            if self.episode_counter % self.save_every_episodes == 0:
                pd.DataFrame(self.logged_data).to_csv(self.csv_path, index=False)
        return True
    def _on_training_end(self) -> None:
        if self.writer:
            self.writer.flush()
            self.writer.close()
        if self.csv_path:
            pd.DataFrame(self.logged_data).to_csv(self.csv_path, index=False)
            print(f": {self.csv_path}")
timestamp = str(int(time.time()))
models_dir = f"dqn_models/{timestamp}"
logdir = f"dqn_logs/{timestamp}"
csv_path = f"{logdir}/episode_data.csv"
os.makedirs(models_dir, exist_ok=True)
os.makedirs(logdir, exist_ok=True)
#env = DummyVecEnv([lambda: SnakeEnv()])
#env = VecMonitor(env)
env = SnakeEnv()
model = DQN(
    "MlpPolicy",
    env,
    verbose=1,
    tensorboard_log=logdir,
    learning_rate=1e-4,
    buffer_size=50000,
    learning_starts=1000,
    batch_size=32,
    tau=1.0,
    gamma=0.99,
```

```
    train_freq=4,
    target_update_interval=1000,
    exploration_fraction=0.1,
    exploration_final_eps=0.01
)
writer = SummaryWriter(logdir)
callback = EpisodeLoggingCallback(csv_path=csv_path, writer=writer, save_every_episodes=100)
TIMESTEPS = 10000
iters = 0
while True:
    iters += 1
    model.learn(
        total_timesteps=TIMESTEPS,
        reset_num_timesteps=False,
        tb_log_name="DQN",
        callback=callback
    )
    model.save(f"{models_dir}/{TIMESTEPS * iters}")
```

**5 priedas. PPO metodo kodo fragmentas**

```
import os
import time
import pandas as pd
from stable_baselines3 import PPO
from stable_baselines3.common.vec_env import DummyVecEnv, VecMonitor
from stable_baselines3.common.callbacks import BaseCallback
from torch.utils.tensorboard import SummaryWriter
from VIPkliutysAplinka import SnakeEnv
class EpisodeLoggingCallback(BaseCallback):
    def __init__(self, csv_path, writer, save_every_episodes=100, verbose=0):
        super().__init__(verbose)
        self.csv_path = csv_path
        self.writer = writer
        self.logged_data = []
        self.save_every_episodes = save_every_episodes
        self.episode_counter = 0
    def _on_step(self) -> bool:
        infos = self.locals["infos"]
        dones = self.locals["dones"]
        step = self.num_timesteps
        for info, done in zip(infos, dones):
            if done:
                episode_data = {
                    "step": step,
                    "reward_sum": info.get("reward_sum", 0.0),
                    "clients_served": info.get("clients_served", 0),
```

```
                "total_revenue": info.get("total_revenue", 0.0),
                "vip_clients_served": info.get("vip_clients_served", 0),
                "vip_clients_failed": info.get("vip_clients_failed", 0),
                "distance_reward_total": info.get("distance_reward_total", 0.0),
                "step_penalty_total": info.get("step_penalty_total", 0.0),
                "bad_food_collected": info.get("bad_food_collected", 0),
                "obstacle_collisions": info.get("obstacle_collisions", 0),
                "obstacles_spawned": info.get("obstacles_spawned", 0),
                "obstacles_expired": info.get("obstacles_expired", 0)
            }
            self.logged_data.append(episode_data)
            self.episode_counter += 1
            for k, v in episode_data.items():
                if k != "step":
                    self.writer.add_scalar(f"ep/{k}", v, self.episode_counter)
            if self.episode_counter % self.save_every_episodes == 0:
                pd.DataFrame(self.logged_data).to_csv(self.csv_path, index=False)
        return True
    def _on_training_end(self) -> None:
        if self.writer:
            self.writer.flush()
            self.writer.close()
        if self.csv_path:
            pd.DataFrame(self.logged_data).to_csv(self.csv_path, index=False)
            print(f": {self.csv_path}")
timestamp = str(int(time.time()))
models_dir = f"ppo_models/{timestamp}"
logdir = f"ppo_logs/{timestamp}"
csv_path = f"{logdir}/episode_data.csv"
os.makedirs(models_dir, exist_ok=True)
os.makedirs(logdir, exist_ok=True)
env = DummyVecEnv([lambda: SnakeEnv()])
env = VecMonitor(env)
model = PPO(
    "MlpPolicy",
    env,
    verbose=1,
    tensorboard_log=logdir,
    n_steps=2048,
    batch_size=64,
    gae_lambda=0.95,
    gamma=0.99,
    n_epochs=10,
    ent_coef=0.01
)
writer = SummaryWriter(logdir)
```

```
callback = EpisodeLoggingCallback(csv_path=csv_path, writer=writer, save_every_episodes=100)
TIMESTEPS = 10000
iters = 0
while True:
    iters += 1
    model.learn(
        total_timesteps=TIMESTEPS,
        reset_num_timesteps=False,
        tb_log_name="PPO",
        callback=callback
    )
    model.save(f"{models_dir}/{TIMESTEPS * iters}")
```

## 6 priedas. Verslo procesų imitavimo kodo fragmentas

```
import gymnasium as gym
from gymnasium import spaces
import numpy as np
import random
from enum import IntEnum
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
class Direction(IntEnum):
    UP = 0
    RIGHT = 1
    DOWN = 2
    LEFT = 3
    @staticmethod
    def to_vector(direction):
        return {
            Direction.UP: (0, -1),
            Direction.RIGHT: (1, 0),
            Direction.DOWN: (0, 1),
            Direction.LEFT: (-1, 0),
        }[direction]
    @staticmethod
    def is_opposite(dir1, dir2):
        return abs(dir1 - dir2) == 2
class Snake:
    def __init__(self, x, y):
        self.positions = [(x, y)]
        self.last_direction = Direction.RIGHT
    def get_head(self):
        return self.positions[0]
    def get_positions(self):
        return self.positions
```

```
    def move(self, direction):
        if Direction.is_opposite(direction, self.last_direction):
            direction = self.last_direction
        dx, dy = Direction.to_vector(direction)
        head_x, head_y = self.get_head()
        new_head = (head_x + dx, head_y + dy)
        self.positions = [new_head] + self.positions[:-1]
        self.last_direction = direction
        return self.positions[-1]
    def grow(self, x, y):
        self.positions = [(x, y)] + self.positions
class SnakeEnv(gym.Env):
    metadata = {"render_modes": ["human"], "render_fps": 10}
    def __init__(self, width=10, height=10, max_steps_per_episode=400, render_mode=None):
        super().__init__()
        self.width = width
        self.height = height
        self.max_steps = max_steps_per_episode
        self.render_mode = render_mode
        self.action_space = spaces.Discrete(4)
        self.observation_space = spaces.Box(low=-1, high=1, shape=(18,), dtype=np.float32)
        self.vip_food_pos = None
        self.vip_food_timer = 0
        self.vip_spawn_trigger = 3
        self.vip_collected_counter = 0
        self.bad_food_collected = 0
        self.vip_food_val = 9
        self.head_val = 2
        self.body_val = 1
        self.food_val = 7
        self.bad_food_val = 5
        self.bad_food_positions = []
        self.max_bad_food = 3
        self.obstacle_val = 8
        self.obstacles = []
        self.obstacle_lifetime = 30
        self.max_obstacles = 2
        self.obstacles_spawned = 0
        self.obstacles_expired = 0
        self.obstacle_collisions = 0
        self.reset()
    def reset(self, *, seed=None, options=None):
        super().reset(seed=seed)
        self.board = np.zeros((self.height, self.width), dtype=np.uint8)
        start_x, start_y = self.width // 2, self.height // 2
        self.snake = Snake(start_x, start_y)
```

```
        x, y = self.snake.get_head()
        self.board[y, x] = self.head_val
        self.steps = 0
        self.done = False
        self.steps_since_last_apple = 0
        self.collected_clients = 0
        self.cumulative_distance_reward = 0.0
        self.cumulative_step_penalty = 0.0
        self.total_revenue = 0.0
        self.vip_collected_counter = 0
        self.vip_clients_served = 0
        self.vip_failed = 0
        self.bad_food_collected = 0
        self.obstacles_spawned = 0
        self.obstacles_expired = 0
        self.obstacle_collisions = 0
        self._spawn_food()
        self.vip_food_pos = None
        self.vip_food_timer = 0
        self.bad_food_positions = []
        self._spawn_bad_food(count=1)
        self.obstacles = []
        return self._get_obs(), {}
    def _spawn_food(self):
        empty = [(x, y) for y in range(self.height) for x in range(self.width) if self.board[y, x] == 0]
        self.food_pos = random.choice(empty)
        self.board[self.food_pos[1], self.food_pos[0]] = self.food_val
    def _spawn_vip_food(self):
        empty = [(x, y) for y in range(self.height) for x in range(self.width) if self.board[y, x] == 0]
        if not empty:
            return
        self.vip_food_pos = random.choice(empty)
        self.board[self.vip_food_pos[1], self.vip_food_pos[0]] = self.vip_food_val
        self.vip_food_timer = 0
    def _spawn_bad_food(self, count=1):
        empty = [(x, y) for y in range(self.height) for x in range(self.width)
                 if self.board[y, x] == 0]
        random.shuffle(empty)
        spawned = 0
        for x, y in empty:
            if spawned >= count or len(self.bad_food_positions) >= self.max_bad_food:
                break
            self.bad_food_positions.append((x, y))
            self.board[y, x] = self.bad_food_val
            spawned += 1
    def _spawn_obstacles(self, count):
```

```
        empty = [(x, y) for y in range(self.height) for x in range(self.width)
                 if self.board[y, x] == 0 and (x, y) not in self.snake.get_positions()]
        random.shuffle(empty)
        added = 0
        for (x, y) in empty:
            if len(self.obstacles) >= self.max_obstacles or added >= count:
                break
            self.obstacles.append((x, y, self.obstacle_lifetime))
            self.board[y, x] = self.obstacle_val
            self.obstacles_spawned += 1
            added += 1
    def _valid(self, x, y):
        if not (0 <= x < self.width and 0 <= y < self.height):
            return False
        head = self.snake.get_head()
        body = self.snake.get_positions()[1:]
        target = (x, y)
        if target in body:
            return False
        if target == self.food_pos:
            return True
        if self.vip_food_pos is not None and target == self.vip_food_pos:
            return True
        if target in self.bad_food_positions:
            return True
        if any((x == ox and y == oy) for ox, oy, _ in self.obstacles):
            return False
        return True
    def _get_obs(self):
        head_x, head_y = self.snake.get_head()
        food_x, food_y = self.food_pos
        dx = (food_x - head_x) / self.width
        dy = (food_y - head_y) / self.height
        if self.vip_food_pos:
            vip_dx = (self.vip_food_pos[0] - head_x) / self.width
            vip_dy = (self.vip_food_pos[1] - head_y) / self.height
        else:
            vip_dx, vip_dy = 0.0, 0.0
        blocked = [
            not self._valid(head_x, head_y - 1),
            not self._valid(head_x + 1, head_y),
            not self._valid(head_x, head_y + 1),
            not self._valid(head_x - 1, head_y),
        ]
        direction = [int(self.snake.last_direction == d) for d in Direction]
        length_ratio = len(self.snake.get_positions()) / (self.width * self.height)
```

```
        vip_active = 1.0 if self.vip_food_pos else 0.0
        if self.bad_food_positions:
            closest = min(self.bad_food_positions, key=lambda pos: abs(pos[0] - head_x) + abs(pos[1] - head_y))
            bad_dx = (closest[0] - head_x) / self.width
            bad_dy = (closest[1] - head_y) / self.height
        else:
            bad_dx, bad_dy = 0.0, 0.0
        if self.obstacles:
            closest = min(self.obstacles, key=lambda o: abs(o[0] - head_x) + abs(o[1] - head_y))
            obs_dx = (closest[0] - head_x) / self.width
            obs_dy = (closest[1] - head_y) / self.height
        else:
            obs_dx, obs_dy = 0.0, 0.0
        return np.array([dx, dy, vip_dx, vip_dy, bad_dx, bad_dy, obs_dx, obs_dy] + blocked + direction + [length_ratio, vip_active], dtype=np.float32)
    def step(self, action):
        action = int(action)
        if Direction.is_opposite(action, self.snake.last_direction):
            action = self.snake.last_direction
        self.steps += 1
        self.steps_since_last_apple += 1
        for x, y in self.snake.get_positions():
            self.board[y, x] = 0
        head_x, head_y = self.snake.get_head()
        dx, dy = Direction.to_vector(action)
        new_x, new_y = head_x + dx, head_y + dy
        if any((new_x == ox and new_y == oy) for ox, oy, _ in self.obstacles):
            self.obstacle_collisions += 1
            self.done = True
            return self._get_obs(), -10.0, True, False, self._get_info()
        if not self._valid(new_x, new_y):
            self.done = True
            return self._get_obs(), -10.0, True, False, self._get_info()
        reward = 0.0
        got_food = (new_x, new_y) == self.food_pos
        got_vip = self.vip_food_pos is not None and (new_x, new_y) == self.vip_food_pos
        got_bad = (new_x, new_y) in self.bad_food_positions
        if got_food:
            self.snake.grow(new_x, new_y)
            self.collected_clients += 1
            self._spawn_food()
            self.steps_since_last_apple = 0
            base_reward = 10.0
            length_bonus = len(self.snake.get_positions()) / (self.width + self.height)
            efficiency_bonus = max(0, (1.0 - self.steps_since_last_apple / 100)) * 5.0
```

```
            reward += base_reward + length_bonus + efficiency_bonus
            self.total_revenue += reward
            self.vip_collected_counter += 1
            if self.vip_collected_counter >= self.vip_spawn_trigger:
                self._spawn_vip_food()
                self.vip_collected_counter = 0
        elif got_vip:
            self.snake.grow(new_x, new_y)
            efficiency_bonus = max(0, (1.0 - self.vip_food_timer / 30)) * 50.0
            reward += 100.0 + efficiency_bonus
            self.total_revenue += reward
            self.vip_clients_served += 1
            self.vip_food_pos = None
            self.vip_food_timer = 0
        elif got_bad:
            self.snake.grow(new_x, new_y)
            reward -= 10.0
            self.total_revenue -= 10.0
            self.bad_food_positions.remove((new_x, new_y))
            self._spawn_bad_food()
            self.bad_food_collected += 1
        else:
            old_tail = self.snake.move(action)
            old_dist = abs(head_x - self.food_pos[0]) + abs(head_y - self.food_pos[1])
            new_dist = abs(new_x - self.food_pos[0]) + abs(new_y - self.food_pos[1])
            distance_delta = old_dist - new_dist
            distance_reward = distance_delta * 0.5
            idle_penalty = -0.02
            reward += distance_reward + idle_penalty
            self.cumulative_distance_reward += distance_reward
            self.cumulative_step_penalty += abs(idle_penalty)
        for i, (x, y) in enumerate(self.snake.get_positions()):
            self.board[y, x] = self.head_val if i == 0 else self.body_val
        if self.vip_food_pos:
            self.vip_food_timer += 1
            if self.vip_food_timer > 30:
                self.board[self.vip_food_pos[1], self.vip_food_pos[0]] = 0
                self.vip_food_pos = None
                self.vip_food_timer = 0
                self.total_revenue -= 10.0
                self.vip_failed += 1
                self._spawn_obstacles(count=2)
        for x, y in self.bad_food_positions:
            self.board[y, x] = self.bad_food_val
        new_obstacles = []
        for x, y, life in self.obstacles:
```

```
            life -= 1
            if life > 0:
                new_obstacles.append((x, y, life))
                self.board[y, x] = self.obstacle_val
            else:
                self.board[y, x] = 0
                self.obstacles_expired += 1
        self.obstacles = new_obstacles
        if self.steps_since_last_apple >= 100:
            self.done = True
        self.done = self.done or self.steps >= self.max_steps
        return self._get_obs(), reward, self.done, self.done, self._get_info()
    def _get_info(self):
        return {
            "reward_sum": self.total_revenue,
            "clients_served": self.collected_clients,
            "total_revenue": self.total_revenue,
            "vip_clients_served": self.vip_clients_served,
            "vip_clients_failed": self.vip_failed,
            "bad_food_collected": self.bad_food_collected,
            "distance_reward_total": self.cumulative_distance_reward,
            "step_penalty_total": self.cumulative_step_penalty,
            "obstacles_spawned": self.obstacles_spawned,
            "obstacles_expired": self.obstacles_expired,
            "obstacle_collisions": self.obstacle_collisions
        }
    def render(self):
        if self.render_mode != "human":
            return
        if not hasattr(self, 'fig'):
            self.fig, self.ax = plt.subplots()
            self.im = self.ax.imshow(self.board, cmap=self._get_cmap(), vmin=0, vmax=9)
            self.ax.set_title("Business Snake Simulation")
            self.ax.axis('off')
            plt.ion()
            plt.show()
        else:
            self.im.set_data(self.board)
            self.fig.canvas.draw()
            self.fig.canvas.flush_events()
    def _get_cmap(self):
        colors = {
            0: "white",
            1: "green",
            2: "darkgreen",
            5: "gray",
```

```
            7: "red",
            9: "gold",
            8: "black",
        }
        color_list = [colors.get(i, "black") for i in range(10)]
        return mcolors.ListedColormap(color_list)
    def close(self):
        pass
```

## 7 priedas. PPO ir DQN hiperparametrų reikšmės

Darbo metu naudojami Stable-Baselines3 skatinamojo mokymosi bibliotekos metodų hiperparametrų reikšmės.

| Hiperparametras | Metodas | |
|---|---|---|
| | DQN | PPO |
| learning_starts | 100 | - |
| n_epochs | - | 10 |
| gradient_steps | 1 | - |
| exploration_initial_eps | 1 | - |
| clip_range | - | 0.2 |
| clip_range_vf | - | - |
| normalize_advantage | - | True |
| vf_coef | - | 0.5 |
| max_grad_norm | 10 | 0.5 |
| use_sde | - | False |
| sde_sample_freq | - | -1 |
| target_kl | - | - |
| stats_window_size | 100 | 100 |
| tensorboard_log | - | - |
| policy_kwargs | - | - |
| verbose | 0 | 0 |
| seed | - | - |
| device | auto | auto |
| _init_setup_model | True | True |

## 8 priedas. Žaidybinimo kodo fragmentas

```
import time
from stable_baselines3 import PPO
from VIPKliutysAplinka import SnakeEnv
model_path = "ppo_models/1748094394/140000"
model = PPO.load(model_path)
env = SnakeEnv(render_mode="human")
obs, _ = env.reset()
done = False
while not done:
    action, _ = model.predict(obs, deterministic=True)
```

```
    obs, reward, terminated, truncated, info = env.step(action)
    done = terminated or truncated
    env.render()
    time.sleep(0.1)
env.close()
```

## 9 priedas. Statistinės analizės kodo fragmentas

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
csv_path = "dqn_logs/1748115909/episode_data.csv"
start_episode = 1
end_episode = 100000
gamma = 0.99
rolling_window = 100
df = pd.read_csv(csv_path)
labels = {
    "clients_served": "Aptarnautų klientų skaičius",
    "total_revenue": "Bendras atlygis",
    "vip_clients_served": "Aukštos vertės aptarnauti klientai",
    "vip_clients_failed": "Aukštos vertės neaptarnauti klientai",
    "distance_reward_total": "Atlygis už efektyvų judėjimą",
    "step_penalty_total": "Bauda už neefektyvų judėjimą",
    "move_efficiency": "Judėjimo efektyvumas",
    "bad_food_collected": "Rizikingi klientai",
    "obstacle_collisions": "Sūsidūrimas su Rinkos kliūtimi",
    "obstacles_spawned": "Rinkos kliūtys",
    "obstacles_expired": "Išnykusios rinkos kliūtys",
}
if "total_revenue" in df.columns:
    df["G_t"] = df["total_revenue"] * gamma**0
    rolling_gt = df["G_t"].iloc[start_episode:end_episode].rolling(window=rolling_window).mean()
    plt.plot(range(start_episode, end_episode), rolling_gt, linewidth=2.5, color="blue")
    plt.title("")
    plt.xlabel("Epizodai")
    plt.ylabel("Atlygio slankusis vidurkis")
    plt.grid(True)
    plt.show()
print("=== Statistinė Santrauka ===")
print(df.iloc[start_episode:end_episode].describe().T)
for column in df.columns:
    if column not in {"step", "G_t", "episode","reward_sum"}:
        label = labels.get(column, column)
        sns.kdeplot(df[column].iloc[start_episode:end_episode].dropna(), fill=True)
        plt.title(f"")
```

```
        plt.xlabel(label)
        plt.ylabel("Tankis")
        plt.grid(True)
        plt.show()
for column in df.columns:
    if column not in {"step","G_t", "episode","reward_sum"}:
        label = labels.get(column, column)
        df[column].iloc[start_episode:end_episode].dropna().hist(bins=30)
        plt.title(f"")
        plt.xlabel(label)
        plt.ylabel("Dažnis")
        plt.grid(True)
        plt.show()
for column in df.columns:
    if column not in {"step","G_t", "episode","reward_sum"}:
        label = labels.get(column, column)
        plt.plot(range(start_episode, end_episode), df[column].iloc[start_episode:end_episode])
        plt.title(f"")
        plt.xlabel("Epizodai")
        plt.ylabel(label)
        plt.grid(True)
        plt.show()
if {"vip_clients_served", "vip_clients_failed"}.issubset(df.columns):
    vip_success_ratio = df["vip_clients_served"].iloc[start_episode:end_episode] / (
        df["vip_clients_served"].iloc[start_episode:end_episode] +
        df["vip_clients_failed"].iloc[start_episode:end_episode] + 1e-8)
    plt.plot(range(start_episode, end_episode), vip_success_ratio)
    plt.title("")
    plt.xlabel("Epizodai")
    plt.ylabel("Sėkmė")
    plt.grid(True)
    plt.show()
if {"distance_reward_total", "step_penalty_total"}.issubset(df.columns):
    move_eff = df["distance_reward_total"].iloc[start_episode:end_episode] / (
        df["step_penalty_total"].iloc[start_episode:end_episode] + 1e-8)
    plt.plot(range(start_episode, end_episode), move_eff)
    plt.title("")
    plt.xlabel("Epizodai")
    plt.ylabel("Efektyvumas")
    plt.grid(True)
    plt.show()
for column in df.columns:
    if column not in {"step", "G_t", "episode", "reward_sum"}:
        label = labels.get(column, column)
        rolling = df[column].iloc[start_episode:end_episode].rolling(window=rolling_window).mean()
        plt.plot(range(start_episode, end_episode), rolling, label=f"{label} ({rolling_window})")
```

```
        plt.title(f"")
        plt.xlabel("Epizodas")
        plt.ylabel("Vidurkis")
        plt.legend()
        plt.grid(True)
        plt.show()
for column in df.columns:
    if column not in {"step", "G_t", "episode", "reward_sum", "clients_served", "vip_clients_served",
"vip_clients_failed", "distance_reward_total","step_penalty_total","move_efficiency"}:
        label = labels.get(column, column)
        rolling = df[column].iloc[start_episode:end_episode].rolling(window=rolling_window).mean()
        plt.plot(range(start_episode, end_episode), rolling, label=f"{label} ({rolling_window})")
        plt.title(f"")
        plt.xlabel("Epizodas")
        plt.ylabel("Atlygio slankusis vidurkis")
        plt.legend()
        plt.grid(True)
        plt.show()
```