Kauno technologijos universitetas

Informatikos fakultetas

Dirbtinio intelekto metodų taikymas strateginiams žaidimams

Baigiamasis magistro studijų projektas

Tadas Laurinaitis

Projekto autorius

Prof. Dr. Tomas Blažauskas

Vadovas

Kaunas, 2025

Kauno technologijos universitetas

Informatikos fakultetas

Dirbtinio intelekto metodų taikymas strateginiams žaidimams

Baigiamasis magistro studijų projektas

Programų sistemų inžinerija (6211BX011)

Tadas Laurinaitis

Projekto autorius

Prof. Dr. Tomas Blažauskas

Vadovas

Dr. Šarūnas Packevičius

Recenzentas

Kaunas, 2025

Kauno technologijos universitetas

Informatikos fakultetas

Tadas Laurinaitis

Dirbtinio intelekto metodų taikymas strateginiams žaidimams

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.

Tadas Laurinaitis

Patvirtinta elektroniniu būdu

Kauno technologijos universitetas

Informatikos fakultetas

Baigiamojo magistro projekto užduotis

Projekto tema: Dirbtinio intelekto metodų taikymas strateginiams žaidimams

Reikalavimai ir sąlygos (tikslinti pavadinimą pagal poreikį):

Vadovas / Vadovė: Prof. Dr. Tomas Blažauskas

(vadovo pareigos, vardas, pavardė, parašas) (data)

Laurinaitis Tadas. Dirbtinio intelekto metodų taikymas strateginiams žaidimams. Magistro studijų baigiamasis projektas vadovas Prof. Dr. Tomas Blažauskas; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Programų sistemų inžinerija.

Reikšminiai žodžiai: Dirbtinio intelekto metodas, strateginis žaidimas, procedūrinis žemėlapių generavimas, kelio radimo algoritmas, sprendimų medis, blackboard sistema.

Kaunas, 2025. 84 p.

Santrauka

Šiame darbe aprašytas dirbtinio intelekto metodų taikymas strateginiams kompiuteriniams žaidimams, naudojantis sukurtu žaidimo prototipu. Žaidimo prototipas naudojamas kaip tyrimo pagrindas dirbtinio intelekto metodų veikimui demonstruoti ir įvertinti. Žaidimo prototipo ir tyrimo paskirtis - palengvinti mažesnių kompanijų ir pavienių programuotojų, kuriančių strateginius kompiuterinius žaidimus, darbą, duodant viešai prieinamus dirbtinio intelekto metodų panaudojimo pavyzdžius. Tokiu būdu siekiama sumažinti atotrūkį tarp didelių žaidimų kūrimo korporacijų, turinčių savo viešai neprieinamus dirbtinio intelekto metodus, ir plačiosios visuomenės.

Literatūros apžvalgoje analizuojamas dirbtinio intelekto taikymas kompiuteriniuose žaidimuose, jo istorija, tipai. Taip pat analizuojamos pasirinktos strateginių žaidimų sritys, kuriuose pritaikomi dirbtinio intelekto metodai – procedūrinis žemėlapių generavimas ir kelio radimas. Analizuojami DI metodų, taikomų šiose srityse privalumai ir trūkumai.

Darbo tiriamojoje dalyje pristatomi 3 tyrimai, kurių metu vertinami skirtingi DI metodai: procedūrinis žemėlapių generavimas, kelio radimas ir blackboard sistema. Kiekvienam iš jų suformuluotos hipotezės, nustatyta tyrimo metodika, atlikti palyginimai su alternatyvomis ir aprašyti rezultatai. Naudojantis tyrimų pagrindu, pateikiamos rekomendacijos tolesniam metodų tobulinimui, kur reikia pasiūlomos alternatyvos, patikrinamos hipotezės ir pateikiamos išvados.

Laurinaitis Tadas. Application of artificial intelligence methods for strategy games. Master's Final Degree Project supervisor Prof. Dr. Tomas Blažauskas; Informatics faculty, Kaunas University of Technology.

Study field and area (study field group): Software Engineering.

Keywords: Artificial intelligence method, strategy game, procedural map generation, pathfinding algorithm, decision tree, blackboard system.

Kaunas, 2025. 84 p.

Summary

This paper presents the application of artificial intelligence methods in strategy genre computer video games, using a developed prototype. The game prototype serves as a foundation for AI method demonstration and evaluation. The purpose of the game prototype and research is to assist individual strategy game creators in the game creation process by giving them open-source AI application examples. In doing so, this work aims to reduce the gap between game development corporations, who keep their proprietary AI models private, and the general game developer community.

The literature review analyses AI application in video games, its history and types. It also analyses the selected key areas in strategy games, where AI methods, such as procedural map generation and pathfiding. The advantages and disadvantages of the selected AI methods, used in these areas are also analysed.

The experimental part of this work presents 3 distinct studies that evaluate 3 different AI methods: procedural map generation, pathfinding and blackboard system. For each of them, hypotheses were formulated, reasearch methodology was established, comparisons with alternatives were made, and the outcomes were documented. Based on the results of these studies, this work provides recommendations for further improvement of aforementioned methods, proposes alternatives where necessary, checks the validity of the formulated hypotheses and summarizes key findings.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

DI – Dirbtinis intelektas (angl. AI - Artificial Intelligence);

CI – Skaičiuojamasis intelektas (angl. Computational Intelligence)

RTS – Realaus laiko strateginis žaidimas (angl. Real Time Strategy)

Įvadas

Dirbtinis intelektas – kompiuterių sistemos gebėjimas atlikti kompleksines užduotis, kurios paprastai atliekamos žmonių - planavimas, sprendimų priėmimas ir kūryba [1]. DI padeda sukurti dinamiškesnius, prisitaikančius ir sudėtingesnius strateginius žaidimus. Pasitelkus DI, strateginiai žaidimai sukuria įtraukiančią patirtį kurdami gyvenimiškus situacinius pokyčius žaidimo progrese ir suteikdami jaudulio jausmą [2]. Svarbu prisiminti, jog strateginiai žaidimai, kaip ir dauguma kitų žaidimų žanrų, remiasi nuspėjamumu, nesvarbu, kokie sudėtingi jie gali pasirodyti. Todėl iki šiol dažniausiai strateginiuose žaidimuose naudojami DI metodai yra deterministiniai - susiję su vidinės logikos automatizavimu, kelio radimu, įvairių veikėjų automatizavimu ir žemėlapių generavimu – sudėtingomis, tačiau nuspėjamomis rezultatus duodančiomis žaidimų dalimis [2][3].

Darbo naujumas ir aktualumas

Sparčiai augant kompiuterinių žaidimų, o tuo tarpu, kartu ir strateginių žaidimų rinkai, ir didėjant žaidėjų skaičiui, taip pat didėja ir žaidimų kūrėjų kiekis. Nors didžioji dalis žaidimų kūrėjų yra nepriklausomi, o mažoji dalis dirba žaidimus kuriančiose korporacijose, dauguma populiariausių ir komerciškai sėkmingų žaidimų yra sukurti tos mažosios dalies. Tokia situacija egzistuoja, kadangi korporacijos turi daugiau žmogiškųjų ir finansinių išteklių, kuriuos skiria žaidimų ir DI metodų juose vystymui, žaidimų publikavimui ir matomumo socialinėse medijose didinimui [4]. Korporacijų sukurti DI algoritmai ir žaidimų programinis kodas nėra viešai prieinami plačiajai visuomenei. Nors sumažinti atskirtį tarp nepriklausomų žaidimų kūrėjų ir korporacijų publikavimo ir socialinių medijų srityse būtų labai sunku, ją sumažinti galima žaidimų vystymo srityje. Šio tyrimo metu analizuoti ir įgyvendinti DI metodai yra aktualūs, kadangi jais naudojantis, nepriklausomi žaidimų kūrėjai sutaupys daug laiko, kurį patys turėtų išleisti analizuojant ir įgyvendinant šiuos ir panašius metodus.

Tikslas ir uždaviniai

Tikslas: išanalizuoti, pritaikyti, įvertinti ir arba patobulinti, arba pasiūlyti alternatyvas dirbtinio intelekto metodams, kurie padeda kurti kokybišką žaidimo logiką.

Uždaviniai:

1. Atlikti dirbtinio intelekto metodų, naudojamų strateginiuose kompiuteriniuose žaidimuose, analizę.
2. Pritaikyti ir kur reikia patobulinti dirbtinio intelekto metodus kuriant strateginio kompiuterinio žaidimo prototipą.
3. Atlikti pritaikytų dirbtinio intelekto metodų ir jiems atliktų patobulinimų įvertinimą, ir kur reikia, pasiūlyti alternatyvas.

Dokumento struktūra

Šiame dokumente pateikiama DI metodų, naudojamų strateginių žaidimų kūrime, analizė, taikymo pavyzdžiai ir tyrimas, remiantis sukurtu strateginio žaidimo prototipu. Analitinėje dalyje nagrinėjama DI istorija, tipai, skirtingų metodų privalumai ir trūkumai. Atliekama populiarių strateginių žaidimų analizė. Projektinėje dalyje aprašoma sukurto žaidimo prototipo architektūra, komponentai ir jų tarpusavio sąveika. Tiriamojoje dalyje pateikiami 3 atlikti tryimai, kuriuose nagrinėjamas procedūrinis žemėlapių generavimas, kelio radimo metodai ir blackboard architektūrą naudojanti automatizuota ekonomikos valdymo sistema.

1. Analitinė dalis

1.1. Dirbtinis intelektas kompiuteriniuose žaidimuose

DI kompiuteriniuose žaidimuose sudaro įvairūs metodai, kurie leidžia skirtingoms žaidimo dalims priimti sprendimus ir atlikti tam tikrus veiksmus be tiesioginės žaidėjo kontrolės. Strateginių žaidimų DI dažniausiai apima kelių paieškos metodus, ekonomikos valdymo mechanizmus, taktinio mąstymo replikavimą, prisitaikymą prie žaidėjo veiksmų ir priešininkų elgsenos modeliavimą. Nors išgirdus žodžius „dirbtinis intelektas“ šiame kontekste iškart kyla asociacijos su pažangiomis, mokymusi pagrįstomis sistemomis, dauguma komerciškai sėkmingų žaidimų vis dar paremti deterministiniais metodais, kurie pasižymi stabilumu ir nuspėjamumu. [5]

Dažna klaida daroma painiojant dirbtinio intelekto sąvoką su skaičiuojamuoju intelektu. Nors dirbtinio intelekto ir skaičiuojamojo intelekto tikslas tas pats – pasiekti bendrinį intelektą, t.y. intelektą kuris galėtų atlikti bet kokią užduotį, kurią gali atlikti žmogus, skiriasi būdai, kaip tas tikslas siekiamas. Dirbtinis intelektas paremtas kietojo skaičiavimo metodais - skaičiavimu, kurio pagrindas yra aiški dvejetainė logika, o skaičiuojamasis intelektas paremtas minkštojo skaičiavimo metodais - neuroniniais tinklais, fuzzy logika, genetiniais algoritmais ir kitais nedeterministiniais metodais [6]. Skaičiuojamojo intelekto metodai demonstruoja gerus rezultatus mokslinėse publikacijose ir laboratoriniuose tyrimuose, tačiau jų taikymas yra ribotas realiems žaidėjams orientuotuose strateginiuose žaidimuose, kadangi jie susiduria su rimtais iššūkiais – yra neprognozuojami, sunkiai suderinami ir su savimi atsineša didelius kaštus ir skaičiavimo galios reikalavimus [7].

Dėl šios priežasties, šiame darbe dėmesys sutelkiamas į deterministinius DI metodus, kurie yra pagrindinis pasirinkimas daugumoje strateginių žaidimų, sukurtų individualių kūrėjų ir žaidimus kuriančių korporacijų. Siekiant suprasti DI svarbą, plėtrą ir augimą kompiuteriniuose žaidimuose, svarbu pažvelgti į šios srities vystymosi istoriją. [7]

1.1.1. Dirbtinio intelekto kompiuteriniuose žaidimuose vystymasis

Strateginiai žaidimai yra ideali aplinka DI galimybių tyrinėjimui, kadangi jie suteikia uždarą aplinką su tam tikromis, kiekvienam žaidimui specifinėmis problemomis ir taisyklėmis. Šiose aplinkose gali būti išbandoma ir vertinama plati aibė problemų sprendimų būdų ir technikų, galiausiai pritaikant šiuos būdus ir technikas sprendžiant realaus pasaulio problemas. Pirmasis DI pritaikymas žaidime buvo 1951 metais sukurtame, matematiniame strateginiame žaidime „Nim“ [8], kurio tikslas buvo dviems žaidėjams paeiliui nuiminėti objektus iš dviejų krūvų. Šiame žaidime DI sugebėdavo reguliariai laimėti prieš žmogų. Tais pačiais metais Mančesterio universitete buvo sukurtos dvi senų, kelis amžius egzistuojančių žaidimų – šaškių ir šachmatų, kompiuterinės versijos, kurios taip pat naudojo DI. Ateinančiais dešimtmečiais DI algoritmai naudojami šiuose žaidimuose buvo tobulinami, o šio tobulinimo kulminacija buvo pasiekta 1997 metais, IBM Deep Blue kompiuteriui nugalėjus tuometinį šachmatų „Grandmaster“ titulą laikantį Garry Kasparov [9]. Praeito amžiaus aštuntajame dešimtmetyje DI tapo integralia kompiuterinių žaidimų dizaino dalimi. 1978 metais sukurtas „Space Invaders“ demonstravo vis sunkėjančius lygius su skirtingais judėjimo modeliais, priklausančiais nuo žaidėjo judėjimo, naudojant elementarų DI grįstą judesių modelį [10]. Nuo to laiko DI buvo bandomas pritaikyti daugumos žanrų žaidimams. DI sporto žaidimuose pasirodė praeito amžiaus devintajame dešimtmetyje išleidus tokius žaidimus, kaip „Earl Weaver Baseball“, „Madden Football“ ir „Tony La Russa Baseball“, kuriuose priešininkai bandė atkartoti realių trenerių ir komandų vadovų vadovavimo stilius ir naudojamas taktikas. Vėlesniuose sporto žaidimuose atsirado galimybė sukurti

DI priešininkus naudojant žaidėjo nustatytus kintamuosius. Formalūs DI įrankiai, tokie kaip baigtinių būsenų mašinos, buvo pradėti naudoti praeito amžiaus paskutiniame dešimtmetyje įvairiuose naujuose žaidimų žanruose. DI buvo pradedamas naudoti realaus laiko strateginiuose žaidimuose kelio radimo problemoms spręsti, atlikti realaus laiko sprendimus ir planuoti kompiuterinio priešininko ekonomiką. Pirmųjų strateginių žaidimų DI nebuvo tobulas – pavyzdžiui 1990 Žaidimo „Herzog Zwei“ kelio radimas buvo praktiškai neveiksnus ir naudojo paprastą trijų baigtinių būsenų mašiną padalinių kontrolei, o 1992 žaidimo „Dune II“ DI sukčiaudavo, kad gautų nesąžiningą persvarą prieš žaidėją.

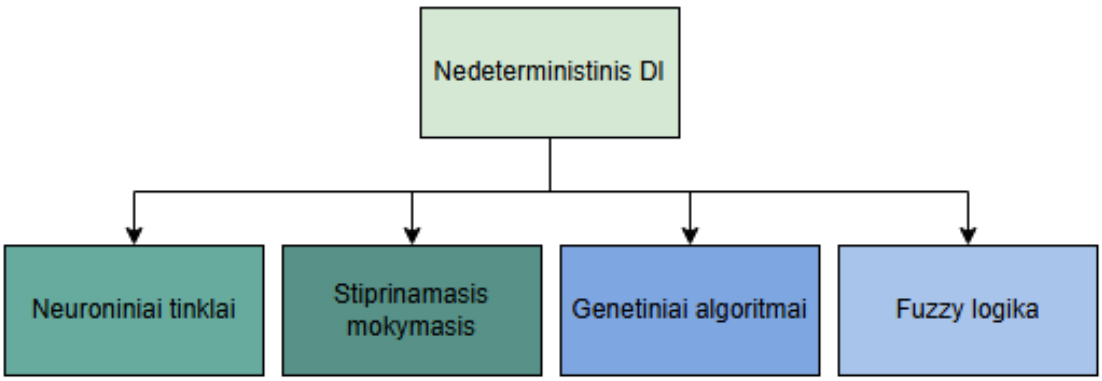
Laikui bėgant, o DI metodams ir algoritmams tobulėjant, didžioji dalis šių problemų buvo išspręsta, o vėlesni žaidimai pradėjo naudoti „bottom-up“ (liet. Iš apačios į viršų) DI metodus, kurie sugeba įvertinti žaidėjo veiksmus ir pagal juos spręsti tolesnius veiksmus. Šiuo metu, kompiuteriniuose žaidimuose, o tuo pačiu ir strateginiuose žaidimuose naudojamas DI pagal veikimo principą yra skirstomas į dvi rūšis – nedeterministinius (angl. Non-Deterministic) ir deterministinius (angl. Deterministic) metodus [11].

1.2. Dirbtinio intelekto tipai

1.2.1. Nedeterministinis dirbtinis intelektas

Nedeterministiniuose metoduose egzistuoja tam tikras neapibrėžtumo laipsnis, kadangi jų elgesys ir veikimas yra nenuspėjamas, o nenuspėjamumo laipsnis priklauso nuo panaudoto DI metodo ir jo supratimo lygio. Šie metodai taip pat gali mokytis ir ekstrapoliuoti patys, skatinti atsirandantį naują elgesį, kuris ne visada atsiranda dėl aiškių nurodymų. Tradiciškai, žaidimų kūrėjai žiūrėdavo į nedeterministinius DI metodus kaip į nepatikimus ir sunkiai suvaldomus, tačiau paskutiniais metais šis požiūris po truputį pradeda keistis. [12]

Panaudojus nedeterministinį metodą jį sunku ištestuoti ir užtikrinti, jog neatsiras nenorima elgsena, todėl esant trumpam žaidimo kūrimo ciklui, sunku garantuoti, kad žaidimas bus pilnai ištestuotas ir atitiks kokybės standartus, kurių tikisi potencialūs žaidėjai. Puikus nedeterministinio metodo pavyzdys būtų kompiuterio valdomo priešininko mokymasis ir prisitaikymas prie žaidėjo kovos taktikų, o tokį prisitaikymą galima būtų pasiekti naudojantis neuroniniu tinklu arba genetiniu algoritmu. [12]



1 pav. Žaidimuose naudojami nedeterministiniai DI metodai [13][14][15][16].

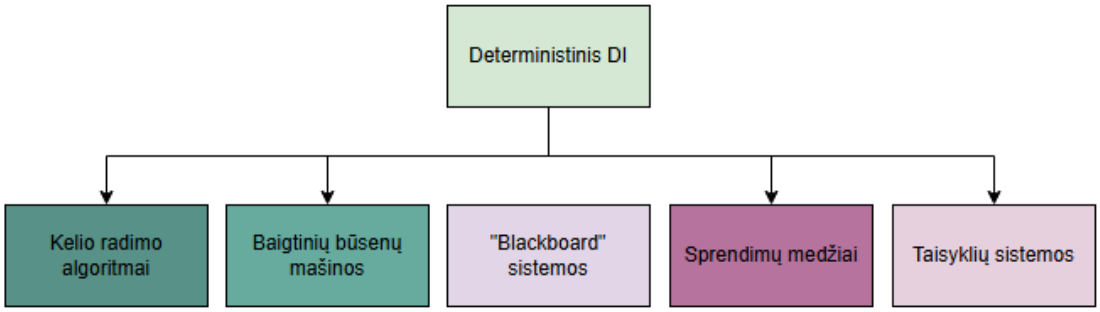
Žaidimuose naudojami nedeterministiniai metodai (žiūrėti **1 pav.**):

– Dirbtiniai neuroniai tinklai (angl. Artificial Neural Networks) – leidžia žaidimo moduliams mokytis iš žaidėjo veiksmų, atpažinti šablonines situacijas ir priimti optimalius sprendimus [13].
– Genetiniai algoritmai (angl. Genetic Algorithms) – evoliucijos proceso imitacija, kur kiekviena algoritmo karta vis labiau prisitaiko prie žaidimo aplinkos ir efektyviau siekia užsibrėžto tikslo, kiekvienoje kartoje pasirenkant geriausiai veiksmus atlikusius egzempliorius [14].
– Fuzzy logika (angl. Fuzzy Logic) – vietoj dvejetainio sprendimų priėmimo – „taip“ ir „ne“, naudojama apytikslių sprendimų vertinimo skalė, pavyzdžiui - „mažai“, „vidutiniškai“, „daug“ [15].
– Stiprinamasis mokymasis (angl. Reinforced Learning) – mokomasi iš patirties, vertinant savo atliktų veiksmų rezultatus [16].

1.2.2. Deterministinis dirbtinis intelektas

Skirtingai nei nedeterministiniuose DI metoduose, deterministinių DI elgesys ir veikimas yra visiškai konkretus ir nuspėjamas – kiekvienas sprendimas ir atliktas veiksmas priklauso nuo konkrečių, iš anksto apibrėžtų sąlygų, todėl nėra jokio neaiškumo [17]. Šio tipo metodai jau kelis dešimtmečius yra kompiuterinių žaidimų pagrindas, o iš jų dažniausiai sudaroma didelė dalis pagrindinės žaidimo logikos. Šie metodai yra greiti, stabilūs, sąlyginai lengvai implementuojami, suprantami ir ištestuojami – visi šie kriterijai yra svarbūs norint sukurti aukštos kokybės subalansuotą strateginį žaidimą [7].

Deterministiniai metodai žaidimo kūrėjui duoda visišką kontrolę. Kūrėjas gali tiksliai suplanuoti žaidimo reakcijas į žaidėjo veiksmus ir padaryti, kad žaidimo tėkmė būtų logiška, o patirtis – vientisa ir sąžininga. Šie metodai taip pat gali būti tobulinami įvedant tam tikrus nedeterministinius elementus, tokius kaip - atsitiktinumas ir nedidelė adaptacija. Nedeterministinių elementų įmaišymas į deterministinius metodus padaro žaidimą dinamiškesnį, mažiau nuspėjamą, tačiau vis tiek suvaldomą ir ištestuojamą [7].



2 pav. Žaidimuose naudojami deterministiniai DI metodai [5][18][19][20][21].

Žaidimuose naudojami deterministiniai metodai (žiūrėti **2 pav.**):
– Kelio radimo algoritmai (angl. Pathfinding Algorithms) – apskaičiuoja kelią nuo taško A iki taško B, atsižvelgia į kliūtis [5].
– Baigtinių būsenų mašinos (angl. Finite State Machines – FSM) – turi iš anksto apibrėžtas būsenas. Naudojami veikėjų elgesiui modeliuoti [18].
– „Blackboard“ sistemos (angl. Blackboard Systems) – apjungianti architektūrinė struktūra, kurios centre yra bendras duomenų modelis, o iš šonų – moduliai (angl. Expert Systems),

kurie su centriniu moduliu dalinasi informacija ir priima sprendimus pagal esančią dabartinę situaciją [19].
- Sprendimų medžiai (angl. Decision Trees) – strukturos, kuriose mazgai atitinka sąlygas, o galutinės šakos – veiksmus [20].
- Taisyklių sistemos (angl. Rule-Based Systems) – paremtos „jei x tada y“ principu [21].

1.3. Žaidimuose nusistovėjęs dirbtinis intelektas

Dažniausiai žaidimuose suteikti DI žmogišką intelektą nėra pagrindinis žaidimų kurėjų tikslas. Dažnu atveju rašomas kodas būna skirtas kontroliuoti personažą, kuris nėra žmogus, o pavyzdžiui - robotas, drakonas ar katė. Nors DI žaidimuose pasitelkiamas spręsti įvairias sudėtingas problemas, kartais kuriami veikėjai, kurie nepasižymi protingomis tendencijomis, nes tai duoda žaidimo turiniui įvairovės. Taip pat DI pasitelkiamas kaip įrankis, duoti ne žaidėjo valdomiems personažams skirtingas asmenybes, emocijas ar polinkius – pavaizduoti juos laimingus, išsigandusius, suirzusius ir liūdnus. [22]

Dar vienas iš labiausiai paplitusių DI metodų žaidimuose yra sukčiavimas - įprastas būdas, padedantis kompiuterio valdomam žaidėjui suteikti pranašumą prieš realų žaidėją. Ši metodika pasireiškia įvairiausiais būdais, o vienas iš pavyzdžių būtų kariniame strateginiame žaidime kompiuterio valdomam žaidėjui visos informacijos, susijusios su tikru žaidėju, suteikimas – tikro žaidėjo bazės vieta, valdomų personažų vietos, kiekiai ir tipai. Kompiuterio valdomo veikėjo sukčiavimas gali būti ir blogas dalykas – jeigu tikram žaidėjui pasidaro aišku, jog žaidime priešininkas akivaizdžiai sukčiauja, jam taps nuobodu ir atrodys, kad visos jo dedamos pastangos yra be prasmės. Taip pat nesubalansuotas kompiuterio sukčiavimas gali kompiuterio valdomam žaidėjui suteikti per daug galios, ko pasekoje žaidimas taps neįmanomas, todėl ir šiuo, ir praeitu atveju, kompiuterio valdomo žaidėjo sukčiavimas turi būti subalansuotas taip, kad tikram žaidėjui būtų sudaromas pakankamas iššūkis, o žaidimas atrodytų įdomus ir smagus [22].

1.4. Egzistuojančių strateginių žaidimų analizė

Šiuo metu rinkoje egzistuoja labai didelis kiekis įvairiausių tipų strateginių žaidimų – nuo paprasčiausių, tokių kaip šaškės, iki kompleksiškų, ne vieną DI metodą naudojančių, grandiozinės strategijos žaidimų, tokių kaip „Sid Meier‘s Civilization V“ ar „Age of Empires III“. Jie naudoja DI metodus, kad padidintų žaidimo patrauklumą potencialiems žaidėjams, sukeltų tikro, „gyvo“ ir protingo priešininko jausmą ir užtikrintų, jog žaidimą galima būtų pereiti ne vieną kartą. Šiam tikslui pasiekti, žaidimų kūrėjai ieško vis labiau priimtinų DI metodų, tačiau ne visada renkasi patį naujausią, geresnį, tačiau mažiau ištirtą ir testuotą metodą, kadangi tai padidina kuriamo žaidimo kaštus. Dažnu atveju liekama prie galbūt senesnių, tačiau labiau ištirtų ir laiko patikrintų metodų, kadangi juos naudojant žaidimo kūrimo procesas ir kaštai yra lengvai nuspėjami. Šiame poskyryje atlikta dviejų pasirinktų, vis dar aktualių ir populiarių strateginių žaidimų - „Sid Meier‘s Civilization V“ ir „Age of Empires III“, ir juose naudojamų DI metodų analizė.

1.4.1. Sid Meier‘s Civilization V

„Sid Meier‘s Civilization V“ yra vienas žymiausių visų laikų strateginių kompiuterinių žaidimų, kombinuojančių ėjimus ir realaus laiko strategiją. Sukurtas ir išleistas „Firaxis Games“ studijos 2010 metais, tačiau išlikęs etalonu dėl savo kompleksiškumo ir gilių strateginių galimybių dar iki šių dienų. Žaidimo tikslas - nuo senovės iki naujųjų laikų vystyti savo civilizaciją ir pasiekti vieną iš galimų

pergalių tipų: mokslo, diplomatijos, kultūros ir dominacijos. Žaidėjas valdo vieną iš 43 galimų visų laikų civilizacijų ir rungtyniauja su kompiuterio valdomomis civilizacijomis arba kitais tikrais žaidėjais tinklo rėžimu. Žaidimas įgyvendintas naudojantis tuo metu dar nauju žaidimų varikliu, ir buvo vienas pirmųjų strateginių žaidimų, kurie naudoja šešiakampius laukelius žaidimo žemėlapio objektams. [23][24]

„Civilization V“ naudoja įvarius DI metodus - procedūrinį žemėlapio generavimą, kelio radimo algoritmus ir blackboard architektūrą. Blackboard veikimas paremtas centrinio duomenų modulio komunikacija su šoniniais sprendimų priėmimo žaidimo moduliais, šiuo atveju, mokslo, kultūros, žvalgybos, karo ir ekonomikos. Šiame žaidime taip pat įgyvendinta kompiuterio valdomų priešininkų išankstinių nuostatų sistema, kuri kiekvienai civilizacijai duoda nuomonę ir polinkį į tam tikrą diplomatijos, ekonomikos, mokslo, kultūros ir karo vystymo strategiją. [25]

Nors „Civilization V“ dažnai analizuojamas žaidėjų ir modifikacijų kūrėjų bendruomenėje, oficiali ir akademiškai patikima informacija apie tikslų šio žaidimo DI metodų veikimą nėra viešai prieinama. Ši problema egzistuoja visos žaidimų industrijos mastu – žaidimų kūrimo korporacijos laiko savo sukurtų žaidimų kodą uždarą, o taikomų DI metodų aprašymus viešai neprieinamus. Dėl šios priežasties nepriklausomi kūrėjai turi ribotą kiekį informacijos apie pažangius metodus, naudojamus komerciškai sėkminguose žaidimuose, ir yra priversti „išradinėti dviratį“ patys [26].

1.4.2. Age of Empires III

„Age of Empires III“ yra realaus laiko strategijos (angl. Real Time Strategy - RTS) žanro kompiuterinis žaidimas, kurtas „Ensemble Studios“ ir išleistas „Microsoft“ 2005 metais. Jame vaizduojama Amerikų kolonizacija, o žaidėjas gali pasirinkti vieną iš 14 civilizacijų, kurias gali valdyti. Žaidėjo tikslas - pasiekti pergalę prieš kompiuterio valdomus arba realius žaidėjus tinklo rėžimu, išgaunant resursus, vystant ekonomiką, statant savo miesto pastatus ir įtvirtinimus, apmokant kareivius ir nugalint priešininkus. Žaidimo eiga suskirstyta į amžius, kurie eina žaidėjui tobulinant savo miestą ir atrakinant naujas technologijas. Šis žaidimas strategijos žanre pristatė naujovių, tokių kaip pagrindinio miesto turėjimas ir realaus laiko strategijos kombinaciją su rolių atributais. [27][28]

„Age of Empires III“ naudoja įvairius DI metodus – žemėlapio resursų ir žaidėjų startinių pozicijų generavimą, kelio radimo algoritmus ir sprendimų medžių kombinaciją su taisyklių rinkiniais. Kelio radimo implementacija turi trūkumų – pasitaiko situacijų, kai dėl žaidimo architektūros trūkumų ir ribotos vietos tarp objektų, judantys padaliniai užstringa vienas už kito, todėl mažėja DI efektyvumas taktinėse situacijose. Sprendimų medžių ir taisyklių rinkinių duetas leidžia kompiuterio valdomiems žaidėjams prisitaikyti prie besikeičiančios situacijos ir vykdyti sudėtingesnes, žmogui būdingas strategijas – ekonomikos vystymą, taktines atakas ir gynybines reakcijas. [29]

Kaip ir daugumoje komerciškai sėkmingų žaidimų, informacija apie DI veikimą „Age of Empires III“ yra viešai neprieinama. Didžioji dalis informacijos apie žaidimo vidinį veikimą ateina iš modifikacijų kūrėjų bendruomenės ir jų atliktų stebėjimų ir testavimų. Taip dar kartą įrodoma, kad DI metodai, naudojami komerciškai sėkminguose žaidimuose, dažniausiai lieka uždari praėjus ne vienam dešimtmečiui po jų išleidimo. Nepriklausomi žaidimų kūrėjai turi patys analizuoti ir bandyti pritaikyti DI sprendimus, kurie jau buvo išanalizuoti ir pritaikyti kitų [26].

1.5. Taikomi dirbtinio intelekto metodai

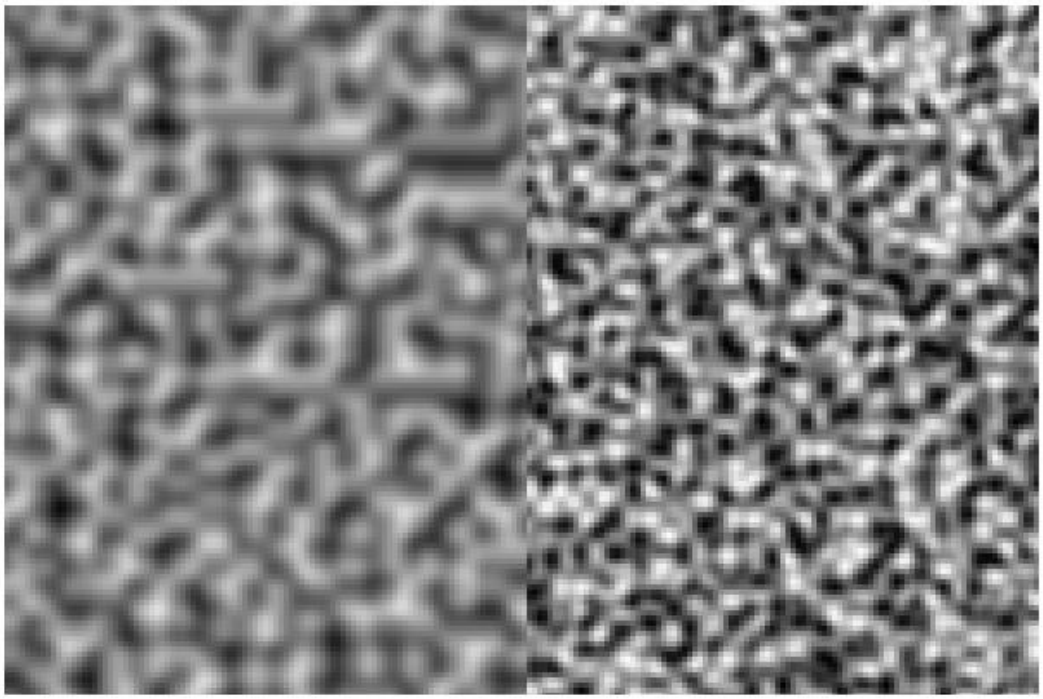
Toliau vardinami ir giliau analizuojami DI metodai, naudojami prieš tai išanalizuotuose žaidimuose, ir kurie bus tiriami šio dokumento tiriamojoje dalyje, siekiant palengvinti nepriklausomų žaidimų kūrėjų darbą, sutaupyti jiems laiko ir suteikti pritaikytų ir veikiančių pavyzdžių.

1.5.1. Procedūrinis žemėlapio generavimas

Žemėlapis – viena iš esminių strateginių žaidimų dalių, kurioje vyksta žaidimo veiksmas. Žemėlapio struktūra, susidedanti iš reljefo ir jame išdėstytų objektų, tokių kaip įvairūs resursai ir pastatai, turi tiesioginę įtaką žaidimo eigai – ji nulemia žaidėjų priimamus sprendimus, taktikų ir strategijų pasirinkimus. [30]

Norint, kad žaidimas turėtų aukštesnį pakartojamumo ir išskirtinumo laipsnį, naudojami įvairūs žemėlapio generavimo metodai. Skirtingai nei statiniai, rankiniu būdu sukurti žemėlapiai, procedūriškai generuoti žemėlapiai kuriami automatiniu būdų, remiantis apibrėžtomis taisyklėmis ir algoritmais, triukšmo funkcijomis ir atsitiktiniais kintamaisiais, dar vadinamais sėklomis (angl. Seeds). Šie metodai patys sugeneruoja žemėlapio reljefą ir jame išdėsto pasirinktus objektus, taip sutaupydami žaidimų kūrėjams laiko. Generuojami žemėlapiai gali būti įvairūs – subalansuoti, estetiški, pasižymėti skirtingomis reljefo zonomis ir būti skirti norimiems žaidimo scenarijams. [30]

Žemėlapių generavimas imituoja dizainerio kūrybinio projektavimo darbą, todėl svarbu nedaryti dažnos klaidos ir nemaišyti šio DI metodo su paprastais žaidimo techniniais aspektais. Generavimas vyksta be tiesioginio žmogaus įsikišimo, o rezultatas – įdomios ir žaidimo balansą išlaikančios aplinkos, kuriose vėliau vyksta žaidimo veiksmas. [31]



3 pav. Perlin triukšmo (kairėje) palyginimas su Simplex triukšmu (dešinėje). [32]

Šis DI metodas naudojamas reljefams ir pasaulio objektams kurti – kalnams, upėms, įvairioms biomoms ir ištekliams. Norint išlaikyti organiškus perėjimus tarp skirtingų žemėlapio zonų,

pasitelkiamos Perlin ir Simplex triukšmo funkcijos, o sugeneruotas triukšmo žemėlapis naudojamas kaip aukščio žemėlapis (angl. Height Map), kuriame išskiriamos lygumos, kalnai, upės, ežerai ir slėniai. **3 pav.** kairėje matomas Perlin triukšmas, kuriame matomi nuoseklūs, bangas primenantys perėjimai, todėl jis ypač tinka natūralių kraštovaizdžių generavimui. Dešinėje matomas Simplex triukšmas ryškesnis ir detalesnis nei Perlin, todėl labiau tinka kompleksiškesnių tekstūrų ar smulkesnių, detalesnių vietų generavimui. Lyginant skaičiavimo efektyvumą, Simplex lenkia Perlin ir pasižymi mažesniu sugeneruotų artefaktų skaičiumi. [32][33]

Triukšmo funkcijos gali būti kombinuojamos, šitaip išgaunant triukšmo sluoksniavimą (angl. Fractal Noise Layering), kuris leidžia sugeneruoti stambias ir labai smulkias reljefo detales.

Procedūriniame žemėlapių generavime taip pat svarbi ir žaidimo objektų, tokių kaip resursų ar pastatų išdėstymas ir pasiskirstymas. Priklausomai nuo norimų rezultatų ir generavimo logikos, galima taikyti įvarias papildomas objektų žemėlapyje generavimo strategijas. Šios strategijos ne tik užtikrina žaidimo balansą, bet ir suteikia įvairovės:

- Klasterines (angl. Cluster) – objektai grupuojami į telkinius – klasterius, sudaromos zonos, dėl kurių žaidėjai gali varžytis, pavyzdžiui – resursų telkiniai.
- Tinklelines (angl. Grid) – objektai paskirstyti tolygiai visame žemėlapyje nematomo tinklelio pagrindu, užtikrinant žemėlapio balansą.
- Spindulines (angl. Radial) – objektai koncentruoti ratu arba spirale aplink tam tikrą tašką – žemėlapio centrą, miesto ar kitos vietovės centrą, šitaip skatindami žaidėjus judėti iš krašto link objektų centro ir atvirkščiai. [34]

Generavimo proceso rezultatų pakartojamumas gali būti pasiektas naudojant sėklas, o tai leidžia sugeneruotais žemėlapiais naudotis tarp skirtingų žaidėjų ir kai kuriais atvejais, naudoti žemėlapius kaip aplinką įvairių DI metodų mokymosi procesams. [32][33]

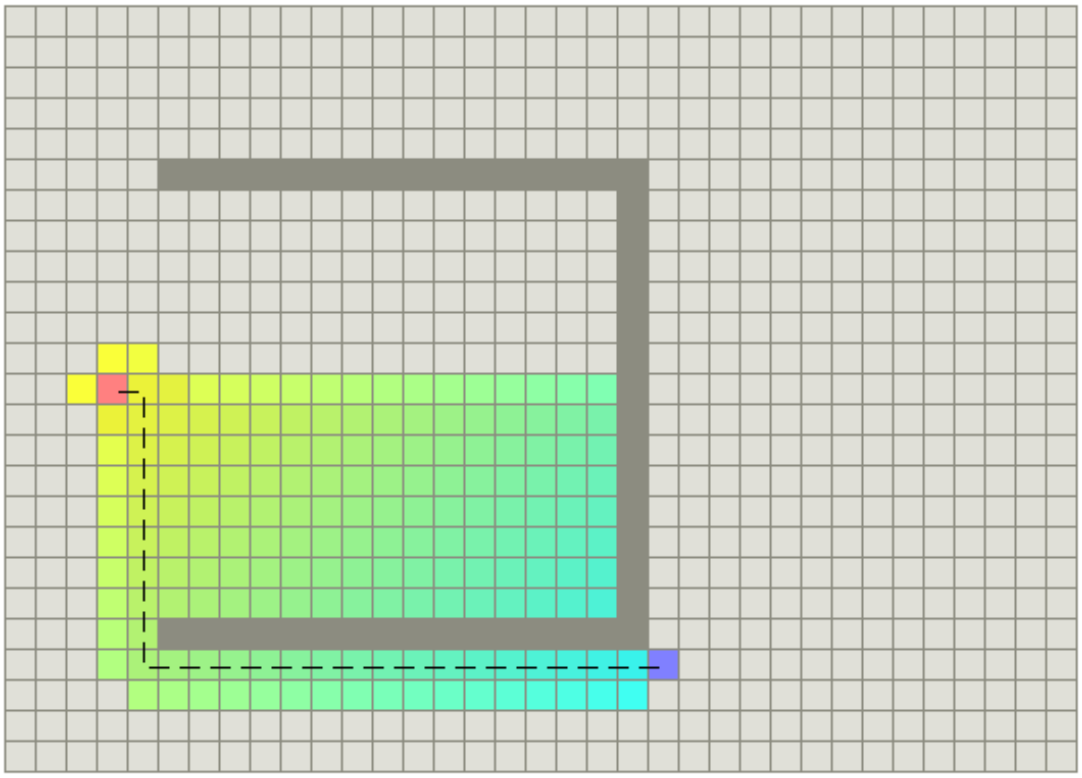
Procedūriškai generuoti žemėlapiai turi įtakos ir žaidimo pasiekiamumo aspektui – žaidėjai, turintys mažesnius techninius sistemos resursus, naudodami žemėlapio konfigūracijas gali sugeneruoti ir vėliau žaisti paprastesniuose, daug resursų nereikalaujančiuose žemėlapiuose. Taip užtikrinama lygi galimybė visiems žaidėjams žaisti žaidimą. [32][33]

Tiriamojoje šio darbo dalyje nagrinėjamas Perlin ir Simplex triukšmo funkcijų veikimas ir našumas, vykdomas palyginimas ir bandomos įvairios šių funkcijų kombinacijos su objektų generavimo strategijomis.

1.5.2. Kelio radimo metodai

Kelio radimo metodai - vieni svarbiausių ir dažniausiai naudojamų deterministinių DI metodų strateginiuose žaidimuose. Jie leidžia žaidimo veikėjams efektyviai judėti statinėse ir danaminėse žaidimų aplinkose, pakeliui išvengti kliūčių neišeinant iš žemėlapio ribų. Tokia veikėjų mobilumo galimybė yra daugumos strateginių ir kitų žanrų žaidimų pagrindas. [35]

Tipinis kelio radimo uždavinys (angl. Pathfinding Problem) strateginiame žaidime – apskaičiuoti trumpiausią ir efektyviausią kelią tarp taško A ir taško B atsižvelgiant į reljefą ir kitus aplinkos objektus. Dažniausiai šio tipo metodai veikia įsivaizduojamo tinklelio arba grafo pagrindu, kuriame taškai būna susieti su kaimyniniais taškais, o brianos arba sujungimai tarp taškų atspindi galimas judėjimo kryptis. [35]



4 pav. A* (A-Star) kelio radimo algoritmo veikimas. [36]

Populiariausi kelio radimo algoritmai:

- Dijkstra algoritmas – užtikrina absoliutų tikslumą, tačiau yra ganėtinai lėtas. Dažniausiai naudojamas, kai trumpiausio kelio radimas yra daug didesnis prioritetas nei kelio radimo logikos našumas.
- Breadth-First-Search (BFS) – sąlyginai paprastas, tačiau neoptimalus algoritmas, kurio pagrindas yra ieškoti trumpiausio kelio pagal mazgų kiekį, neatsižvelgiant į faktinį apskaičiuoto kelio atstumą. Naudojamas mažesnėse ir paprastesnėse aplinkose, tačiau nėra efektyvus.
- A* algoritmas (angl. A-Star Algorithm) – atvaizduotas 4 pav., apjungia Dijkstra algoritmo tikslumą su heuristiniu efektyvumu ir naudoja įvertinimo funkciją krypčių prioretizavimui. Vienas dažniausiai naudojamų algoritmų žaidimų kūrime. [36]

Šio tipo DI metodai laikomi deterministiniais nes jų veikimas yra nuspėjamas, kadangi prie tų pačių pradinių sąlygų visada gaunamas tas pats rezultatas. Strateginiuose žaidimuose šie metodai dažniausiai kombinuojami su kitais DI metodais, tokiais kaip - sprendimų medžiai, baigtinių būsenų mašinos ir blackboard sistemos. [36]

Unity NavMesh

Unity NavMesh – vienas plačiausiai naudojamų kelio radimo metodų žaidimuose įgyvendintuose naudojant Unity žaidimų variklį.

Šio metodo veikimo pagrindas – automatiškai sugeneruojamas navigacijos tinklas (angl. Navigation Mesh), parodantis laisvo judėjimo zonas priklausomas nuo žemėlapio reljefo ir jame egzistuojančių kliūčių. Šis navigacijos tinklas sugeneruojamas iš anksto, prieš pradedant žaisti žaidimą, tačiau

egzistuoja sprendimai, tokie kaip NavMeshObstacle, leidžiantys žaidimo metu dinamiškai atsirandantiems objektams atlikti šio tinklo deformacijas. Sugeneravus navigacijos tinklą, kelio radimo agentai efektyviai atlieka judėjimą jame. [41]

Šio metodo didžiausi privalumai – lengva integracija ir naudojimas. Žaidimo kūrėjui užtenka pažymėti norimus paviršius, kuriais galima naviguoti ir kliūtis, kuriose navigacija negalima, ir sugeneruoti navigacijos tinklą. Žaidimo veikėjams priskiriami NavMesh Agent komponentai, kurie leidžia realiu laiko keisti judėjimo parametrus – greitį, pagreitį, stabdymo atstumą ir kliūčių vengimo atstumą. [41]

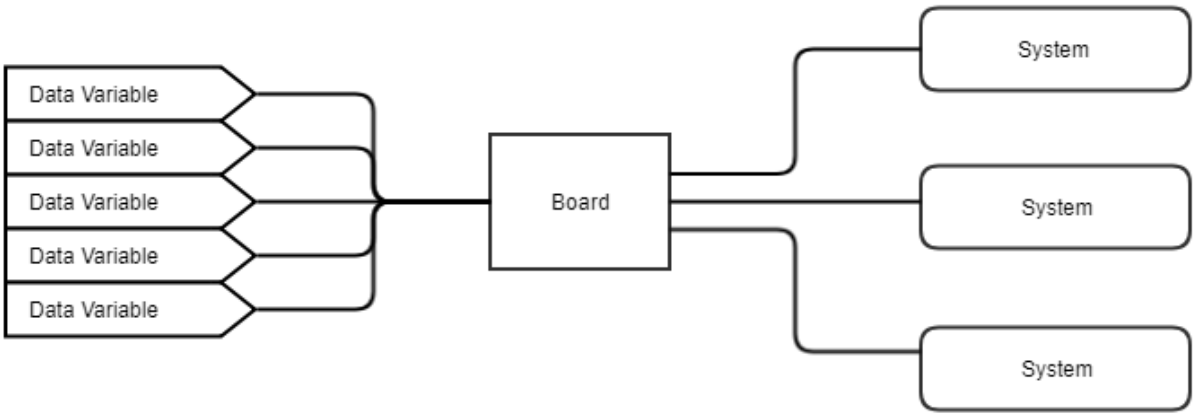
Nepaisant privalumų, Unity NavMesh yra ribotas aplinkose, kuriose reikalinga vertikali navigacija arba egzsituoja objektai, veikiantys tunelio principu. Tokiais atvejais yra būtini papildomų sprendimų įgyvendinimai, todėl sudėtinguose žaidimuose dažniausiai šis metodas naudojamas kartu su kitais DI metodais, tokiais kaip - sprendimų medžiai, baigtinių būsenų mašinos ar blackboard sistemos.

Tiriamojoje šio darbo dalyje nagrinėjamas įgyvendintas A* algoritmo pagrindu veikiantis kelio radimo metodas, kombinuotas su įsivaizduojamo tinklelio sistema. Šio metodo veikimas lyginamas su Unity NavMesh, vertinami įvairūs kiekybiniai ir kokybiniai kriterijai.

1.5.3. Blackboard sistemos

Blackboard architektūra – viena pažangiausių deterministinių DI architektūrų, dažniausiai taikoma kompleksiškuose strateginiuose žaidimuose. Susideda iš skirtingų DI modulių, dar kitaip vadinamų ekspertų sistemomis, ir bendros duomenų struktūros – lentos (angl. Blackboard).

Šiame metode, kiekviena ekspertinė sistema bendradarbiauja su lenta – ekspertinės sistemos analizuoja esamą padėtį apie žaidimo pasaulį iš savo pusės ir dalinasi šia informacija su lenta, o lenta dalinasi gauta apibendrinta informacija iš kitų ekspertinių sistemų. Šie informaciniai mainai padeda užtikrinti, kad ekspertinės sistemos priimtų sprendimus atsižvelgdamos į bendrą žaidimo padėtį, ne tik iš savo pusės. [39][40]



5 pav. Blackboard sistemos pavyzdys. [40]

5 pav. Pavaizduotas blackboard sistemos šablonas, kuriame matomas duomenų suvedimas į centrinę lentą. Ši informacija prieinama visoms ekspertų sistemoms, kurios ją analizuodamos reaguoja pagal savyje apibrėžtas funkcijas. Šis modelis suteikia DI komponentams galimybę pasikliauti faktais apie

žaidimo pasaulio būseną iš visų galimų aspektų ir sinchronizuotai reaguoti į besikeičiančią informaciją. [39][40]

Šis DI metodas ypač naudingas kompleksiškuose žaidimuose, todėl blackboard sistemos dažnai taikomos strategijos žanre, kadangi jame figuruoja žaidimai, turintys daug skirtingų aspektų – resursų ir ekonomikos valdymą, padalinius, pastatų ir miestų statymą, puolimo ir gynybos strategijas. Blackboard sistema leidžia apjungti visus šiuos aspektus, o kompiuterio valdomam žaidėjui apjungtai vertinti situaciją ir priimti sprendimus pagrįstus visa galima informacija. [39][40]

Šio darbo tiriamojoje dalyje nagrinėjama įgyvendinta blackboard architektūra grįsta sistema, naudojama automatiniam žaidimo ekonomikos valdymui ir pastatų statymui. Vykdomi šios sistemos kiekybiniai vertinimai.

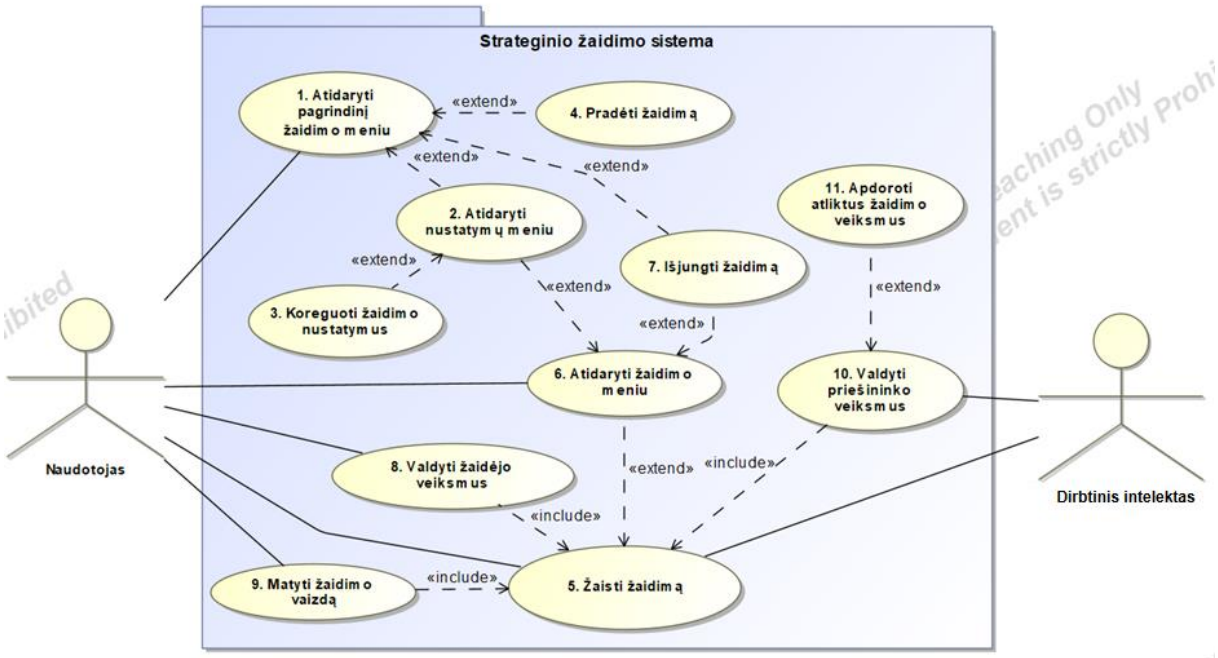
2. Projektinė dalis

Šiame skyriuje aprašoma strateginio žaidimo prototipo projektinė dalis, pasitelkiant panaudojimo atvejų, paketų, klasių, veiklos, būsenų ir išdėstymo diagramas.

2.1. Sistemos sudėtis (panaudojimo atvejų modelis)

Strateginio žaidimo prototipui priskiriami šie panaudojimo atvejai, taip pat atvaizduoti **6 pav.**:

- 1. Atidaryti pagrindinį žaidimo meniu
- 2. Atidaryti nustatymų meniu
- 3. Koreguoti žaidimo nustatymus
- 4. Pradėti žaidimą
- 5. Žaisti žaidimą
- 6. Atidaryti žaidimo meniu
- 7. Išjungti žaidimą
- 8. Valdyti žaidėjo veiksmus
- 9. Matyti žaidimo vaizdą
- 10. Valdyti priešininko veiksmus
- 11. Apdoroti atliktus žaidimo veiksmus



6 pav. Strateginio žaidimo prototipo panaudojimo atvejų diagrama

2.2. Sistemos kūrimo procesas

Šiam projektui parenkamas proceso modelis - tradicinis Krioklio (kaskadinis).

Šis modelis parenkamas todėl, nes kuriamas projektas, kurio visi keliami reikalavimai yra iš anksto žinomi, jų keitimosi tikimybė yra minimali, o pats projekto uždavinys nėra išskirtinis, ir jo sprendimo metodika yra žinoma ir aiški.

Šis modelis turi privalumus puikiai tinkančius šio projekto pobūdžiui - modelio procesas yra aiškus ir lengvai koordinuojamas, o reikalaujamos darbo sąnaudos yra minimalios jeigu nenumatomi grįžimai į praeitus modelio etapus.

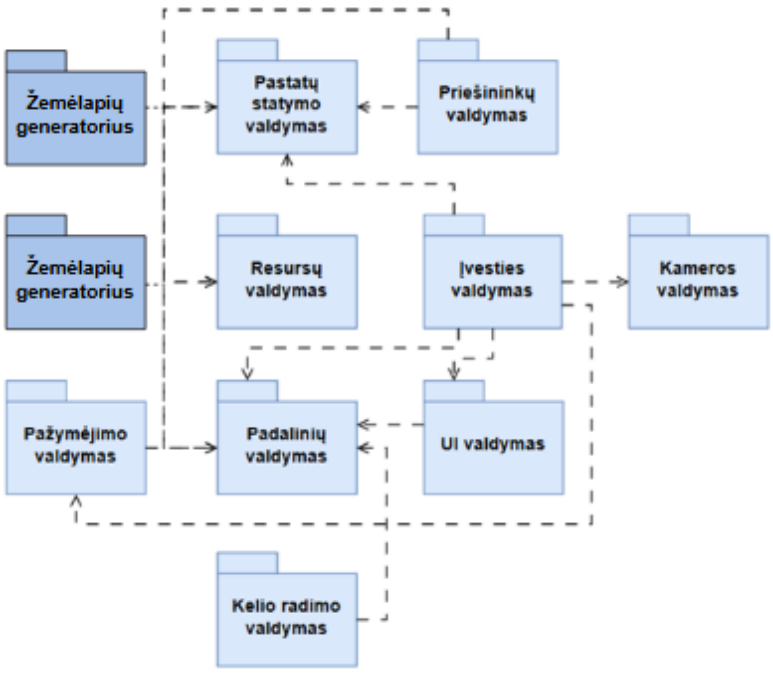
Modelis bus skirstomas į 4 pagrindinius etapus:
1. Reikalavimų surinkimas ir analizė
2. Planavimas ir projektavimas
3. Kodavimas ir testavimas
4. Panaudotų DI metodų tyrimas

2.3. Sistemos statinis vaizdas

2.3.1. Apžvalga

Projekto statinis vaizdas skiriamas į paketus, matomus pateiktame paveikslėlyje (**7 pav.**).

Paketai:
- Pastatų statymo valdymas
- Priešininkų valdymas
- Resursų valdymas
- Įvesties valdymas
- Kameros valdymas
- Pažymėjimo valdymas
- Padalinių valdymas
- UI valdymas
- Kelio radimo valdymas
- Žemėlapių generatorius



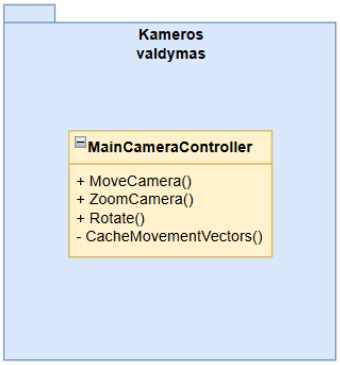
7 pav. Strateginio žaidimo prototipo paketų diagrama

2.3.2. Paketų detalizavimas

Kiekvienam paketui pateikiamos klasių diagramos ir aprašymai.

Kameros valdymo paketas

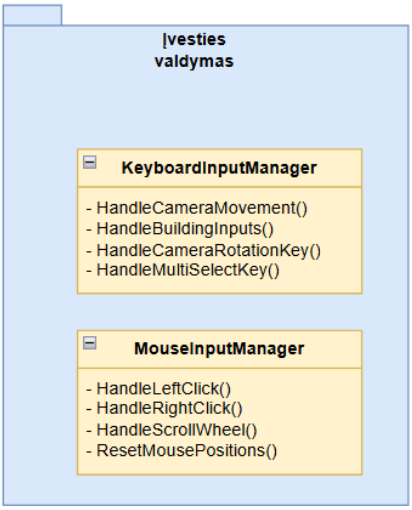
Šio paketo klasės yra atsakingos už pagrindinės žaidimo kameros valdymą. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**8 pav.**) Paveikslėlyje parodyta klasė atlieka šias funkcijas: kameros pozicijos keitimas, kameros rotacijos keitimas ir kameros vaizdo padidinimas.



8 pav. Paketo „Kameros valdymas“ klasių diagrama

Įvesties valdymo paketas

Šio paketo klasės yra atsakingos už žaidėjo įvesties signalų, pateikiamų įvesties įrenginiais, apdorojimą. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**9 pav.**). Paketo klasės atlieka šias funkcijas: gauna žaidėjo įvestį, atsikiria įvesties tipą, prilyginą šį tipa norimam atlikti veiksmui ir galiausiai siunčia veiksmus į kituose paketuose esančius valdiklius.

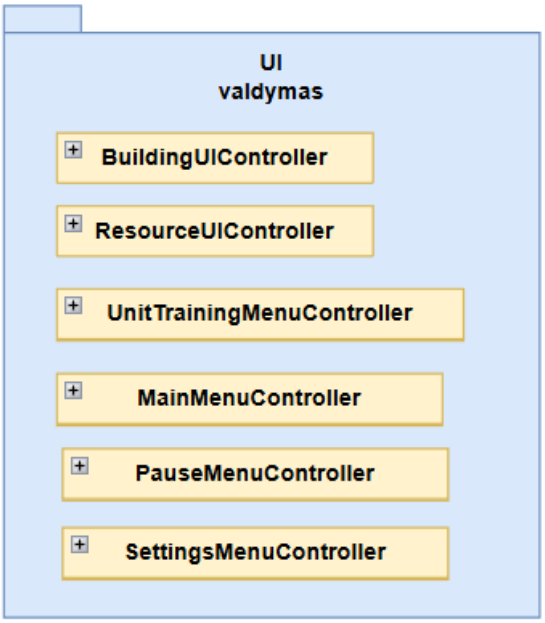


9 pav. Paketo „Įvesties valdymas“ klasių diagrama

UI valdymo paketas

Šio paketo klasės yra atsakingos vartotojo grafinės sąsajos valdymą. Unity žaidimų variklyje, grafinės vartotojo sąsajos įgyvendinimas vykdomas drobių (angl. Canvas) pagrindu. Kiekviena drobė turi savo

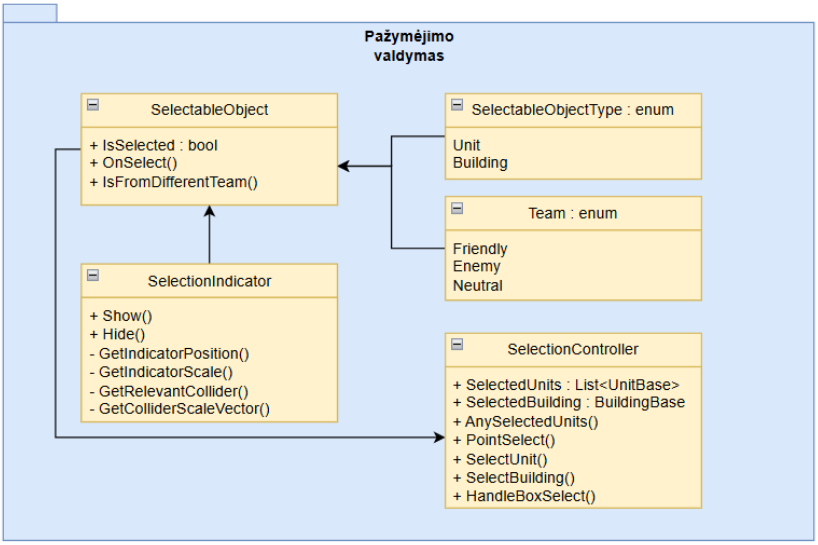
elementus, kurie gali būti skirtingų tipų. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**10 pav.**) Paketo klasės atlieka šias funkcijas: atlieka drobių kūrimą, atnaujinimą bei keitimą, drobių elementų kūrimą ir atnaujinimą. Išskiriami šeši drobių elementų tipai: tekstinis langas (angl. TextBox), mygtukas (angl. Button), mygtukas su tekstiniu langu (angl. ButtonWithTextBox), nustatymų langas (angl. SettingsBox), slankiojimo elementas (angl. Slider) ir slinkimo elementas (angl. Scroller).



10 pav. Paketo „UI“ klasių diagrama

Pažymėjimo valdymo paketas

Šio paketo klasės yra atsakingos už padalinių ir pastatų pažymėjimo logikos valdymą. Šią pažymėjimo logiką taip pat naudoja ir kiti paketai, pagrinde padalinių ir pastatų valdymo paketai. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**11 pav.**).

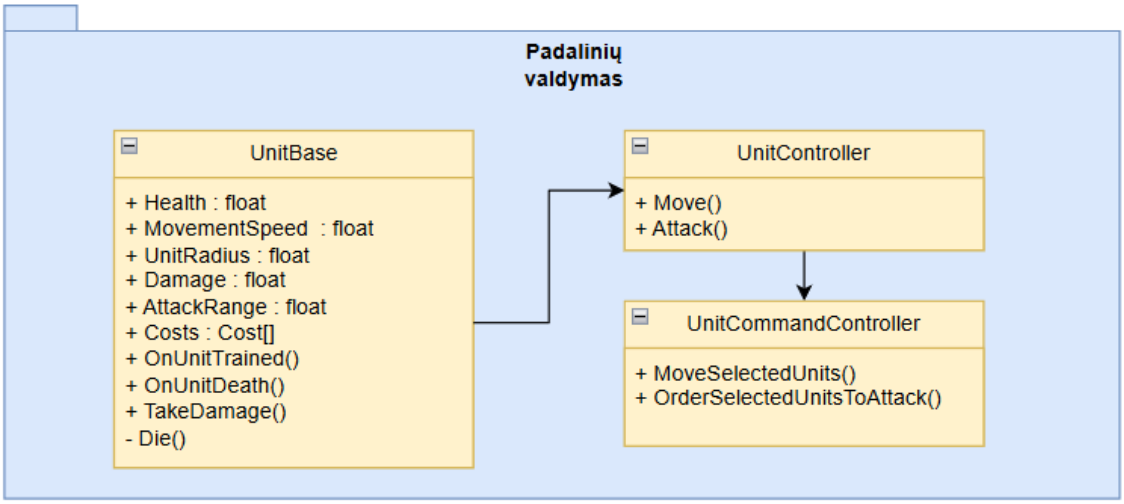


11 pav. Paketo „Pažymėjimo valdymas“ klasių diagrama

Aukščiau pateiktoje diagramoje matoma, kad klasės valdo padalinių ir pastatų pažymėjima. Surinkti pažymėjimų duomenys yra perduodami į kituose paketuose esančias klases.

Padalinių valdymo paketas

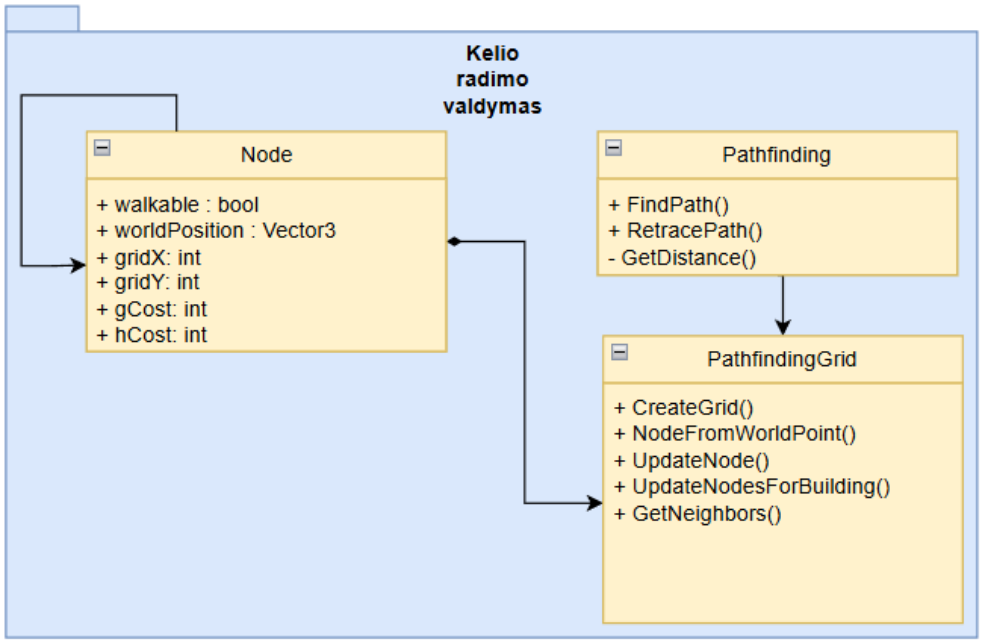
Šio paketo klasės yra atsakingos už padalinių vaikščiojimo ir atakavimo logikos valdymą. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**12 pav.**).



12 pav. Paketo „Padalinių valdymas“ klasių diagrama

Kelio radimo valdymo paketas

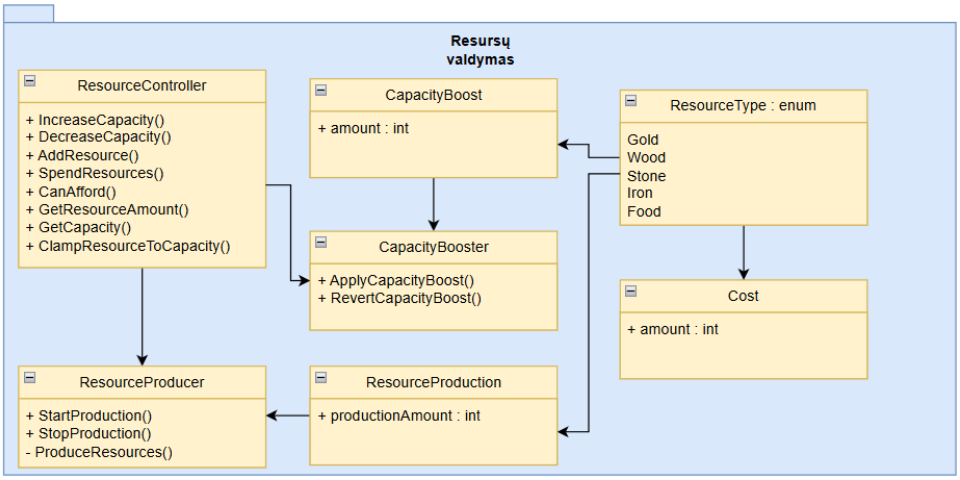
Šio paketo klasės yra atsakingos už kelio radimą žemėlapyje. Pakete esančių klasių duomenis naudoja kiti paketai, pagrinde padalinių ir pastatų paketai. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**13 pav.**).



13 pav. Paketo „Kelio radimo valdymas“ klasių diagrama

Resursų valdymo paketas

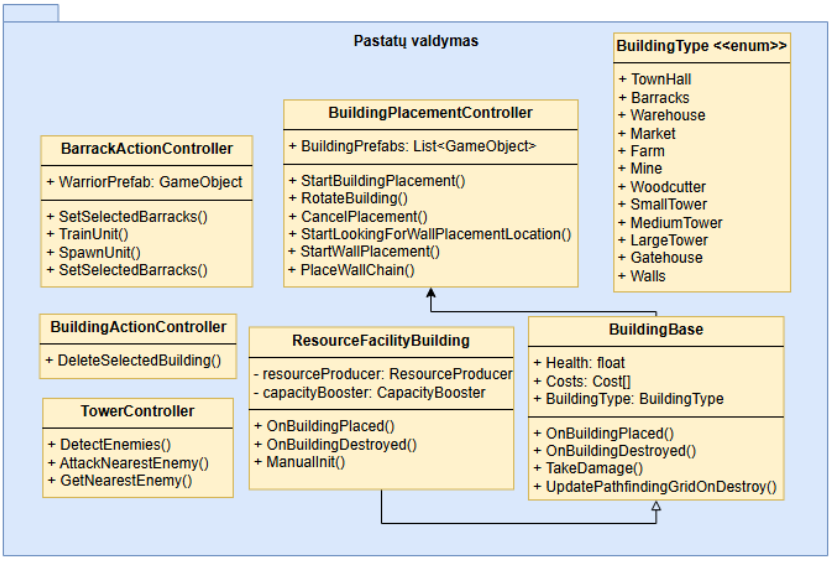
Šio paketo klasės yra atsakingos už viską, kas susiję su resursais – resursų gaminimą, resursų vietos plėtimą, resursų leidimą ir resursų kiekio tikrinimą. Paketo klasės yra plačiai naudojamos kituose paketuose. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**14 pav.**).



14 pav. Paketo „Kelio radimo valdymas“ klasių diagrama

Pastatų statymo valdymo paketas

Šio paketo klasės yra atsakingos už visą veiksmų su pastatais logiką – statymą, vietos ieškojimą, sugriovimą, padalinių gaminimą. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**15 pav.**).

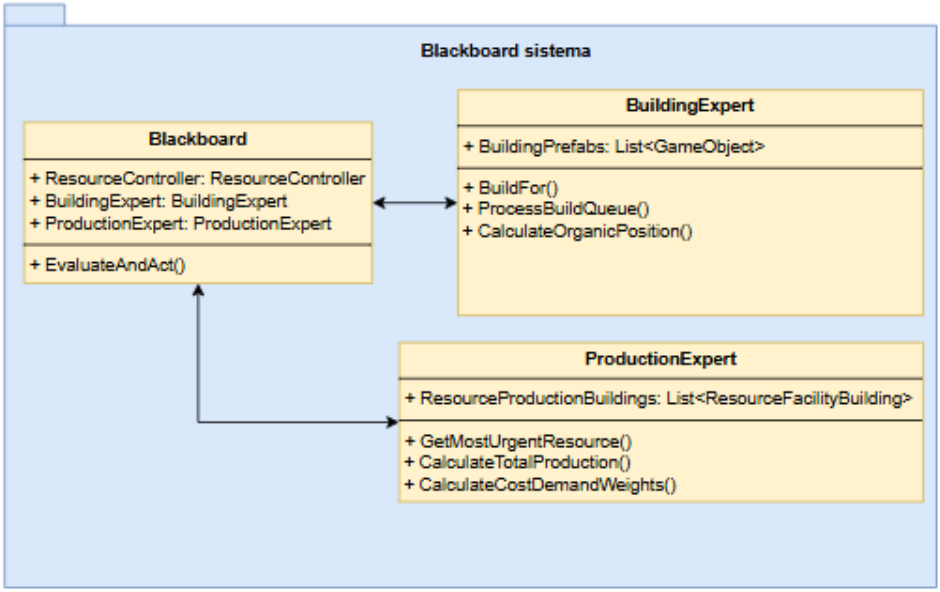


15 pav. Paketo „Pastatų valdymas“ klasių diagrama

Blackboard sistemos paketas

Šio paketo klasės yra atsakingos už blackboard sistemos logiką. Blackboard klasė – bendra duomenų struktūra, kuri bendrauja su BuildingExpert ir ProductionExpert klasėmis. BuildingExpert klasė

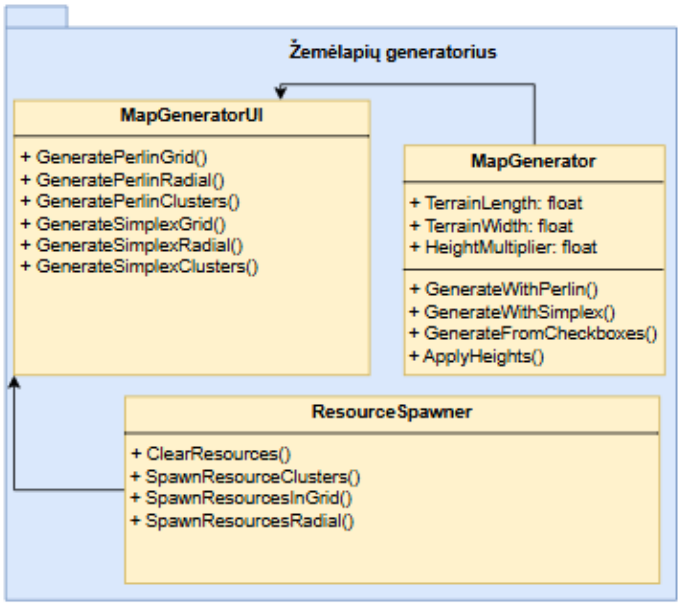
atsakinga už automatizuotą pastatų statymą, o ProductionExpert, atsakinga už ekonominių situacijų interpretavimą. Paketo klasių struktūra pavaizduota žemiau pateiktame paveikslėlyje (**16 pav.**).



16 pav. Paketo „Blackboard sistema“ klasių diagrama

Žemėlapių generatoriaus paketas

Šio paketo klasės yra atsakingos už žemėlapių generavimo logiką. MapGeneratorUI klasė, kuriai Unity žaidimų variklyje yra priskiriami mygtukai, kurių pagalba generuojami žemėlapiai ir juose išdėstomi resursai. MapGenerator klasė atsakinga už žemėlapio reljefo generavimą pagal pasirinktus ilgio, pločio ir aukščio multiplikatoriaus parametrus. ResourceSpawner klasė atsakinga už norimo kiekio resursų išdėstymą sugeneruotame žemėlapyje, pagal pasirinktus parametrus ir strategiją. Paketo klasių struktūra pateikiama žemiau esančiame paveikslėlyje (**17 pav.**).



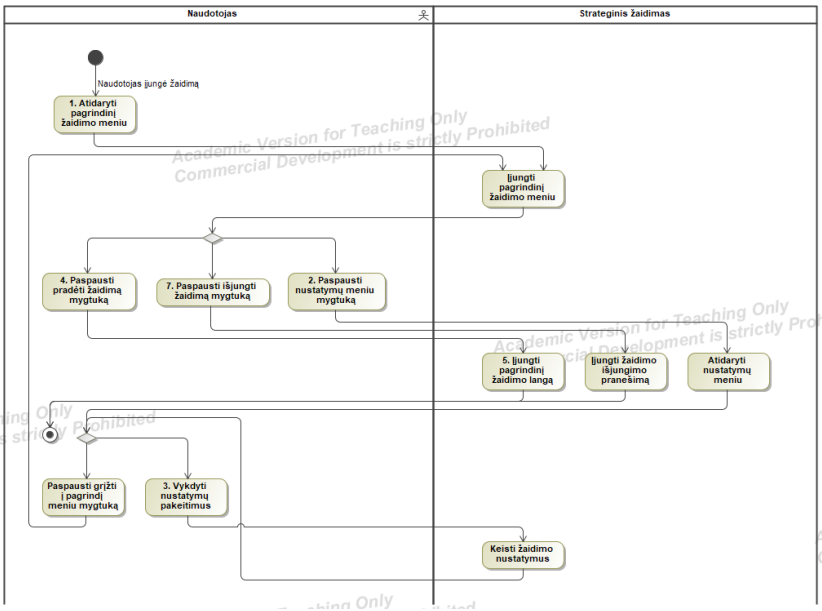
17 pav. Paketo „Žemėlapių generatorius“ klasių diagrama

2.4. Sisemos dinaminis vaizdas

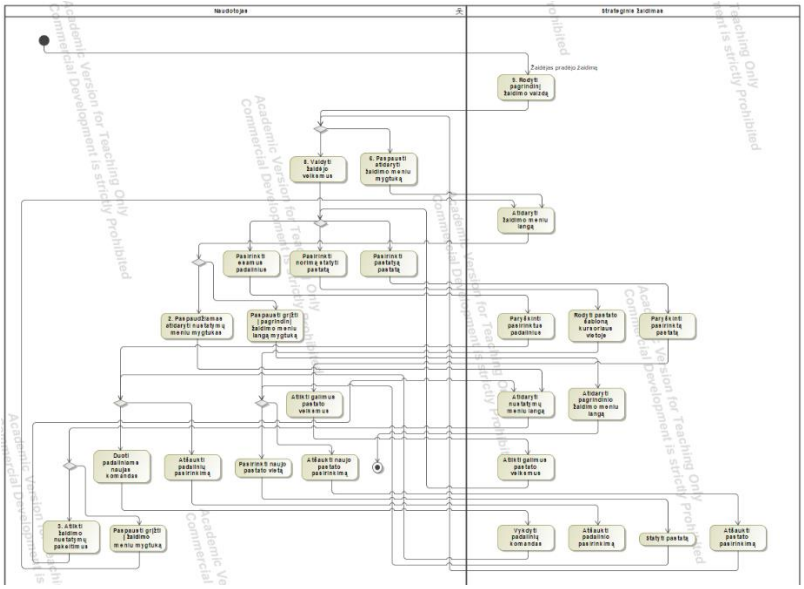
Šiame skyriuje pateikiamos veiklos, būsenų ir sekų diagramos. Diagramos palengvina bendrą strateginio žaidimo prototipo veikimo proceso supratimą ir leidžia greičiau perprasti ryšį tarp skirtingų sistemos dalių.

2.4.1. Veiklos diagramos

Žemiau pateiktos veiklos diagramos pasirinktiems esminiams panaudojimo atvejams – žaidimo pagrindinio meniu navigacijai ir pagrindiniam žaidimo funkcionalumui, sutinkamam žaidimo metu. (**18 ir 19 pav.**).



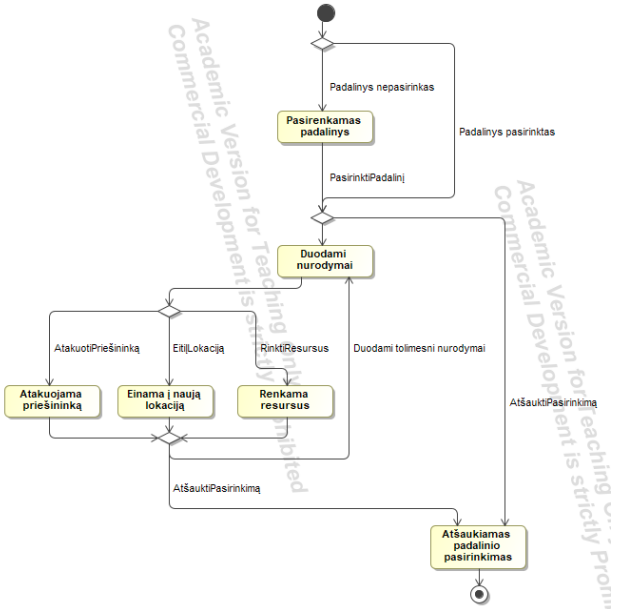
18 pav. Žaidimo pagrindinio meniu navigavimo veiklos diagrama



19 pav. Žaidimo pagrindinio funkcionalumo, sutinkamo žaidimo metu, veiklos diagrama

2.4.2. Būsenų diagramos

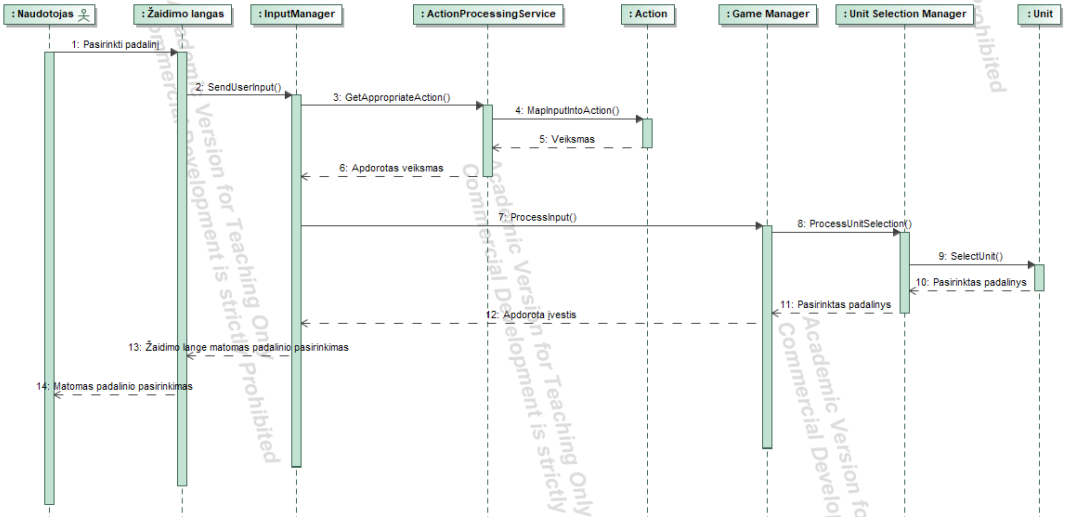
Žemiau pavaizduota būsenų diagrama (**20 pav.**) apibrėžia kiekvieno padalinio būsenas žaidimo metu. Padalinio būsenos kinta priklausomai nuo žaidėjo pasirinktų veiksmų su tuo padaliniu. Padalinio būsenos visada prasideda padalinio pasirinkimu, ir visada baigiasi padalinio pasirinkimo atšaukimu.



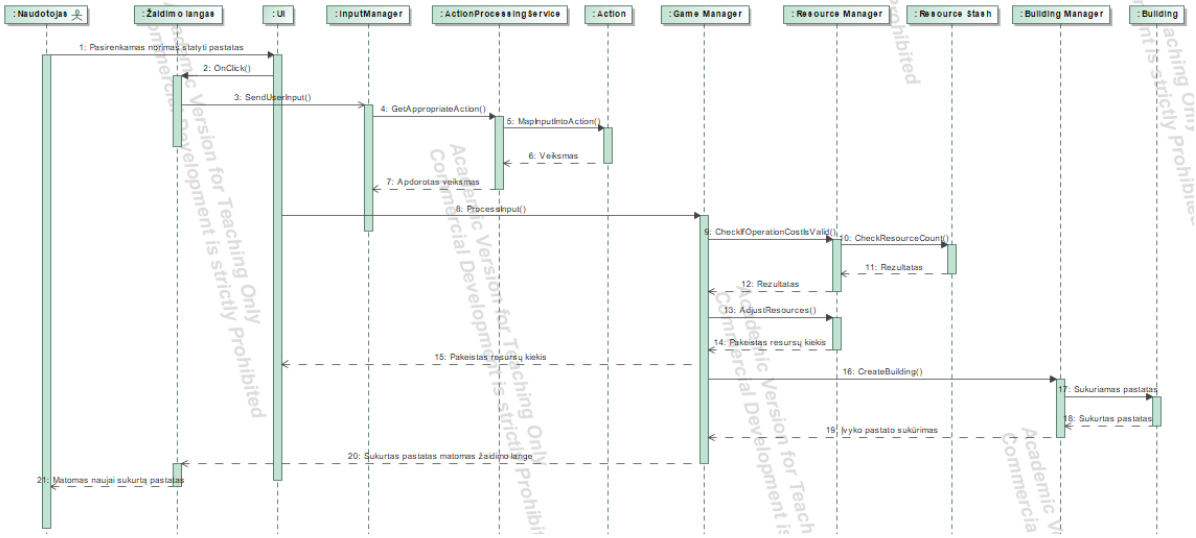
20 pav. Padalinio būsenų diagrama

2.4.3. Sekų diagramos

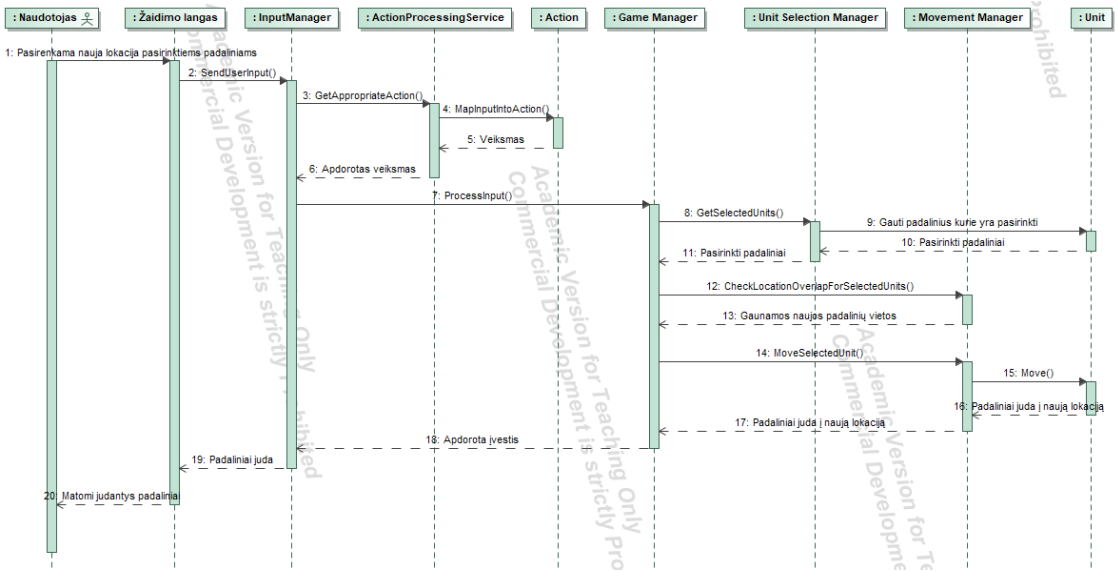
Žemiau esančiuose paveikslėliuose (**21-23 pav.**) pateikiamos sekų diagramos pasirinktam strateginio žaidimo funkcionalumui. Svarbu pastebėti, kad sekų diagramose neapibrėžiami elementarūs strateginio žaidimo panaudojimo atvejai, o apibrėžiamas sudėtingesnis funkcionalumas, iš kurio susideda panaudojimo atvejai „8. Valdyti žaidėjo veiksmus“ ir „5. Žaisti žaidimą“. Funkcionalumas, kuris turi identiškas sekų diagramas, kaip ir žemiau apibrėžtos sekų diagramos, taip pat nėra apibrėžtas.



21 pav. Padalinio pasirinkimo sekų diagrama



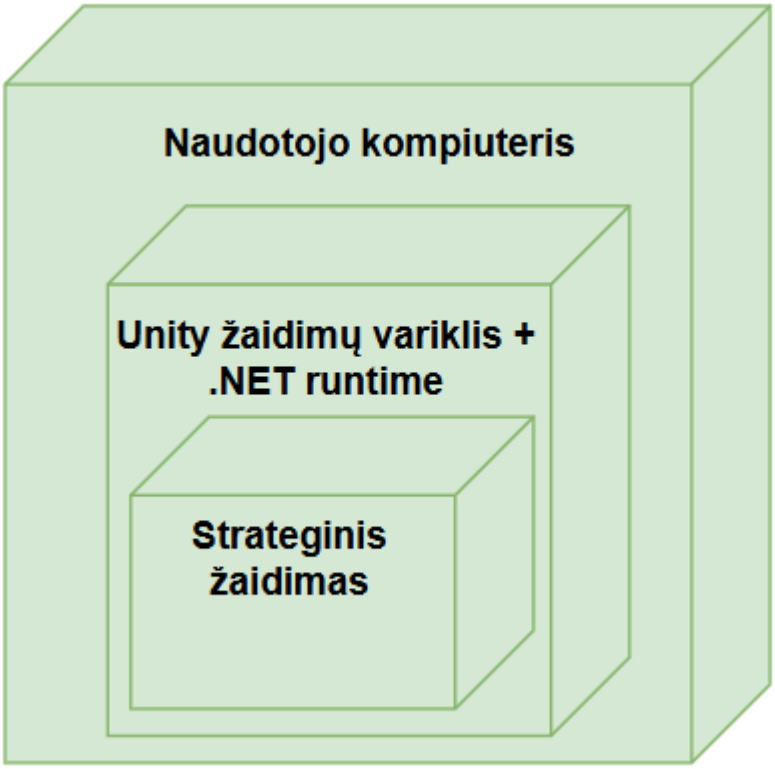
22 pav. Pastatų statymo sekų diagrama



23 pav. Padalinių judėjimo į pasirinktą lokaciją sekų diagrama

2.4.4. Išdėstymo (deployment) vaizdas

Projekto sistemos išdėstymo vaizdas pateikiamas įšdėstymo diagrama (**24 pav.**). Visos žaidimo dalys veikia naudotojo kompiuteryje – strateginis žaidimas veikia Unity žaidimų variklyje, kartu su visa žaidimo logika ir DI metodais.



24 pav. Strateginio žaidimo išdėstymo diagrama

2.4.5. Duomenų vaizdas

Projekto metu kuriama sistema – strateginis žaidimas, nenaudos duomenų bazės. Žaidimo duomenys žaidimo metu bus saugomi darbinėje atmintyje (RAM) Unity žaidimo variklio pagalba, automatiniu būdu.

3. Tiriamoji dalis

3.1. Įvadas

Šioje dalyje aprašomas 3 įgyvendintų dirbtinio intelekto metodų – procedūrinio žemėlapių generavimo, kelio radimo metodo ir blackboard sistemos, tyrimai. Kiekvienas iš šių DI metodų buvo sėkmingai integruotas į strateginio žaidimo prototipą, sukurtą naudojant Unity žaidimų variklį ir C# programavimo kalbą.

Atlikus analizę, šie DI metodai buvo atrinkti atsižvelgiant į jų svarbą ir pritaikomumą strateginių žaidimų logikos kūrime ir įvairių žaidimo pasaulio sričių automatizavime. Kiekvienam DI metodui suformuluoti tikslai ir hipotezės, nustatyta testavimo metodika ir vertinimo kriterijai. Tyrimų tikslas buvo ne tik įgyvendinti šiuos metodus, bet ir palyginti juos su alternatyvomis, įvertinti privalumus ir trūkumus, pateikti patobulinimus arba alternatyvas.

Eksperimentų rezultatų dėka galimas objektyvus pasirinktų DI metodų silpnųjų ir stipriųjų pusių ir įtakos žaidimo kokybei įvertinimas.

Sekančiuose poskyriuose (3.3. – 3.6.) kiekvieno DI metodo tyrimas pateikiamas atskirai, o šiuos tyrimus sudaro – įžanga, tyrimo tikslai ir uždaviniai, hipotezės, hipotezių patikrinimo metodika, tyrimo eiga, tyrimo metu naudota programinė įranga, rezultatai, apribojimai ir išvados. Hipotezių patikrinimas atliekamas kiekvieno tyrimo išvadose.

3.2. Bendros tyrimo sąlygos

Tyrimo sąlygos atliktos naudojantis asmeniniu kompiuteriu, naudojančiu 64-bitų Windows 10 Pro operacinę sistemą. Kompiuterio techninės specifikacijos:

- CPU: Ryzen 7 5800X, 8 branduolių
- RAM: 32GB DDR4
- GPU: NVIDIA GeForce RTX 3080 Ti 12GB
- SSD: 1TB

Žaidimo prototipui įgyvendinti ir eksperimentiniams tyrimams atlikti naudotas Unity žaidimų variklis, kartu su C# 9 programavimo kalba ir Visual Studio 2022 Community integruota kūrimo aplinka.

3.3. Procedūrinio žemėlapių generavimo tyrimas

3.3.1. Įžanga į tyrimą

Žemėlapio struktūra ir jame esantys objektai – vienas iš pagrindinių strateginių žaidimų aspektų. Žemėlapis nėra tik vieta kur vyksta žaidimo veiksmas, jis nulemia žaidėjų atliekamus veiksmus, žaidimo eigą, ir taktikas, kurios naudojamos pergalei pasiekti. [30] Dėl šių priežasčių procedūrinis žemėlapių generavimas yra svarbus komponentas strateginių žaidimų kūrime. Palyginus, rankiniu būdų kurti žemėlapiai reikalauja daugiau laiko ir darbo, žaidėjui greitai tampa žinomi ir dažniausiai neturi didelės pakartojamumo vertės. Norėdami išvengti šių trūkumų, žaidimų kūrėjai dažnai renkasi procedūrinio generavimo metodus žemėlapių kūrimui, kadangi jų pagalba, pagal apibrėžtas taisykles ir parametrus kūrimo procesas vyksta automatizuotai.

Procedūrinis žemėlapių generavimas imituoja žmogaus kūrėjo sprendimus – kur dėti resursus, kokio tipo reljefą formuoti ir kaip išlaikyti balansą tarp žaidėjų. Šis metodas - automatinis, parametrizuotas ir gebantis priimti sprendimus pagal iš anksto nustatytas taisykles. Pasirinkus tinkamas logines ir heuristines taisykles, šį DI metodą galima išplėsti taip, kad jis prisitaikytų prie žaidėjo žaidimo stiliaus arba jau jo atliktų veiksmų. [31][32][33]

Tyrimo metu analizuojama, kaip skirtingos triukšmo funkcijos ir resursų išdėstymo strategijos veikia žemėlapio generavimo techninį našumą, panaudojamumą, vizualinį vientisumą, reljefo struktūrą ir žaidybinį balansą.

3.3.2. Įgyvendinimas

Įgyvendintas procedūrinis 3D žemėlapių generatorius, naudojantis Perlin ir Simplex triukšmo funkcijas žemėlapių aukščių generavimui. Šių funkcijų pagalba generuojamos natūraliai atrodančios žaidimo aplinkos – kalnai, slėniai, lygumos ir kiti reljefo elementai, o parametrų keitimo galimybė užtikrina žemėlapių įvairumą.

- Perlin triukšmas – pasižymi sklandžiais ir nuosekliais perėjimais tarp skirtingų reljefo zonų ir yra dažnai naudojamas gamtos kraštovaizdžių simuliacijoms.
- Simplex triukšmas – efektyvesnė Perlin alternatyva, kuri generuoja mažiau artefaktų, pasižymi didesniu detalumo lygiu ir našumu trimačiuose ar didelės raiškos žemėlapiuose.

Kartu su reljefo generavimu, buvo įgyvendintos trys resursų išdėstymo sugeneruotame žemėlapyje strategijos, kurių pseudo kodas pateikiamas **25 pav.**

```
Method SpawnResourcesInClusters(woodCount, rockCount):
    Clear all previously spawned resources
    For each resource type (wood, rock):
        spawned = 0
        While spawned < requested count:
            clusterCenter = random point on map
            For i = 0 to clusterSize:
                If spawned >= requested count, break
                offset = small random offset from center
                position = clusterCenter + offset
                position = clamped and aligned to terrain
                If position is valid:
                    spawn resource at position
                    increment spawned

Method SpawnResourcesInGrid(woodCount, rockCount):
    Clear all previously spawned resources
    gridSize = sqrt(totalCount)
    cellSize = mapSize divided by gridSize
    spawnedWoods = 0
    spawnedRocks = 0
    For each grid cell (x, z):
        If both resources are fully spawned, stop
        position = center of cell with small random offset
        position = clamped and aligned to terrain
        If wood needed and position is valid:
            spawn wood
            increment spawnedWoods
        Else if rock needed and position is valid:
            spawn rock
            increment spawnedRocks

Method SpawnResourcesRadial(woodCount, rockCount):
    Clear all previously spawned resources
    center = map center
    maxRadius = half of map size
    ringCount = sqrt(totalCount)
    ringSpacing = maxRadius / ringCount
    spawnedWoods = 0
    spawnedRocks = 0
    For each ring:
        resourcesInRing = evenly divide totalCount by ringCount
        radius = ring * ringSpacing
        For each resource in ring:
            If both resources are fully spawned, stop
            angle = evenly spaced around circle
            position = center offset by radius and angle
            position = clamped and aligned to terrain
            If wood needed and position is valid:
                spawn wood
                increment spawnedWoods
            Else if rock needed and position is valid:
                spawn rock
                increment spawnedRocks
```

25 pav. Resursų išdėstymo strategijų pseudo kodas.

Resursų išdėstymo strategijų aprašymai:

- Klasterinė (angl. Cluster) - Klasterinė strategija grupuoja resursus į telkinius, suteikdama žaidimui natūralesnį, realistiškesnį pojūtį ir skatinanti žaidėją planuoti aplink šias zonas.
- Tinklelinė (angl. Grid) - Tinklelinė strategija užtikrina tolygų resursų pasiskirstymą visame žemėlapyje, o tai tinka subalansuotiems žaidimams, kur kiekvienas žaidėjas pradeda su vienodomis galimybėmis.
- Spindulinė (angl. Radial) - Spindulinė strategija leidžia išdėstyti resursus koncentriniais ratais aplink pagrindinius taškus (pvz., žaidėjų pradines bazes), kas sukuria simetrijos įspūdį ir leidžia geriau valdyti žaidimo balansą, ypač mažesniuose žemėlapiuose.

Prieš padedant kiekvieną resursą reljefe, parenkamas atsitiktinis jo dydžio daugiklis – nuo 0.8 iki 3 ir atsitiktinis pasukimo kampas pagal Y ašį – nuo 0 iki 360 laipsnių.

3.3.3. Tyrimo tikslai ir uždaviniai

Tyrimo tikslas – įvertinti procedūrinio žemėlapių generavimo metodų taikymą strateginio žaidimo aplinkoje, palyginant skirtingas triukšmo funkcijas - šiuo atveju Perlin ir Simplex, ir resursų išdėstymo strategijas - klasterinę, tinklelinę, spindulinę, pagal jų vizualinį vientisumą, balansą ir techninį našumą.

Uždaviniai:
1. Sukurti reljefo generavimo sistemą, naudojančią Perlin ir Simplex triukšmo funkcijas, skirtas žemėlapių aukščio duomenų generavimui ir pagal parametrus galinčią sugeneruoti bent 5 skirtingo tipo žemėlapius.
2. Įgyvendinti klasterinę, tinklelinę ir spindulinę resursų generavimo strategijas, integruoti jas kartu su reljefo generavimo metodais ir užtikrinti, kad kiekviena iš jų leistų gauti bent 200 objektų kiekviename žemėlapyje.
3. Apibrėžti ir taikyti kiekybinius ir kokybinius vertinimo kriterijus žemėlapių tyrimui.
4. Eksperimentiškai įvertinti Perlin ir Simplex triukšmo funkcijų rezultatus naudojant vienodus parametrus – intensyvumą, žemėlapio dydį ir triukšmo dažnį.
5. Pateikti rekomendacijas apie žemėlapio generavimo metodų kombinaciją, pasižyminčią geriausiu balansu tarp generavimo laiko, vizualinės kokybės ir žaidimo įvairovės.

3.3.4. Tyrimo hipotezės

Atlikus šio DI metodo analizę (1 skyrius), iškeltos šios hipotezės:

H1: Žemėlapis, sugeneruotas Simplex triukšmo funkcija pasižymės didesniu reljefo tolygumu nei Perlin triukšmas.

H2: Tinklelinė resursų išdėstymo strategija užtikrins labiausiai subalansuotą pasiskirstymą.

H3: Simplex triuškmo funkcijos ir tinklelinės resursų išdėstymo strategijos kombinacija pasieks geriausius rezultatus generavimo laiko, vizualinio vientisumo ir žemėlapio balanso srityse.

3.3.5. Hipotezių patikrinimo metodika

Hipotezės bus tikrinamos atliekant eksperimentus, kurių metu bus sugeneruoti skirtingi žemėlapiai naudojant šias triukšmo funkcijų ir resursų išdėstymo strategijų kombinacijas:
- Triukšmo funkcijos: Perlin ir Simplex
- Resursų išdėstymo strategijos: Klasterinė, tinklelinė ir spindulinė.

Iš viso 6 skirtingos kombinacijos – 2 triukšmo funkcijos x 3 resursų išdėstymo strategijos. Siekiant užtikrinti rezultatų patikimumą, kiekvienai kombinacijai bus sugeneruoti 3 unikalūs žemėlapiai – todėl iš viso bus atliekama 18 eksperimentų.

Vertinimo kriterijai:
1. Reljefo generavimo laikas milisekundėmis – parodo, kiek laiko užtrunka sugeneruoti reljefą.
2. Resursų generavimo laikas milisekundėmis – parodo, kiek laiko užtrunka reljefe išdėstyti resursus.
3. Panaudojamo reljefo procentas – vertina, kiek reljefo teritorijos yra galima naudoti žaidimo veikėjų judėjimui, apskaičiuojama pagal tai, ar reljefo šlaito kampas ne didesnis nei 30 laipsnių.
4. Sugeneruotų resursų kiekis vienetais – parodo, kiek resursų buvo išdėstyta reljefe.
5. Resursų standartinis nuokrypis – naudojant šilumos žemėlapį (angl. Heatmap) ir kvadratinio nuokrypio nuo vidurkio formulę, matuojamas resursų išdėstymo balansas.
6. Resursų klasteriavimo koeficientas – skaičiuojant vidutinį atstumą iki artimiausio kaimyno, matuoja kaip arti vienas kito yra išdėstyti resursai.

Eksperimentų eiga:
1. Sugeneruojamas reljefas naudojant Perlin arba Simplex triukšmo funkciją.
2. Vykdomas resursų generavimas naudojant klasterinę, tinklelinę arba spindulinę strategiją.
3. Atliekamas vertinimas pagal aukščiau aprašytus vertinimo kriterijus.
4. Kiekvienos kombinacijos rezultatai pateikiami lentelėmis ir ekrano iškarpų pavidalu.

Atlikus eksperimentus, rezultatai bus lyginami tarpusavyje, siekant nustatyti kombinaciją, turinčią geriausius vertinimus. Taip pat bus patvirtinamos arba paneigiamos iškeltos hipotezės.

3.3.6. Tyrimo eiga

Tyrimo metu buvo atlikti 18 eksperimentų, kuriuose buvo naudojamos skirtingos triukšmo funkcijų ir resursų išdėstymo strategijų kombinacijos. Naudotos dvi triukšmo funkcijos – Perlin ir Simplex, ir trys resursų išdėstymo strategijos – tinklelinė, klasterinė ir spindulinė. Kiekviena iš kombinacijų generuota tris kartus, šitaip siekiant panaikinti atsitiktinumo įtaką eksperimentuose.

Žemėlapių reljefų generavimas

Žemėlapių reljefai buvo sugeneruoti naudojantis Perlin ir Simplex triukšmo funkcijomis, kurios kiekvienam reljefo taškui suteikia skaitinę vertę, kuri vėliau transformuojama į aukštį (angl. Heightmap). Perlin funkcija sukuria vientisus, banguotus perėjimus. Simplex funkcija pasižymi didesniu detalumu ir mažesniu artefaktų kiekiu.

Kiekvieno eksperimento metu naudoti šie parametrai reljefo generavimui:
- Reljefo ilgis ir plotis – 512 Unity žaidimų variklio vienetų.
- Aukščio daugiklis – 8.

Kiekvieno eksperimento metu buvo matuojami šie aspektai:
1. Reljefo generavimo laikas – matuojamas milisekundėmis, naudojant .NET laikmačio klasę „Stopwatch.cs“.
2. Panaudojamo reljefo procentas – apskaičiuojama reljefo dalis, neturinti šlaitų, kurių kampas yra didesnis nei 30 laipsnių. Šlaitai turintys didesnį kampą nei 30 laipsnių yra nepasiekiami žaidimo veikėjams ir ant jų nėra generuojami resursai. Naudojama formulė:

$$P = \frac{Panaudojami\ reljefo\ taškai}{Bendras\ reljefo\ taškų\ skaičius} \times 100$$

Resursų generavimas

Naudotos trys resursų išdėstymo strategijos:

- Tinklelinė – resursai išdėstomi tolygiai pagal nematomą tinklelį visame žemėlapyje.
- Klasterinė – resursai išdėstomi telkiniai aplink atsitiktinai parenkamus centrus.
- Spindulinė – resursai išdėstomi ratu aplink šalia centro atsitiktinai parinktą tašką.

Kiekvieno eksperimento metu naudoti šie parametrai resursų išdėstymui:

- Generuojamų medžio resursų maksimalus kiekis – 256.
- Generuojamų akmens resursų maksimalus kiekis – 256.
- Klasterių spindulio ribos, naudojamos klasterinio išdėstymo strategijos metu – nuo -5 iki 5.
- Klasterių dydžio ribos, naudojamos klasterinio išdėstymo strategijos metu – nuo -5 iki 5.

Kiekvieno eksperimento metu buvo matuojami šie aspektai:

1. Resursų generavimo laikas – matuojamas milisekundėmis, naudojant .NET laikmačio klasę „Stopwatch.cs“.
2. Bendras sugeneruotų resursų kiekis.
3. Standartinis resursų nuokrypis – skaičiuojamas iš resursų 2D šilumos žemėlapio (angl. Heatmap) pagal formulę:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_1 - \bar{x})^2}$$

kur x_i – resursų kiekis kiekviename šilumos žemėlapio kvadrate, o $\bar{x}$ – vidutinis resursų kiekis.

4. Klasterizavimo koeficientas – vidutinis artimiausių kaimyninių resursų atstumas kiekvienam resursui, apskaičiuojamas pagal formulę:

$$K = \frac{1}{N}\sum_{i=1}^{N} \underset{j \neq i}{mindist}(r_i, r_j)$$

kur n – visų sugeneruotų resursų kiekis, mindist(r_i, r_j) – mažiausias euklidinis atstumas tarp resurso r_i ir r_j.

Mažesnis standartinis resursų nuokrypis ir mažesnis klasterizavimo koeficientas reiškia didesnį žemėlapio balansą ir tolygesnį išteklių prieinamumą.

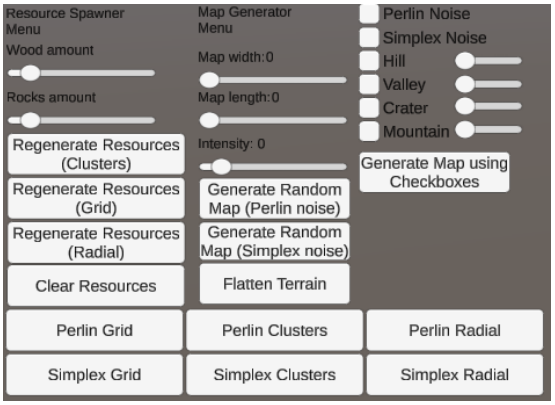
Duomenų rinkimas

Kiekvieno eksperimento kiekiniai rezultatai buvo matomi Unity žaidimų variklio konsolės lange ir vėliau surinkti ir atvaizduoti lentelėmis. Sugeneruotų žemėlapių ekrano iškarpos pateikiamos tyrimo rezultatų poskyryje ir šio dokumento prieduose.

Eksperimentų metu gauti duomenys buvo analizuojami, kad patikrinti tyrimo hipotezes ir įvertinti praktinį metodų efektyvumą.

3.3.7. Tyrimo metu naudota programinė įranga

Visa žemėlapių ir resursų generacija vyko Unity žaidimų variklio aplinkoje, naudojantis eksperimentams sukurtais, grafinės sąsajos elementais parodytais **26 pav.**



26 pav. Žemėlapių generavimo grafinė sąsaja.

Iš visų mygtukų, matomų **26 pav.**, eksperimentų metu naudoti 6 – „Perlin Grid“, „Perlin Clusters“, „Perlin Radial“, „Simplex Grid“, „Simplex Clusters“ ir „Simplex Radial“. Kiekvienas iš jų surištas su specialiai tyrimui sukurta testavimo logika, parašyta C# programavimo kalba naudojant „Visual Studio 2022 Community“ integruota kūrimo aplinką, matoma 26 pav.

```
140     private void GenerateSimplexGrid()
141     {
142         var stopwatch = new Stopwatch();
143
144         stopwatch.Start();
145         mapGenerator.GenerateWithSimplex();
146         var mapGenerationTime = stopwatch.ElapsedMilliseconds;
147         stopwatch.Stop();
148         stopwatch.Reset();
149
150         stopwatch.Start();
151         resourceSpawner.SpawnResourcesInGrid(WoodCount, RockCount);
152         var resourceSpawnerTime = stopwatch.ElapsedMilliseconds;
153         stopwatch.Stop();
154
155         var resourceCount = resourceSpawner.spawnedResources.Count;
156         var standardDeviation = resourceSpawner.CalculateResourceBalanceStandardDeviation();
157         var clusteringCoefficient = resourceSpawner.CalculateClusteringCoefficient();
158         var usableTerrainPercentage = mapGenerator.CalculateUsableTerrainPercentage();
159
160         UnityEngine.Debug.Log($"[Simplex][Grid] Map generation time: {mapGenerationTime}ms;" +
161             $" Resource generation time: {resourceSpawnerTime}ms;" +
162             $" Usable Terrain Percentage: {usableTerrainPercentage}," +
163             $" Generated resource count: {resourceCount}," +
164             $" Standard Resource Deviation:  {standardDeviation} ," +
165             $" Resource Clustering Coefficient: {clusteringCoefficient}.");
166     }
167
```

27 pav. Žemėlapių generavimo testavimo logika.

Kiekvienai triukšmo funkcijos ir resursų išdėstymo strategijos kombinacijai sukurti 6 analogiški metodai, atitinkantys **27 pav.** matomą struktūrą. Testavimo metodas susideda iš šių dalių:

1. Paleidžiamas laikmatis
2. Generuojamas reljefas naudojantis Perlin arba Simplex triukšmo funkcijomis.
3. Užfiksuojamas laikmačio parodymas.
4. Per nauja paleidžiamas laikmatis.
5. Reljefui generuojami resursai, naudojantis tinkleline, klusterine arba spinduline išdėstymo strategija.
6. Apskaičiuojamas sugeneruotų resursų kiekis.

7. Apskaičiuojamas standartinis resursų nuokrypis.
8. Apskaičiuojamas klusteriškumo koeficientas.
9. Apskaičiuojamas panaudojamo reljefo procentas.
10. Laikmačio ir apskaičiavimų rezultatai atspausdinami Unity žaidimų variklio konsolės lange.

3.3.8. Tyrimo rezultatai

Tyrimo rezultatai išskiriami į 3 dalis – našumo rezultatai, kokybiniai rezultatai ir vizualiniai rezultatai. Bendra visų rezultatų lentelė, pateikta 1 priede, buvo išskirstyta į atskiras lenteles pagal tiriamus aspektus.

Našumo rezultatai

Našumo rezultatai pateikiami lentelės pavidalu (1 lentelė). Kiekvienam eksperimentui apskaičiuoti reljefo ir resursų generavimo laikai, apskaičiuoti kiekvienos triukšmo funkcijos ir resursų išdėstymo strategijos kombinacijos laikų vidurkiai per tris eksperimentus.

1 lentelė. Eksperimentų našumo rezultatai.

Triukšmo funckija	**Resursų išdėstymo strategija**	**Eksperimento numeris**	**Reljefo generavimo laikas (ms)**	**Vidutinis reljefo generavimo laikas (ms)**	**Resursų generavimo laikas (ms)**	**Vidutinis resursų generavimo laikas (ms)**
Perlin	Tinklelinė	1	26	26	11	12
		2	26		13	
		3	26		12	
	Klasterinė	1	26	26,33	21	22,33
		2	26		21	
		3	27		25	
	Spindulinė	1	26	26	12	13
		2	26		12	
		3	26		15	
Simplex	Tinklelinė	1	82	82	10	11
		2	82		10	
		3	82		13	
	Klasterinė	1	83	82,67	24	21,67
		2	82		21	
		3	83		20	
	Spindulinė	1	83	82,67	10	8,67
		2	83		8	
		3	82		8	

Iš lentelėje atvaizduotų rezultatų matoma, kad reljefo generavimas naudojant Perlin triukšmo funkciją buvo žymiai greitesnis – vidutiniškai 26 milisekundės, nei generavimas naudojant Simplex triukšmo funkciją – vidutiniškai 83 milisekundės. Šis skirtumas kylo iš to, kad Perlin triukšmo funkcijos matematinis skaičiavimas yra paprastesnis, todėl ši funkcija labiau tinka strateginiams žaidimams,

kuriuose reikia pastoviai procedūriškai generuoti žemėlapius realiu laiku. Nors reljefai generuoti Simplex triukšmo funkcija yra detalesni, jos matematinis skaičiavimas yra sudėtingesnis ir trunka ilgiau.

Nors Simplex reljefo generavimas yra lėtesnis, šiuose reljefuose resursų generavimas yra greitesnis, o šis skirtumas labiausiai matomas kombinuojant šią triukšmo funkciją su spinduline resursų išdėstymo strategija. Šis skirtumas tarp resursų generavimo strategijų našumo Perlin ir Simplex reljefuose kyla iš to, kad Simplex reljefai yra detalesni, o tai reiškia jog jie turi daugiau statesnių šlaitų, o bendras plotas, skirtas resursams išdėstyti yra mažesnis.

Tinklelinės strategijos našumo rezultatai yra stabiliausi – naudojant Perlin ir Simplex reljefus, resursai generuojami greitai – vidutiniškai per 11.5 milisekundžių.

Ilgiausias laikas matomas klasterinės resursų generavimo strategijos atveju – Perlin reljefe vidutiniškai truko 22.3 milisekundes, o Simplex reljefe – 21.6 milisekundės. Tai parodo, kad generuojant resursus klasterine strategija prireikia daugiau iteracijų tinkamų pozicijų paieškai, ypač esant labiau suskaidytam reljefui.

Kokybiniai rezultatai

Kokybiniai rezultatai pateikiami lentelės pavidalu (2 lentelė). Kiekvienai triukšmo funkcijos ir resursų išdėstymo strategijos kombinacijai buvo atlikti 3 eksperimentai, kurių metu užfiksuotas panaudojamo reljefo procentas, sugeneruotų resursų kiekis, standartinis resursų nuokrypis ir resursų klasteriavimo koeficientas.

2 lentelė. Eksperimentų kokybės rezultatai.

Triukšmo funckija	**Resursų išdėstymo strategija**	**Eksperimento numeris**	**Panaudojamo reljefo procentas (%)**	**Sugeneruotų resursų kiekis (skaičius)**	**Standartinis resursų nuokrypis**	**Resursų klasteriškumo koeficientas**
Perlin	Tinklelinė	1	96.97	339	0.276	16.365
		2	96.95	375	0.290	15.806
		3	96.92	343	0.277	16.498
	Klasterinė	1	97.07	512	0.365	9.395
		2	96.97	512	0.355	9.775
		3	96.95	512	0.354	9.766
	Spindulinė	1	96.95	354	0.281	11.559
		2	96.99	353	0.282	11.490
		3	96.99	350	0.283	11.607
Simplex	Tinklelinė	1	74.32	303	0.261	17.657
		2	73.56	304	0.263	17.348
		3	75.17	301	0.260	17.854
	Klasterinė	1	75.05	512	0.352	10.142
		2	73.63	512	0.352	10.263
		3	74.05	512	0.357	10.257

	Spindulinė	1	75.05	299	0.262	12.504
		2	74.49	238	0.237	14.561
		3	74.93	242	0.236	13.805

Panaudojamo reljefo procentas parodė didelį kontrastą tarp Perlin ir Simplex triukšmo funkcijomis generuotų reljefų. Perlin triukšmo generuoti reljefai žymiai tolygesni, kadangi panaudojamo reljefo procentas buvo 96.92-97.07%, lyginant su Simplex 73.56-75.17%. Šis skirtumas parodo, kad naudojant Simplex triukšmo funkciją didesnė reljefo dalis turi per daug stačius šlaitus, dėl kurių atsiranda daugiau nepasiekiamų žemėlapių vietų.

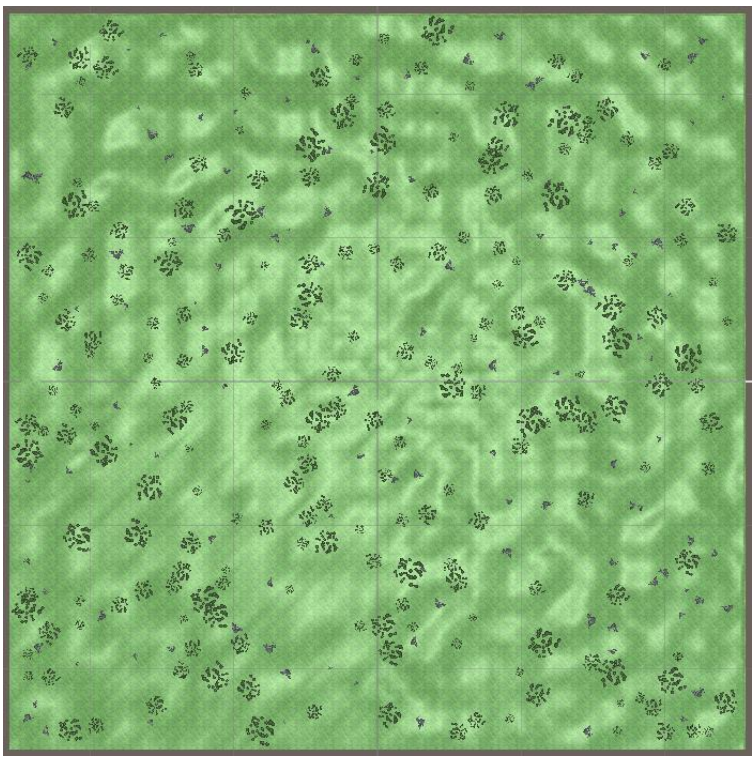
Stebint sugeneruotų resursų kiekius, matoma, kad didžiausi šio rodiklio skaičiai užfiksuoti naudojant klasterinę išdėstymo strategiją, nepriklausomai nuo naudotos triukšmo funkcijos. Tuo tarpu Simplex triukšmo funkcijos ir spindulinės resursų išdėstymo strategijos kombinacija sugeneravo mažiausiai resursų – 238-299, o taip yra dėl mažesnio panaudojimo reljefo procento ir didesnio atmetamų resursų pozicijų kiekio dėl pačios strategijos vidinės logikos. Iš sugeneruotų resursų kiekio tendencijų matoma, kad panaudojamo reljefo procentas turi įtaką sugeneruotam resursų kiekiui tinklelinėje ir spindulinėje resursų generavimo strategijose.

Mažesnis standartinis resursų nuokrypis indikuoja tolygesnį resursų pasiskirstymą visame žemėlapyje. Geriausi rezultatai šiam rodikliui gauti naudojant Simplex triukšmo funkciją kartu su spinduline resursų išdėstymo strategija – 0.236-0.262. Šis rezultatas yra ypač įdomus, kadangi nors ši kombinacija sugeneravo mažiausius resursų kiekius, jie buvo geriausiai pasiskirstę. Praščiausius rezultatus rodė Perlin triukšmo funkcijos ir klasterinės išdėstymo strategijos kombinacija – 0.354-0.365, tačiau šis rezultatas nėra netikėtas dėl pačios strategijos veikimo pobūdžio.

Resursų klasteriškumo koeficientas parodė vidutinius atstumus iki artimiausių kaimyninių resursų, o mažesnės šio rodiklio reikšmės reiškia didesnį resursų susitelkimą glaudžiose grupėse. Kaip ir tikėtasi, Perlin triukšmo funkcijos ir klasterinės išdėstymo strategijos kombinacija pasižymėjo mažiausiais koeficientais – 9.395-9.775. Didžiausi koeficientai buvo pastebėti Simplex triukšmo funkcijos ir tinklelinės resursų generavimo strategijos kombinacijoje – 17.348-17.854, kas parodo, jog generuojami resursai yra plačiai išskaidyti, nėra koncentruotų grupių. Šie rezultatai padeda įvertinti, kokią triukšmo funkcijos ir resursų išdėstymo strategijos kombinaciją geriausia naudoti pagal norimus rezultatus.

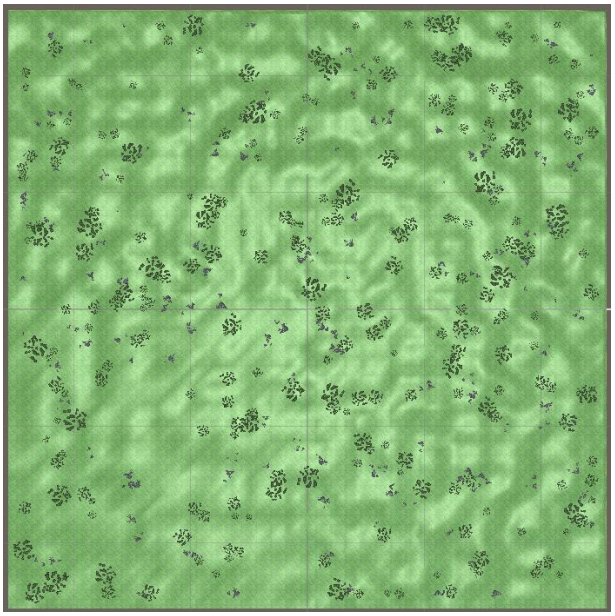
Ekspertinis vizualinių rezultatų vertinimas

Kiekvienai triukšmo funkcijos ir resursų išdėstymo strategijos kombinacijai parinktas vienas iš trijų sugeneruotų žemėlapių. Šis žemėlapis aprašytas įvertinant bendrą estetiką, organiškumą, resursų pasiskirstymo natūralumą ir aiškumą.



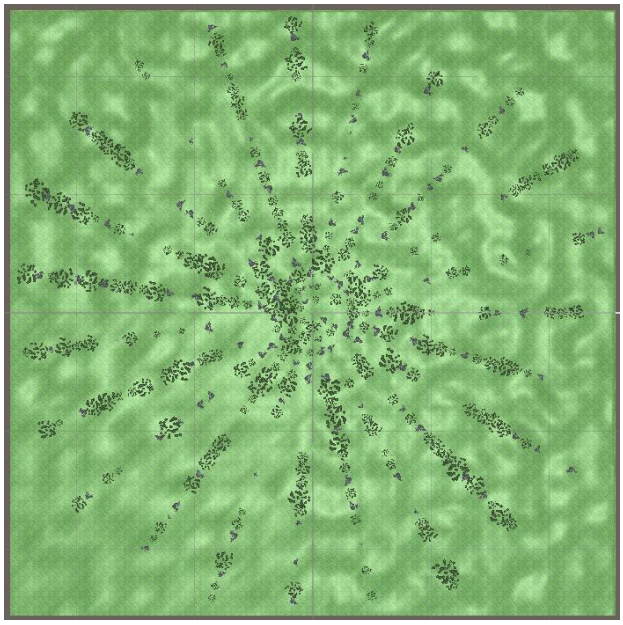
28 pav. Žemėlapis, sugeneruotas naudojant Perlin triukšmo funkciją ir tinklelinę resursų išdėstymo strategiją.

28 pav. matoma Perlin triukšmo funkcijos ir tinklelinės resursų išdėstymo strategijos kombinacija pasižymi tolygiu, tačiau dirbtinu resursų išdėstymu. Resursų dėstymas pagal nematomą tinklelį užtikrina aukštą žaidimo balansą, tačiau vizualiai žemėlapis atrodo dirbtinai. Estetinis patrauklumas – žemas, tačiau aukštas balanso lygis užtikrina žaidybinį efektyvumą.



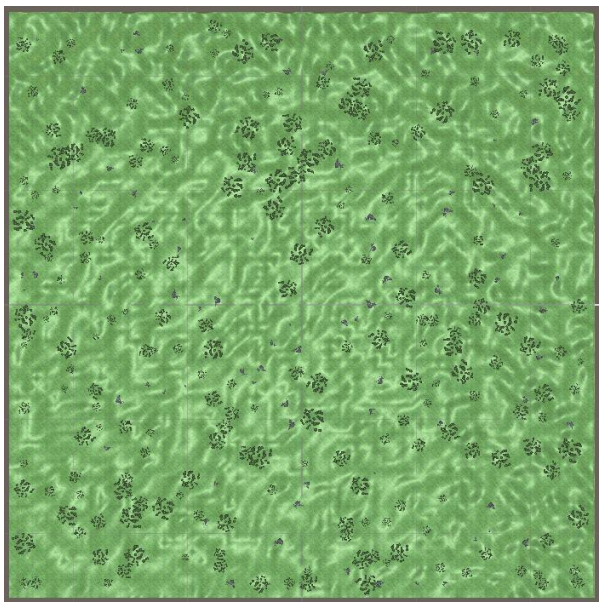
29 pav. Žemėlapis, sugeneruotas naudojant Perlin triukšmo funkciją ir klasterinę resursų išdėstymo strategiją.

29 pav. matoma Perlin triukšmo ir klasterinės resursų išdėstymo strategijos kombinacija padaro, kad resursų grupės atrodytų natūraliai. Perlin triukšmo pagalba išgaunamos perėjimo zonos tarp resursų telkinių, todėl žemėlapis atrodo labiau realistiškas ir vizualiai malonus. Ši kombinacija turi aukštą estetinį ir funkcinį įvertinimą.



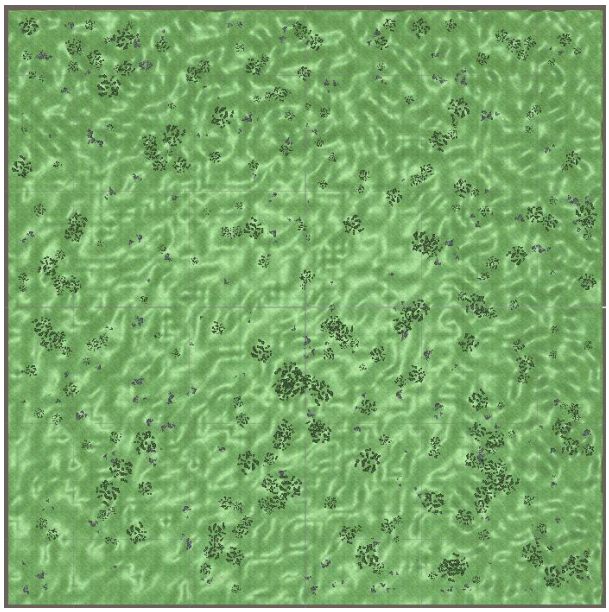
30 pav. Žemėlapis, sugeneruotas naudojant Perlin triukšmo funkciją ir spindulinę resursų išdėstymo strategiją.

30 pav. matoma Perlin triukšmo ir spindulinės resursų išdėstymo strategijos kombinacija vizualiai atrodo labai įdomiai. Nors žemėlapio kraštuose matomas netolygus resursų pasiskirstymas, šio tipo žemėlapis galėtų būti naudojamas strateginio žaidimo scenarijuje, kuriame plėtra vyksta iš kraštų link vidurio. Šio tipo žemėlapis taip pat tiktų chaotiškiems arba magiškiems strateginių žaidimų scenarijams.



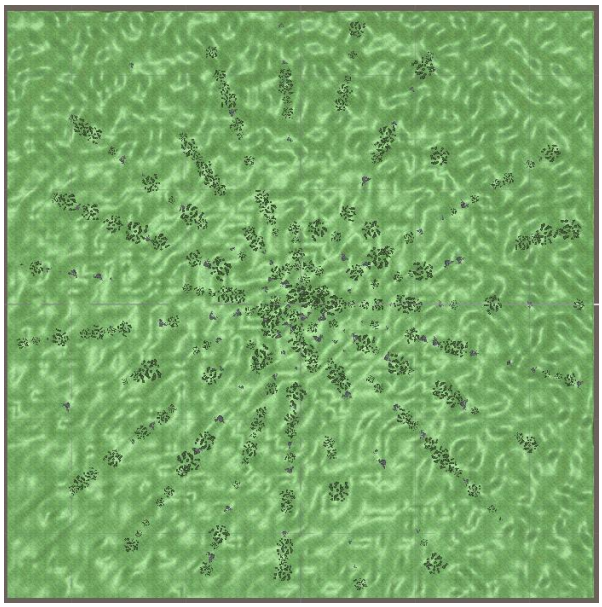
31 pav. Žemėlapis, sugeneruotas naudojant Simplex triukšmo funkciją ir tinklelinę resursų išdėstymo strategiją.

31 pav. matoma Simplex triukšmo funkcijos ir tinklelinės resursų išdėstymo strategijos kombinacija, kurioje pastebimas didesnis detalumas, lyginant su Perlin triukšmu generuotais reljefais. Šioje kombinacijoje atsiranda disonansas tarp reljefo detalumo ir resursų išdėstymo pagal nematomą tinklelį – vaizdas perkrautas, o žaidimo metu gali pasidaryti sunku reljefe išskirti reikšmingus žaidime dalyvaujančius objektus.



32 pav. Žemėlapis, sugeneruotas naudojant Simplex triukšmo funkciją ir klasterinę resursų išdėstymo strategiją.

32 pav. matoma Simplex triukšmo funkcijos ir tinklelinės resursų išdėstymo strategijos kombinacija išsiskiria kaip viena geriausių. Simplex generuoja detalų reljefą, o klasterinis išdėstymas suteikia natūraliai pasiskirstančių resursų jausmą, ko pasekoje gaunamas estetinis balansas tarp chaosiško ir organizuoto, tikroviško žemėlapio.



33 pav. Žemėlapis, sugeneruotas naudojant Simplex triukšmo funkciją ir spinduline resursų išdėstymo strategiją.

33 pav. matoma Simplex triukšmo ir spindulinės resursų išdėstymo strategijos kombinacija, kaip ir Perlin triukšmo ir spindulinės strategijos kombinacija, vizualiai atrodo labai įdomiai – pasižymi neįprastu ir nenatūraliu resursų išdėstymu. Ši kombinacija taip pat galėtų būti taikoma specifiniams strateginių žaidimų scenarijams.

3.3.9. Tyrimo apribojimai

Atliktame tyrime atliekamas išsamus procedūrinio žemėlapių generavimo ir resursų išdėstymo algoritmų palyginimas, tačiau egzistuoja apribojimai, galintys turėti įtakos rezultatų patikimumui ir jų pritaikymui didesnėje skalėje:

1. Atsitiktinumo įtaka – visi žemėlapiai generuoti naudojant kai kuriuos atsitiktinius kintamuosius, kad eksperimentai turėtų skirtingus rezultatus, tačiau taip prarandamas rezultatų atkuriamumas. Naudojant fiksuotus kintamuosius būtų galima užtikrinti tikslų rezultatų atkuriamumą.
2. Ribotas testavimo eksperimentų kiekis – nors kiekviena triukšmo funkcijos ir resursų išdėstymo strategijos kombinacija buvo testuota po tris kartus, norint pilnai užtikrinti statistinį rezultatų patikimumą, šis skaičius turėtų būti didesnis.
3. Vienodi resursų skaičiai sugeneruotuose žemėlapiuose – nors vienodas resursų skaičius užtikrina eksperimentų objektyvumą, realiose situacijose šie skaičiai būna dinamiški.
4. Subjektyvumas vizualinėje rezultatų apžvalgoje – nors didžioji dalis rezultatų buvo objektyviai apibrėžti, vizualinė sugeneruotų žemėlapių rezultatų apžvalga išlieka subjektyvi, o skirtingi tyrėjai gali skirtingai vertinti žemėlapių estetiškumą, įdomumą ir natūralumą.
5. Našumo rezultatų skirtumas kitose sistemose – tyrimo eksperimentai atlikti naudojant vieną techninę sistemos konfigūraciją, todėl skirtingose sistemose našumo eksperimentų rezultatai gali skirtis.
6. Ribotas teritorijos sudėtingumas – eksperimentų metu naudotas 512 x 512 dydžio reljefas be sudėtingesnių geografinių objektų – upių, kalnų, slėnių, ežerų, todėl nors šis reljefas yra tinkamas eksperimentiniam testavimui, neparodoma kaip algoritmai elgtųsi dar labiau realistiškose aplinkose.

3.3.10. Pasiūlymai

Atlikus procedūrinio žemėlapio generavimo eksperimentinius tyrimus, siūloma individualiai pasirinktinai naudoti vieną ar kelias iš ištestuotų triukšmo funkcijų ir resursų išdėstymo strategijų pagal norimus pasiekti rezultatus. Realistiškiems žemėlapiams generuoti siūloma taikyti Perlin triukšmo funkciją kartu su klasterine išdėstymo strategija. Aukšta balansą turintiems žemėlapiams generuoti siūloma taikyti Perlin triukšmo funkciją kartu su tinkleline resursų išdėstymo strategija. Žemėlapiuose, kuriuose norima, kad reljefo formų ir išdėstytų resursų kombinacija turėtų didelę įtaką žaidimo eigai, siūloma naudoti Perlin triukšmo funkcijos ir spindulinę resursų išdėstymo strategiją arba Simplex triukšmo funkciją ir spindulinę resursų išdėstymo strategiją.

3.3.11. Tyrimo išvados

Atlikus 18 eksperimentų, kurių metu buvo naudojamos Perlin ir Simplex triukšmo funkcijų ir tinklelinės, klasterinės ir spindulinės resursų išdėstymo strategijų kombinacijos, buvo empiriškai patikrintos iškeltos hipotezės ir padarytos išvados:

H1: Žemėlapis, sugeneruotas Simplex triukšmo funkcija pasižymės didesniu reljefo tolygumu nei Perlin triukšmas.

Ši hipotezė nepasitvirtino. Nors Simplex triukšmo generuojami reljefai pasižymi didesniu detalumo lygiu, vizualiai reljefo vientisumas buvo blogesnis dėl per didelio triukšmingumo. Remiantis panaudojamo reljefo procentais, Simplex triukšmo funkcija nusileido Perlin – 74.5% prieš 96.9%, o tai rodo, kad Simplex generuoja statesnius šlaitus ir turi daugiau nepasiekiamų zonų.

H2: Tinklelinė resursų išdėstymo strategija užtikrins labiausiai balansuotą pasiskirstymą.

Ši hipotezė pasitvirtino. Tinklelinė resursų išdėstymo strategija eksperimentų metu generavo resursus, kurie turėjo mažiausius standartinius nuokrypius – 0.26-0.29, ir didžiausius resursų klasteriavimo koeficientus – 16.3-17.8. Todėl galima teigti, kad ši strategija geriausiai užtikrina tolygų ir subalansuotą resursų pasiskirstymą sugeneruotame reljefe.

H3: Simplex triuškmo funkcijos ir tinklelinės resursų išdėstymo strategijos kombinacija pasieks geriausius rezultatus generavimo laiko, vizualinio vientisumo ir žemėlapio balanso srityse.

Ši hipotezė nepasitvirtino. Nors ši kombinacija pasižymėjo greitu resursų generavimo laiku – 10-13 milisekundžių ir neblogu resursų balansu, reljefo generavimo laikas – 82-83 milisekundės, buvo 3 kartus didesnis nei Perlin – 26 milisekundės, o vizualinis aspektas nebuvo geras – reljefas atrodė mažiau natūralus ir vientisas. Pastebėta, kad žemėlapiai generuoti naudojantis Simplex triukšmo funkcija dažniau turėjo nepasiekiamų vietų dėl statesnių reljefo šlaitų.

Perlin triukšmo funkcija, nors ir senesnė nei Simplex, labiau tinkama strateginių žaidimų procedūriniam reljefo generavimui, kai tikslas yra išlaikyti žemėlapio vizualinį natūralumą ir didesnį pasiekiamo paviršiaus plotą.

Tiklelinė resursų išdėstymo strategija, nepriklausomai nuo reljefo generacijai naudotos triukšmo funkcijos, duoda didžiausią resursų balansą ir užtikrina strateginio žaidimo sąžiningumą.

3.4. Kelio radimo algoritmo tyrimas

3.4.1. Įžanga į tyrimą

Kelio radimo algoritmai – vieni svarbiausių DI metodų, taikomų strateginiuose žaidimuose. Nuo jų priklauso žaidėjo valdomų ir automatizuotų žaidimo veikėjų jūdėjimas it jo efektyvumas žaidimo aplinkoje, o šių metodų sutrikimai ar nelogiškas veikimas turi itin didelę įtaką žaidėjo patirčiai žaidžiant žaidimą. Strateginių žaidimų žemėlapiai dažniausiai yra pilni įvairiausių kliūčių, tokių kaip – reljefo objektai, pastatai ar kiti veikėjai, todėl būtina nuodugniai apsvarstyti apie naudojamus kelio radimo algoritmus, ir jų galimybes įveikti šias kliūtis randant optimaliausią kelią iš taško A į tašką B. [35][36]

Tyrimo metu buvo įvertinta sukurto metodo struktūra ir efektyvumas, rezultatai žaidimo aplinkoje, ir atlikti palyginimai su Unity žaidimų variklio siūlomu NavMesh sprendimu. Šis tyrimas leido įvertinti A* algoritmu pagrįsto DI metodo veikimą dinamiškoje aplinkoje ir vertę lyginant su prieinamu Unity žaidimų variklio sprendimu.

3.4.2. Įgyvendinimas

Įgyvendintas ir analizuotas A* (angl. A-star) principu paremtas kelio radimo metodas. A* algoritmas, plačiai pripažintas kaip vienas efektyviausių paieškos algoritmų, plačiai naudojamų strateginiuose

žaidimuose, buvo pritaikytas žaidimo prototipo aplinkai, kurioje kelio nustatymas turi būti dinaminis, kadangi ji pastoviai keičiasi – statomi nauji pastatai, atsiranda nauji žaidimo veikėjai, keičiasi esamų žaidimo veikėjų vieta žemėlapyje.

```
Method FindPath(startPos, targetPos):
    startNode = get node from startPos
    targetNode = get node from targetPos

    openSet = list containing startNode
    closedSet = empty set

    While openSet not empty:
        current = node in openSet with lowest fCost (if tie, lowest hCost)
        remove current from openSet
        add current to closedSet

        If current is targetNode:
            return path from startNode to targetNode via parent links

        For each neighbor of current:
            If neighbor not walkable or in closedSet, skip
            newCost = current.gCost + distance(current, neighbor)

            If newCost < neighbor.gCost or neighbor not in openSet:
                neighbor.gCost = newCost
                neighbor.hCost = distance(neighbor, targetNode)
                neighbor.parent = current

                If neighbor not in openSet:
                    add neighbor to openSet

    Return empty path
```

34 pav. Žemėlapio tinklelio sistemos pagrindinio metodo pesudo kodas.

Dinamiškumui ir realaus laiko kelių skaičiavimo galimybei užtikrinti buvo įgyvendinta žemėlapio tinklelio sistema, kurios pagrindinis metodas matomas **34 pav.**, leidžianti pastoviai žinoti pasiekiamas žaidimo zonas ir atnaujinant šį tinklelį statant ir naikinant pastatus, veikėjams keičiant pozicijas. Kartu su šiuo algoritmu buvo įgyvendintas veikėjų jūdėjimo mechanizmas ir papildoma elgsenos logika, leidžianti veikėjams pulti priešininkus ir gintis.

3.4.3. Tyrimo tikslai ir uždaviniai

Tyrimo tikslas – įvertinti A* algoritmu grįsto DI metodo veikimą dinamiškoje strateginio žaidimo aplinkoje ir palyginti jo pritaikymo tinkamumą su Unity NavMesh sprendimu.

Uždaviniai:
1. Sukurti ir integruoti į strateginio žaidimo prototipą A* pagrįstą kelio radimo algoritmą.
2. Pritaikyti Unity NavMesh sistemą strateginio žaidimo prototipui ir užtikrinti jos veikima analogiškuose žaidimo scenarijuose.
3. Implementuoti A* pagrindu sukurto algoritmo gebėjimą realiu laiku teisingai nustatyti kliūtis 3D žaidimo aplinkoje.
4. Apibrėžti ir taikyti objektyvius vertinimo kriterijus kelio radimo algoritmų tyrimui – kelio radimo laikas, kelio ilgis, sėkmingai surastų kelių procentas ir reagavimo į naujas kliūtis greitis.
5. Eksperimentiškai įvertinti A* pagrindu sukurto algoritmo ir Unity NavMesh rezultatus naudojant vienodas sąlygas – žemėlapio struktūrą, kliūtis, atstumus iki tikslo.
6. Aptarti abiejų metodų pranašumus, trūkumus ir pritaikomumą strateginių žaidimų, turinčių dinamiškai kintančias aplinkas, kurime.

3.4.4. Tyrimo hipotezės

Atlikus šio DI metodo analizę (1 skyrius), iškeltos šios hipotezės:

H1: Esant vienodoms sąlygoms, A* algoritmu grįstas metodas suranda trumpesnius maršrutus nei Unity NavMesh sistema.

H2: Unity NavMesh sistema randa kelius greičiau nei A* algoritmu grįstas metodas.

H3: A* algoritmu grįstas metodas greičiau prisitaiko prie dinamiškai kintančios žaidimo aplinkos lyginant su Unity NavMesh sistema.

3.4.5. Hipotezių patikrinimo metodika

Hipotezės bus tikrinamos atliekant eksperimentus, kurių metu bus testuojami du kelio radimo būdai – A* algoritmu grįstas metodas ir Unity NavMesh sistema. Eksperimentai bus atliekami vienodomis sąlygomis, atliekant kelio paiešką tarp taškų įvairiuose, iš anksto apibrėžtuose žaidimo scenarijuose.

Eksperimentinio testavimo scenarijai:
- Tuščias žemėlapis be jokių kliūčių.
- Žemėlapis su statinėmis kliūtimis.
- Žemėlapis su dinamiškai atsirandančiomis kliūtimis.

Vertinimo kriterijai:
1. Kelio ilgis – parodo, apskaičiuoto kelio atstumą Unity žaidimų variklio matavimo vienetais, skaitomus kaip metrai.
2. Kelio generavimo laikas milisekundėmis – parodo, kiek laiko trunka kelio radimas.
3. Prisitaikymo prie dinamiškai kintančios aplinkos laikas milisekundėmis – vertina, kaip greitai metodai reaguoja į dinaminius pasikeitimus žemėlapyje.

Eksperimentų eiga:
1. Testavimo scenarijai paleidžiami po 10 kartų kiekvienam metodui.
2. Atliekami skaičiavimai pagal aukščiau aprašytus vertinimo kriterijus
3. Kiekvieno eksperimento rezultatų duomenys pateikiami lentelėmis.

Atlikus eksperimentus, rezultatai bus lyginami tarpusavyje, kad patikrinti iškeltas hipotezes.

3.4.6. Tyrimo eiga

Tyrimo metu buvo atlikta abiejų kelio radimo metų – A* algoritmu grįsto metodo ir Unity NavMesh eksperimentai. Iš viso atlikta 30 eksperimentų, kadangi kiekvienam testavimo scenarijui – tuščio žemėlapio, žemėlapio su statiniais objektais ir žemėlapio su dinamiškai atsirandančiais objektais, buvo atlikta po 10 eksperimentų. Eksperimentų metu apskaičiuotos ir fiksuotos vertinimo kriterijų reikšmės.

Kriterijų reikšmių skaičiavimas

1. Kelio ilgis skaičiuotas kaip visų kelio taškų porų atstumo suma, pagal formulę, kur P_i – dabartinis taškas, P_{i+1} – sekantis taškas, dist() – atstumo formulė:

$$k = \sum_{L=}^{n-1} dist(P_i, p_{i+1})$$

2. Kelio radimo laikas, skaičiuotas C# Stopwatch klasės pagalba, pradedant skaičiuoti laiką kelio radimo pradžioje, o baigiant skaičiuoti pabaigoje.

3. Prisitaikymo prie dinamiškai kintančios aplinkos laikas skaičiuotas kaip skirtumas tarp metodo atsinaujinimo ir kliūties atsiradimo C# Stopwatch klasės pagalba.

Tuščio žemėlapio scenarijus

Kiekvieno eksperimento metu parinktos atsitiktinės pradinių ir galutinių taškų poros žemėlapyje, kuriame nėra daugiau jokių objektų. Abiems kelio radimo metodams paduodamų pradinių ir galutinių taškų poros buvo vienodos.

Žemėlapio su statiniais objektais scenarijus

Kiekvieno eksperimento metu parinktos atsitiktinės pradinių ir galutinių taškų poros žemėlapyje, kuriame yra iš anksto padėtų objektų, blokuojančių kelio trajektorijas. Abiems kelio radimo metodams paduodamų pradinių ir galutinių taškų poros buvo vienodos.

Žemėlapio su dinaminiais objektais scenarijus

Kiekvieno eksperimento metu parinktos atsitiktinės pradinių ir galutinių taškų poros žemėlapyje. Kelio eigoje, buvo pastatytas pastatas, blokuojantis kelio trajektoriją ir buvo žiūrėta, ar kelio radimo metodai sureagavo į pokytį ir pakeitė kelią. Abiems kelio radimo metodams paduodamų pradinių ir galutinių taškų poros buvo vienodos.

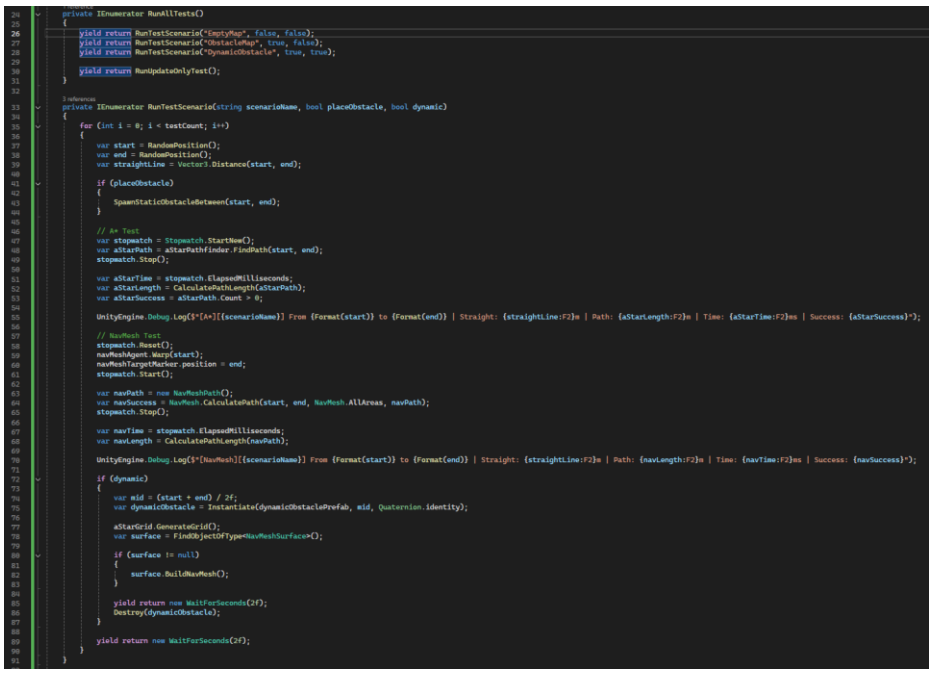
Duomenų rinkimas

Kiekvieno eksperimento rezultatai buvo matomi Unity žaidimų variklio konsolės lange ir vėliau surinkti ir atvaizduoti lentelėmis. Gauti rezultatai atvaizduoti rezultatų poskyryje.

Eksperimentų metu gauti duomenys buvo analizuojami, kad patikrinti tyrimo hipotezes ir įvertinti kiekybinį ir kokybinį metodų efektyvumą.

3.4.7. Tyrimo metu naudota programinė įranga

Visi kelio radimo metodų eksperimentai vyko Unity žaidimų variklio aplinkoje, naudojantis palyginimams sukurtais testiniais metodais.



```
private IEnumerator RunAllTests()
{
    yield return RunTestScenario("EmptyMap", false, false);
    yield return RunTestScenario("ObstacleMap", true, false);
    yield return RunTestScenario("DynamicObstacle", true, true);

    yield return RunUpdateOnlyTest();
}

3 references
private IEnumerator RunTestScenario(string scenarioName, bool placeObstacle, bool dynamic)
{
    for (int i = 0; i < testCount; i++)
    {
        var start = RandomPosition();
        var end = RandomPosition();
        var straightLine = Vector3.Distance(start, end);

        if (placeObstacle)
        {
            SpawnStaticObstacleBetween(start, end);
        }

        // A* Test
        var stopwatch = Stopwatch.StartNew();
        var aStarPath = aStarPathfinder.FindPath(start, end);
        stopwatch.Stop();

        var aStarTime = stopwatch.ElapsedMilliseconds;
        var aStarLength = CalculatePathLength(aStarPath);
        var aStarSuccess = aStarPath.Count > 0;

        UnityEngine.Debug.Log($"[A*][{scenarioName}] From {Format(start)} to {Format(end)} | Straight: {straightLine:F2}m | Path: {aStarLength:F2}m | Time: {aStarTime:F2}ms | Success: {aStarSuccess}");

        // NavMesh Test
        stopwatch.Reset();
        navMeshAgent.Warp(start);
        navMeshTargetMarker.position = end;
        stopwatch.Start();

        var navPath = new NavMeshPath();
        var navSuccess = NavMesh.CalculatePath(start, end, NavMesh.AllAreas, navPath);
        stopwatch.Stop();

        var navTime = stopwatch.ElapsedMilliseconds;
        var navLength = CalculatePathLength(navPath);

        UnityEngine.Debug.Log($"[NavMesh][{scenarioName}] From {Format(start)} to {Format(end)} | Straight: {straightLine:F2}m | Path: {navLength:F2}m | Time: {navTime:F2}ms | Success: {navSuccess}");

        if (dynamic)
        {
            var mid = (start + end) / 2f;
            var dynamicObstacle = Instantiate(dynamicObstaclePrefab, mid, Quaternion.identity);

            aStarGrid.GenerateGrid();
            var surface = FindObjectOfType<NavMeshSurface>();

            if (surface != null)
            {
                surface.BuildNavMesh();
            }

            yield return new WaitForSeconds(2f);
            Destroy(dynamicObstacle);
        }

        yield return new WaitForSeconds(2f);
    }
}
```

35 pav. Kelio radimo metodų palyginimams sukurti metodai.

35 pav. matomi sukurti testavimo metodai, naudojami visiems kelio radimo eksperimentams vykdyti. Kiekvienam iš trijų testavimo scenarijų – tuščio žemėlapio, žemėlapio su statine kliūtimi ir žemėlapiui su dinamiškai atsirandančia kliūtimi buvo vykdoma 10 testų su atsitiktiniai pradiniais ir galutiniais taškais. Pagal testavimo scenarijų buvo parenkamas statinis arba dinaminis kliūties pridėjimas.

```
1 reference
private IEnumerator RunUpdateOnlyTest()
{
    for (int i = 0; i < testCount; i++)
    {
        var randomPos = RandomPosition();
        GameObject obstacle = Instantiate(dynamicObstaclePrefab, randomPos, Quaternion.identity);

        var stopwatch = Stopwatch.StartNew();
        aStarGrid.GenerateGrid();
        stopwatch.Stop();

        UnityEngine.Debug.Log($"[A*][GridUpdate] Update at {Format(randomPos)} | Time: {stopwatch.ElapsedMilliseconds}ms");

        stopwatch.Reset();
        var surface = FindObjectOfType<NavMeshSurface>();

        stopwatch.Start();
        surface.BuildNavMesh();
        stopwatch.Stop();

        UnityEngine.Debug.Log($"[NavMesh][Update] Update at {Format(randomPos)} | Time: {stopwatch.ElapsedMilliseconds}ms");

        Destroy(obstacle);
        yield return new WaitForSeconds(0.2f);
    }
}
```

36 pav. Kelio radimo metoduose naudojamų tinklelių testavimui sukurtas metodas.

Kelio radimo metoduose naudojamų tinklelių palyginimui buvo sukurtas testinis metodas, matomas **36 pav.**, matuojantis kaip greitai tinkleliai atsinaujina pridėjus naują objektą žemėlapyje.

3.4.8. Tyrimo rezultatai

Tyrimo rezultatai išskiriami į 2 dalis – tinklelio atnaujinimo našumo rezultatai ir kelio radimo efektyvumo rezultatai. Bendros visų rezultatų lentelės, pateiktos 2 ir 3 prieduose, buvo išskirstytos į atskiras lenteles pagal tiriamus aspektus.

Tinklelių atnaujinimo našumo rezultatai

Tinkelių atnaujinimo atsiradus naujam objektui našumo rezultatai pateikiami lentelės pavidalu. Dešimties eksperimentų metu palygintas tinklelio atnaujinimo laikas atsitiktinėje vietoje atsiradus naujam objektui.

3 lentelė. Tinklelių atnaujinimo našumo rezultatai.

Kelio radimo metodo tinklelis	**Eksperimento numeris**	**Objekto atsiradimo vieta**	**Tinklelio atnaujinimo laikas (ms)**
A*	1	(17,0, -1,6)	12,64ms
Unity NavMesh			23,50ms
A*	2	(-4,5, -7,7)	15,47ms
Unity NavMesh			37,35ms
A*	3	(10,2, -16,6)	13,13ms
Unity NavMesh			27,90ms
A*	4	(-12,7, 14,6)	13,48ms
Unity NavMesh			18,43ms
A*	5	(-19,4, 17,3)	10,33ms
Unity NavMesh			21,05ms
A*	6	(-6,2, -13,7)	15,62ms
Unity NavMesh			21,10ms
A*	7	(-0,4, 19,8)	11,62ms
Unity NavMesh			37,73ms
A*	8	(-15,3, 4,5)	15,32ms
Unity NavMesh			23,24ms
A*	9	(-18,2, -15,0)	10,38ms
Unity NavMesh			24,09ms
A*	10	(-17,7, -17,1)	15,58ms
Unity NavMesh			39,43ms

Remiantis 3 lentelėje pateiktais eksperimentiniais duomenimis, matoma, kad A* tinklelio atnaujinimo greitis buvo žymiai greitesnis ir stabilesnis nei Unity NavMesh. A* tinklelio atnaujinimas vidutiniškai truko apie 13 milisekundžių, o Unity NavMesh atnaujinimo laikas svyravo tarp 18 ir 39 milisekundžių. Šie rezultatai parodo, kad A* metodo tinklelis greičiau prisitaiko prie žemėlapio pokyčių, kas yra kritiškai svarbu norint užtikrinti stabilią žaidimo patirtį ir aukštą žaidimo kokybę.

Kelio radimo metodų efektyvumo rezultatai

Kelio radimo metodų efektyvumo eksperimentai buvo vykdomi 3 scenarijais – tuščiame žemėlapyje, žemėlapyje su statinėmis kliūtimis ir žemėlapyje su dinamiškai atsirandančiomis kliūtimis.

Eksperimentų metu gauti rezultatai pateikiami lentelių pavidalu (4-6 lentelės). Svarbu paminėti, kad į kelio radimo laikus nėra įskaičiuotas tinklelio pergeneravimo laikas.

4 lentelė. Kelio radimo metodų efektyvumo tuščiame žemėlapyje eksperimentų rezultatai.

Kelio radimo metodas	**Eksperimento numeris**	**Pradinio taško koordinatės**	**Galutinio taško koordinatės**	**Tiesus atstumas**	**Apskaičiuoto kelio ilgis**	**Kelio radimo laikas**
A*	1	(-11,9, -2,5)	(13,0,16,1)	31,05m	35,10m	17,31ms
NavMesh					38,57m	9,26ms
A*	2	(-10,0, -19,2)	(15,4,-16,4)	25,58m	28,79m	17,13ms
NavMesh					30,53m	6,63ms
A*	3	(15,7, -2,2)	(-12,8,1,1)	28,69m	32,19m	16,94ms
NavMesh					35,16m	9,75ms
A*	4	(6,3, -14,6)	(14,0,-8,6)	9,74m	10,88m	8,84ms
NavMesh					11,70m	8,30ms
A*	5	(3,2, 2,2)	(-16,9,-18,8)	29,04m	35,33m	15,75ms
NavMesh					35,68m	6,00ms
A*	6	(8,0, 16,0)	(-16,5,15,7)	24,48m	29,09m	12,80ms
NavMesh					28,48m	9,39ms
A*	7	(13,4, -11,4)	(16,3,16,5)	28,05m	32,32m	14,39ms
NavMesh					31,68m	12,19ms
A*	8	(-5,1, -0,9)	(-8,1,-8,6)	8,24m	9,77m	8,12ms
NavMesh					10,05m	10,76ms
A*	9	(14,4, 16,5)	(-13,9,-2,4)	33,96m	41,78m	14,44ms
NavMesh					37,67m	11,19ms
A*	10	(-18,5, 8,8)	(4,4,-13,9)	32,27m	39,13m	10,03ms
NavMesh					38,53m	10,80ms

Iš 4 lentelėje pateiktų eksperimentinių rezultatų matoma, kad žemėlapyje neturinčiame jokių kliūčių, A* algoritmo pagrindu veikiantis metodas ir Unity NavMesh randa kelius keliais metrais ilgesnius nei trumpiausias, tiesus atstumas tarp pradinio ir galutinio taško. Vidutiniškai, A* kelio radimo metodas rodė geresnius rezultatus nei Unity NavMesh, tačiau dauguma atvejų trukdavo žymiai ilgiau.

5 lentelė. Kelio radimo metodų efektyvumo, žemėlapyje su statine kliūtimi, eksperimentų rezultatai.

Kelio radimo metodas	**Eksperimento numeris**	**Pradinio taško koordinatės**	**Galutinio taško koordinatės**	**Tiesus atstumas**	**Apskaičiuoto kelio ilgis**	**Kelio radimo laikas**
A*	1	(-0,7, 11,1)	(-19,9,-5,5)	25,33m	34,19m	19,70ms
NavMesh					33,54m	11,09ms
A*	2	(5,4, 2,8)	(-11,5,-7,0)	19,50m	24,61m	17,83ms
NavMesh					25,39m	12,34ms

A*	3	(17,9, -8,4)	(14,0,19,3)	28,03m	39,20m	16,63ms
NavMesh					35,71m	11,06ms
A*	4	(2,4, -6,7)	(-5,8,-8,0)	8,31m	10,93m	19,50ms
NavMesh					10,78m	15,98ms
A*	5	(9,0, -2,5)	(7,9,-11,7)	9,28m	11,92m	16,18ms
NavMesh					12,17m	12,50ms
A*	6	(-12,5, -3,2)	(-8,4,-16,7)	14,09m	19,68m	14,90ms
NavMesh					19,12m	14,95ms
A*	7	(-9,1, 17,3)	(4,9,-10,1)	30,77m	38,97m	20,34ms
NavMesh					42,85m	12,08ms
A*	8	(5,2, -14,4)	(2,8,-13,6)	2,55m	3,57m	17,46ms
NavMesh					3,39m	12,24ms
A*	9	(13,8, -15,7)	(-3,7,19,2)	38,99m	54,44m	15,91ms
NavMesh					53,81m	15,61ms
A*	10	(-1,8, 18,1)	(-5,5,16,6)	3,99m	5,41m	19,08ms
NavMesh					5,39m	13,90ms

Iš 5 lentelėje pateiktų eksperimentinių rezultatų matoma, kad žemėlapyje, turinčiame statinių kliūčių, A* ir Unity NavMesh sėkmingai sugebėjo naviguoti aplink kelyje esančią kliūtį. Tačiau skirtumas tarp Unity NavMesh ir A* metodo aiškiai matomas apskaičiuoto kelio ir kelio radimo laiko rodikliuose – Unity NavMesh pranoko A* metodą. Nors A* veikė patikimai, jis ne visada rado optimaliausią kelią aplink statines kliūtis, todėl Unity NavMesh yra efektyvesnis šio scenarijaus atveju.

6 lentelė. Kelio radimo metodų efektyvumo, žemėlapyje su dinamiškai atsirandančia kliūtimi, eksperimentų rezultatai.

Kelio radimo metodas	**Eksperimento numeris**	**Pradinio taško koordinatės**	**Galutinio taško koordinatės**	**Tiesus atstumas**	**Apskaičiuoto kelio ilgis**	**Kelio radimo laikas**
A*	1	(14,8, -11,8)	(7,7,19,0)	31,61m	43,26m	25,39ms
NavMesh					39,48m	15,29ms
A*	2	(7,3, -14,6)	(16,2,10,7)	26,85m	33,55m	23,49ms
NavMesh					31,36m	15,90ms
A*	3	(13,2, -7,4)	(-3,0,-8,1)	16,23m	21,73m	19,91ms
NavMesh					19,28m	13,82ms
A*	4	(5,4, 14,9)	(-11,3,-1,2)	23,21m	31,72m	18,23ms
NavMesh					27,05m	14,92ms
A*	5	(18,4, -5,1)	(-2,7,13,7)	28,28m	36,82m	26,00ms
NavMesh					34,18m	16,32ms
A*	6	(10,5, -14,3)		2,44m	3,29m	19,26ms

NavMesh			(11,6,-16,4)		2,97m	14,32ms
A*	7	(16,0, -14,9)	(-14,8,13,7)	41,95m	56,79m	22,20ms
NavMesh					51,13m	14,01ms
A*	8	(7,4, 10,0)	(-4,4,-5,1)	19,13m	23,15m	17,61ms
NavMesh					21,80m	13,14ms
A*	9	(12,5, 12,4)	(-19,4,-2,8)	35,27m	44,56m	21,58ms
NavMesh					43,97m	11,53ms
A*	10	(9,6, -5,1)	(-1,2,-7,5)	11,03m	14,70m	18,50ms
NavMesh					13,24m	11,44ms

Iš 6 lentelėje pateiktų eksperimentinių rezultatų matoma, kad scenarijuose, kur kliūtys atsiranda dinamiškai, Unity NavMesh pranoko A*. A* generuodavo kelius ilgiau, o patys keliai dauguma atvejų buvo ilgesni. Remiantis šiais rezultatais galima teigti, kad Unity NavMesh yra labiau tinkamas ir statiniuose, ir dinamiškai kintančiuose žemėlapiuose.

3.4.9. Tyrimo apribojimai

Atliktas išsamus A* ir Unity NavMesh kelio radimo metodų palyginimas, tačiau egzistuoja apribojimai, galintys turėti įtakos rezultatų patikimumui ir šių metodų pritaikymui žaidimų kūrime:

1. Dėl laiko ir išteklių ribotumo, A* algoritmo pagrindu įgyvendintas kelio radimo metodas nebuvo pilnai optimizuotas – statiniuose žemėlapiuose su kliūtimis ir dinamiškai kintančiuose žemėlapiuose reikalingos papildomos optimizacijos, kad šis metodas galėtų lygiuotis su Unity NavMesh sistema.
2. Ribotas eksperimentų kiekis kiekvienam testavimo scenarijui galėjo pilnai neatskleisti visų kelio radimo logikos aspektų – dinaminėse aplinkose galėjo atsirasti nukrypimų dėl atsirandančios kliūties pozicijos.
3. Nebuvo įvertinti subjektyvūs kelio radimo metodų aspektai – judėjimo sklandumas ir žaidėjo patirtis naudojant abu kelio radimo metodus.

3.4.10. Pasiūlymai

Atsižvelgiant į gautus eksperimentinių tyrimų rezultatus išryškėja dvi galimos kryptys kelio radimo metodų naudojimo srityje:

1. Paprastiems žaidimams, nereikalajauntiems sudėtingų kelio radimo skaičiavimų ar turintiems pakankamai paprastas, statines aplinkas, siūloma naudoti Unity NavMesh sistemą, dėl jos lengvo įgyvendinimo ir gerų rezultatų eksperimentiniuose scenarijuose.
2. Kompleksiškiems žaidimams, reikalaujantiems sudėtingų kelio radimo skaičiavimų ar turintiems aplinkas su aukštu dinamiškumo lygiu, siūloma savarankiškai optimizuoti pateikto kelio radimo metodo, grįsto A* algoritmu, kodą, kad šis veiktų efektyviau.

3.4.11. Tyrimo išvados

Atlikus 30 eksperimentų, kurių metu buvo naudojami du kelio radimo metodai - A* algoritmu pagrįstas metodas ir Unity NavMesh, ir 10 eksperimentų A* metodo ir Unity NavMesh tinklelių našumui testuoti, buvo empiriškai patikrintos iškeltos hipotezės ir pateiktos šios išvados:

H1: Esant vienodoms sąlygoms, A* algoritmu grįstas metodas suranda trumpesnius maršrutus nei Unity NavMesh sistema.

Ši hipotezė nepasitvirtino. A* algoritmu grįstas kelio radimo metodas trumpesnius kelius rado tik tuščiame žemėlapyje, o statinių ir dinamiškai atsirandančių kliūčių turinčiuose žemėlapiuose, Unity NavMesh surastų kelių atstumai buvo trumpesni. Tai rodo, kad A* algoritmo veikimo efektyvumas mažėja didėjant aplinkos sudėtingumui.

H2: Unity NavMesh sistema randa kelius greičiau nei A* algoritmu grįstas metodas.

Ši hipotezė pasitvirtino. Unity NavMesh sistema pasižymėjo didesniu greičiu ir trumpesnių atstumų radimu, ypač statinių ir dinaminškai atsirandančių kliūčių turinčiuose žemėlapiuose.

H3: A* algoritmu grįstas metodas greičiau prisitaiko prie dinamiškai kintančios žaidimo aplinkos lyginant su Unity NavMesh sistema.

Ši hipotezė pasitvirtino. Abiejų kelio radimo metodų tinklelių atsinaujinimo našumo tyrimas parodė, kad A* metodo tinklelis dauguma atvejų atsinaujino greičiau, o tai reiškia greitesnį prisitaikymą prie dinamiškai kintančių žaidimo aplinkų.

A* kelio radimo metodas rado trumpesnius kelius kliūčių neturinčiuose žemėlapiuose, tačiau tokios situacijos realiose strateginių žaidimų aplinkose pasitaiko labai retai, todėl norint taikyti šį metodą, reikia optimizuoti kliūčių valdymo logiką.

A* pasižymėjo didesniu lankstumu, kadangi kodo atžvilgiu, implementacija yra atvira – ją galima keisti ir pritaikyti daugumoje scenarijų. Tuo tarpu Unity NavMesh sistema yra uždara, o jos plėtimo ir keitimo galimybės – labai ribotos.

3.5. Blackboard sistemos tyrimas

3.5.1. Įžanga į tyrimą

Daugumoje strateginių žaidimų, ekonomikos valdymas yra vienas svarbiausių logikos komponentų. Žmogaus ir DI valdomi žaidėjai privalo balansuoti resursų gamybos ir sunaudojimo procesus, o netinkami veiksmai gali greitai privesti prie žaidimo pralaimėjimo prieš labiau organizuotą priešininką. Todėl automatizuotos DI sistemos, kurios geba savarankiškai priimti strateginius ekonominius sprendimus yra ypač svarbios. Vienas iš DI metodų, leidžiančių pasiekti šį tikslą yra blackboard architektūra, kuri sudaryta iš bendros informacijos bazės ir ekspertinių sistemų, atsakingų už skirtingų žaidimo aspektų valdymą. [39][40]

Blackboard architektūra plačiai taikoma strateginiuose žaidimuose, kuriuose kompiuterio valdomas priešininkas turi priimti koordinuotus sprendimus keliuose žaidimo aspektuose, tokiuose kaip – ekonomika, pastatų statymas, karinės ir ginybinės taktikos naudojimas. Ši architektūra leidžia išskaidyti ekspertines sistemas, atsakingas už sprendimų priėmimą skirtingose žaidimo srityse ir užtikrina, kad jos bendrauja netiesiogiai. Kiekviena ekspertinė sistema papildo bendrą informacijos bazę, gauna iš jos visapusišką informaciją apie esamą žaidimo padėtį ir efektyviai priima reikiamus sprendimus. [39][40]

Tyrimo metu atliekami eksperimentai naudojantis įgyvendinta blackboard sistema, testuojamas jos veikimas įvairiuose scenarijuose, matuojamas šio veikimo našumas ir nustatomas priimtų sprendimų korektiškumas.

3.5.2. Įgyvendinimas

Tyrimui sukurta blackboard architektūra grįsta ekonomikos valdymo ir pastatų statymo sistema, susidedanti iš bendros žinių bazės (blackboard) ir dviejų ekspertinių sistemų – resursų ir statybos. Įgyvendintas DI metodas realiu laiku analizuoja resursų poreikį, atsižvelgia į esamas resursų atsargas ir gamybos apimtis, prognozuoja resursų trūkumus ir vykdo pastatų, skirtų resursų gamybai ir sandėliavimui, statybas.

Resursų ekspertinės sistemos supaprastintos logikos pseudo kodas, pavaizduotas **37 pav.**

```
Method GetMostUrgentResource():
    For each resource type:
        amount = get current amount
        capacity = get current capacity
        ratio = if capacity > 0 then amount / capacity else 0
        If ratio >= 0.9:
            Return null

    Initialize production dictionary with zeroes for all resource types
    For each resource producer in scene:
        For each production in producer:
            production[resourceType] += productionAmount

    Initialize demand dictionary with 1.0 for all resource types
    For each building prefab:
        If building or cost data is invalid, skip
        For each cost in building:
            demand[resourceType] += cost amount

    Initialize score dictionary
    For each resource type:
        prod = production[resourceType] or 0
        stock = get current amount
        demandVal = demand[resourceType] or 1.0
        score = (prod + 1) / ((stock + 1) * demandVal)
        scoreDictionary[resourceType] = score

    weakestResource = null
    lowestScore = max float
    For each resource in scoreDictionary:
        If score < lowestScore:
            lowestScore = score
            weakestResource = resource

    Return weakestResource
```

37 pav. Resursų ekspertinės sistemos pseudo kodas.

Statybos ekspertinės sistemos, supaprastintos logikos pseudo kodas, pavaizduotas **38 pav.**

```
Method ExecuteBuildingLogic(resourceOrCommand):
    If resourceOrCommand is a resource type:
        type = resolve building type from resource
    Else if resourceOrCommand is "warehouse":
        type = Warehouse
    Else:
        return

    prefab = get prefab for type
    If prefab is null, return

    If cannot afford prefab's cost, return

    Enqueue type into buildQueue

    If not isPlacing:
        isPlacing = true
        While buildQueue not empty:
            type = dequeue next
            prefab = get prefab for type
            If prefab is null, continue
            If cannot afford prefab's cost, continue
            Spend resources
            radius = sqrt(buildCount + 1) * 5
            attempts = maxPlacementAttempts
            Repeat:
                angle = random angle
                distance = random from 2 to radius
                position = startPos offset by angle and distance
                If position is free (no collisions in "Building" layer), break
                attempts -= 1
            Until valid position found or attempts exhausted

            If no valid position found:
                position = startPos

            Instantiate building at position
            If building has resource facility component:
                call ManualInit and OnBuildingPlaced
            Register building
            buildCount += 1
            Wait buildInterval
        isPlacing = false
```

38 pav. Statybų ekspertinės sistemos pseudo kodas.

Žaidimo prototipe įgyvendinti resursų tipai:

- Gold – auksas, skirtas pastatų statymui.
- Wood – medis, skirtas pastatų statymui.
- Stone – akmuo, skirtas pastatų statymui.
- Iron – geležis, skirta kareivių gaminimui ir pastatų statymui.
- Food – maistas, skirtas kareivių gaminimui.

Žaidimo prototipe įgyvendinti pastatų tipai:

- Miesto centras (angl. Townhall) – pagrindinis pastatas, kurį praradus, žaidimas pralaimimas.
- Barakai (angl. Barracks) – pastatas skirtas kareivių gaminimui.
- Sandėlis (angl. Warehouse) – pastatas, skirtas išplėsti resursų talpą.
- Turgus (angl. Market) – pastatas, periodiškai gaminantis visus resursus.
- Ferma (angl. Farm) – pastatas, periodiškai gaminantis maisto ir medžio resursus.
- Kasykla (angl. Mine) – pastatas, periodiškai gaminantis akmens ir geležies resursus.
- Medžių kirtykla (angl. Woodcutter) – pastatas, periodiškai gaminantis medžio resursą.
- Sienos (angl. Walls) – apsauginis pastatas, skirtas aptverti miestą. Veikia kaip ginybinis pastatas, neleidžiantis priešams lengvai praeiti.
- Ginybiniai vartai (angl. Gatehouse) – ginybinis pastatas, leidžiantis praeiti per sienas.
- Didelis bokštas (angl. Large Tower) – pastatas, automatiškai šaudantis į artimiausius priešus. Turi didelį šaudymo nuotolį ir darantis didelę žalą, tačiau šaudantis lėtai.
- Vidutinis bokštas (angl. Mid Tower) – pastatas, automatiškai šaudantis į artimiausius priešus. Turi vidutinį šaudymo nuotolį ir darantis vidutinę žalą, tačiau šaudantis vidutiniškai greitai.
- Mažas bokštas (angl. Small Tower) – pastatas, automatiškai šaudantis į artimiausius priešus. Turi mažą šaudymo nuotolį ir darantis mažiausią žalą, tačiau šaudantis labai greitai.

Šiame tyrime naudoti pastatai, susiję su žaidimo ekonomikos plėtra – sandėlis, turgus, ferma, kasykla ir medžių kirtykla. Kiekvienas pastato statymas reikalauja resursų, kurių kiekis priklauso nuo pastato tipo.

3.5.3. Tyrimo tikslas ir uždaviniai

Tyrimo tikslas – įvertinti įgyvendinto, Blackboard architektūra grįsto, DI metodo veikimą realaus laiko strateginio žaidimo aplinkoje, esant skirtingoms ekonominėms sąlygoms.

Uždaviniai:

1. Sukurti ir integruoti į strateginio žaidimo prototipą blackboard architektūra grįstą DI metodą, automatizuojantį žaidimo ekonomikos valdymą.
2. Apibrėžti ir taikyti objektyvius vertinimo kriterijus blackboard metodo tyrimui.
3. Eksperimentiškai įvertinti blackboard metodo veikimo galimybes pritaikant skirtingus žaidimo scenarijus.
4. Aptarti teigiamus ir neigiamus įgyvendinto metodo aspektus ir įvertinti pritaikomumą realiems strateginiams žaidimams.

3.5.4. Tyrimo hipotezės

Atlikus šio DI metodo analizę (1 skyrius), iškeltos šios hipotezės:

H1: Esant blogoms ekonominėms aplinkybėms - žemam turimų resursų ir saugojimo talpos kiekiui, metodas prioretizuos resursus generuojančių pastatų statymą.

H2: Esant geroms ekonominėms aplinkybėms – dideliam turimų resursų kiekiui, tačiau žemam saugojimo talpos kiekiui, metodas prioretizuos ir pradės statyti resursus talpinančius pastatus.

H3: Esant labai geroms ekonominėms aplinkybėms – dideliam turimų resursų ir saugojimo talpos kiekiui, metodas subalansuotai paskirstys pastatų statybas, išlaikydamas apytiksliai vienodą visų resursų tipų gamybą.

3.5.5. Hipotezių patvirtinimo metodika

Hipotezių patikrinimui sukurti trys skirtingi eksperimentiniai scenarijai, kurių metu blackboard metodas, veiks skirtingose pradinėse ekonomikos situacijose:

- Bloga ekonominė situacija – mažas pradinių resursų ir talpos kiekis.
- Gera ekonominė situacija – didelis pradinių resursų kiekis, tačiau mažas talpos kiekis.
- Labai gera ekonominė situacija – didelis pradinių resursų ir talpos kiekis.

Kiekvienas eksperimentinis scenarijus bus vykdomas 180 sekundžių, o kas 30 sekundžių fiksuojami šie vertinimo kriterijai:

1. Pastatytų pastatų skaičius per laiką.
2. Resursų talpos užpildymo procentas – nuo 0 iki 100%.
3. Kiekvieno resursų tipo – aukso, medžio, akmens, geležies ir maisto, gamybos kiekis.
4. Resurso balanso koeficientas, parodantis gamybos kiekių tolygumą.

Taip pat, kiekvieno eksperimento pabaigoje fiksuojami visų sprendimų priėmimo laikų vidurkis milisekundėmis.

Eksperimentų eiga:

1. Nustatomi pradinių resursų ir talpos kiekiai.
2. Paleidžiamas blackboard architektūra paremtas DI metodas.
3. Kas 30 sekundžių fiksuojami ir apskaičiuojami kiekybiniai rodikliai.
4. Praėjus 180 sekundžių, fiksuojami visų sprendimų priėmimo laikai.
5. Rezultatų duomenys pateikiami lentelėmis.

Atlikus eksperimentus, rezultatai bus lyginami tarpusavyje, kad patikrinti iškeltas hipotezes.

3.5.6. Tyrimo eiga

Tyrimo metu buvo atlikti trys eksperimentai trijuose skirtinguose ekonominiuose žaidimo scenarijuose – blogoje ekonominėje situacijoje, geroje ekonominėje situacijoje ir labai geroje ekonominėje situacijoje. Eksperimentų metu apskaičiuoti ir fiksuoti vertinimo kriterijų rezultatai.

Kriterijų reikšmių skaičiavimas

1. Pastatų skaičius, pastatytas per paskutines 30 sekundžių, skaičiuojamas vienetais.
2. Resursų talpos užpildymo procentas, skaičiuojamas pagal formulę, kur n – visų resurso tipų skaičius:

$$T = \frac{\sum_{i=1}^{n} Resurso\ kiekis}{\sum_{i=1}^{n} Resurso\ talpa} \times 100$$

3. Kiekvieno resurso gamybos kiekis laiko fiksavimo momentu, skaičiuojamas vienetais.
4. Resursų balanso koeficientas, parodantis kiek netolygiai paskirstyta gamyba. Mažesnė reikšmė rodo geresnį balansą tarp visų resurso tipų gamybos. Skaičiuojamas pagal formulę:

$$k(t) = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{P_i(t)}{\sum_{j=1}^{n} P_j(t)} - \frac{1}{n} \right|$$

5. Visų sprendimų priėmimo laikų vidurkis milisekundėmis.

Blogos ekonominės situacijos scenarijus

Parinkti pradiniai resursų kiekiai – 100 kiekvienam resurso tipui, 200 kiekvieno resurso talpai.

Geros ekonominės situacijos scenarijus

Parinkti pradiniai resursų kiekiai – 500 kiekvienam resurso tipui, 500 kiekvieno resurso talpai.

Labai geros ekonominės situacijos scenarijus

Parinkti pradiniai resursų kiekiai – 1000 kiekvienam resurso tipui, 2000 kiekvieno resurso talpai.

Duomenų rinkimas

Kiekvieno eksperimento rezultatai buvo matomi Unity žaidimų variklio konsolės lange ir vėliau surinkti ir atvaizduoti lentelėmis. Gauti rezultatai atvaizduoti rezultatų poskyryje.

Eksperimentų metu gauti duomenys buvo analizuojami, kad patikrinti tyrimo hipotezes ir įvertinti kiekybinį metodo efektyvumą.

3.5.7. Tyrimo metu naudota programinė įranga

Šio tyrimo eksperimentiniams scenarijams nebuvo sukurti atskiri testavimo metodai. Vietoj to, reikiamose blackboard ir ekspertinių sistemų vietose pridėti laikmačiai, rodiklių skaičiavimo metodai ir apskaičiuotų rezultatų rašymas į Unity žaidimų variklio langą.

3.5.8. Tyrimo rezultatai

Tyrimo rezultatai išskiriami į 2 dalis – ekonominių scenarijų testavimo ir sprendimų priėmimo našumo rezultatus. Bendra visų rezultatų lentelė, pateikta 4 priede, buvo išskirstyta į atskiras lenteles pagal tiriamus aspektus.

Ekonominių scenarijų testavimo rezultatai

Blackboard metodo rezultatai eksperimentinėse ekonominėse situacijose pateikiami 7-9 lentelėse.

7 lentelė. Blackboard metodo rezultatai blogos ekonominės situacijos scenarijuje.

Rezultatų fiksavimo laikas sekundėmis	Resurso tipas	Produkcijos pokytis intervalo metu	Pastatytų pastatų kiekis intervalo metu	Resursų talpos užpildymas procentais	Resursų balanso koeficientas
30	Gold	5	2	31,40%	0,07
	Wood	12			
	Stone	5			
	Iron	5			
	Food	5			
60	Gold	5	4	45,30%	0,0436
	Wood	14			
	Stone	12			
	Iron	12			
	Food	12			
90	Gold	5	4	41,40%	0,0845
	Wood	19			
	Stone	12			
	Iron	12			
	Food	5			
120	Gold	5	5	46,10%	0,0933
	Wood	26			
	Stone	12			
	Iron	12			
	Food	5			
150	Gold	10	4	54,00%	0,0395
	Wood	12			
	Stone	17			
	Iron	17			
	Food	17			
180	Gold	15	4	51,70%	0,0257
	Wood	17			
	Stone	15			
	Iron	15			
	Food	22			

Iš 7 lentelėje pateiktų eksperimento rezultatų matoma, kad esant blogai pradinei ekonominei situacijai, sukurtas blackboard metodas sugebėjo užtikrinti nuoseklų gamybos augimą. Didžiausias pokytis matomas medžio resurso augime, kadangi sukurtas metodas supranta, kad kadangi visi pastatai daugiausiai reikalauja medžio jų statymui, šį resursą reikia prioretizuoti labiausiai. Statomų pastatų kiekis per pirmąsias 30 sekundžių buvo tik 2, tačiau laikui bėgant stabilizavosi ties 4-5. Resursų talpos užpildymas nuosekliai augo viso eksperimento metu ir buvo ganėtinai didelis

paskutiniųjų rezultatų fiksavimo metu, o tai reiškia, kad metodas stengiasi kaupti resursus. Resursų balanso koeficientas svyravo viso eksperimento metu, kas parodo, kad metodas vis išleisdavo resursus, o vėliau stengėsi kompensuoti jų trūkumą pakeldamas gamybą.

8 lentelė. Blackboard metodo rezultatai geros ekonominės situacijos scenarijuje.

Rezultatų fiksavimo laikas sekundėmis	**Resurso tipas**	**Produkcijos pokytis intervalo metu**	**Pastatytų pastatų kiekis intervalo metu**	**Resursų talpos užpildymas procentais**	**Resursų balanso koeficientas**
30	Gold	20	7	33,90%	0,0185
	Wood	29			
	Stone	27			
	Iron	27			
	Food	27			
60	Gold	5	5	44,60%	0,0933
	Wood	26			
	Stone	12			
	Iron	12			
	Food	5			
90	Gold	15	4	52,50%	0,0257
	Wood	17			
	Stone	15			
	Iron	15			
	Food	22			
120	Gold	15	6	53,70%	0,0255
	Wood	29			
	Stone	22			
	Iron	22			
	Food	22			
150	Gold	25	8	47,10%	0,0155
	Wood	34			
	Stone	32			
	Iron	32			
	Food	32			
180	Gold	25	7	50,20%	0,0195
	Wood	27			
	Stone	32			
	Iron	32			
	Food	32			

8 lentelėje pateikti blackboard metodo rezultatai gero ekonominio scenarijaus metu. Pirmasis pastatytas pastatas – sandėlis resursų talpai praplėsti. Šio eksperimento metu pastebėta, kad metodas gebėjo nuosekliai didinti resursų gamybą ir efektyviai paskirstyti resursus gaminančių pastatų statybą. Resursų talpos užpildymas šio scenarijaus metu nuosekliai kilo, kas parodo kad resursai nėra švaistomi. Didžiausias resursų disbalansas pastebėtas 60 sekundžių intervale, kai koeficientas pasiekė 0.0933 reikšmę - metodas bandė užpildyti išleistų resursų trūkumą labiausiai padidindamas medžio resursuo gamybą, šitaip sumažindamas gamybos balansą. Vėlesniuose laiko intervaluose, disbalansas mažėjo – metodas sėkmingai palaipsniui lygino resursų gamybą.

9 lentelė. Blackboard metodo rezultatai labai geros ekonominės situacijos scenarijuje.

Rezultatų fiksavimo laikas sekundėmis	Resurso tipas	Produkcijos pokytis intervalo metu	Pastatytų pastatų kiekis intervalo metu	Resursų talpos užpildymas procentais	Resursų balanso koeficientas
30	Gold	25	10	32,10%	0,0305
	Wood	41			
	Stone	39			
	Iron	39			
	Food	32			
60	Gold	15	6	35,50%	0,0255
	Wood	29			
	Stone	22			
	Iron	22			
	Food	22			
90	Gold	20	9	28,00%	0,0294
	Wood	29			
	Stone	34			
	Iron	34			
	Food	27			
120	Gold	20	7	37,50%	0,0185
	Wood	29			
	Stone	27			
	Iron	27			
	Food	27			
150	Gold	25	6	41,90%	0,017
	Wood	32			
	Stone	25			
	Iron	25			
	Food	25			
180	Gold	35	10	42,40%	0,0171
	Wood	42			
	Stone	42			
	Iron	42			
	Food	35			

Iš 9 lentelės matoma, kad esant labai geroms ekonominėms sąlygoms, metodas iškart stambiu mastu pradėjo resursų gamybos plėtrą. Resursų talpos užpildymo procentas šio eksperimento metu po truputį kilo nuo 32.1% iki 42.4%, o tai rodo pakankamai gerą resursų paskirstymą gamybos plėtrai. Viso eksperimento metu resursų balanso koeficientas neviršijo 0.0305 ribos, o galiausiai nukrito iki 0.0171, kas reiškia kad resursų gamybos balansas buvo aukštas.

Sprendimų priėmimo našumo rezultatai skirtingų scenarijų metu

Sprendimų priėmimo trukmės rezultatai atvaizduoti 10 lentelėje.

10 lentelė. Blackboard metodo sprendimų priėmimo našumo rezultatai.

Scenarijus	**Vidutinė sprendimų priėmimo trukmė milisekundėmis**
Bloga ekonominė situacija	5.02
Gera ekonominė situacija	6.53
Labai gera ekonominė situacija	4.23

Labai geroje ekonominėje situacijoje vidutinė sprendimų priėmimo trukmė buvo mažiausia – 4.23 milisekundės, o tai rodo, kad blackboard ekonomikos valdymo metodas sprendimus priima greičiau, kai nėra resursų apribojimų. Blogos ekonominės situacijos metu vidutinė trukmė siekė 5.02 ms, o tai galima sieti su paprastesne, bet ribojančia pradine būsena, kur sprendimai dažniausiai susiję su resursų stygiaus sprendimu. Tuo tarpu geros ekonominės situacijos scenarijuje sprendimai vidutiniškai užtruko 6,53 ms – ilgiausiai iš visų trijų ekonominių scenarijų. Taip nutiko dėl to, kad šiame scenarijuje metodas turi daugiau pasirinkimo variantų nei blogoje situacijoje - susiduriama su talpos ribojimais, kurie verčia ją spręsti ne tik gamybos plėtimo bet ir talpos plėtimo problemas.

3.5.9. Tyrimo apribojimai

Atliktas išsamus blackboard architektūra grįsto DI metodo tyrimas, tačiau egzistuoja šie apribojimai:
1. Blackboard sistema nebuvo lyginama su kitomis, alternatyviomis ekonominių sprendimų sistemomis, dėl to tyrimo rezultatai neturi atsvaros taško.
2. Testavimas kontroliuojamuose scenarijuose pilnai neapibrėžia metodo veikimo galimybių realaus žaidimo scenarijaus metu – esant labiau chaotiškoms ir mažiau nuspėjamoms sąlygoms.

3.5.10. Pasiūlymai

Įgyvendintas blackboard architektūra grįstas DI metodas gali būti toliau tobulinamas įtraukiant papildomas ekspertines sistemas - gynybinių įtvirtinimų statymo, kareivių ir kitų padalinių paruošimo, ir karinių taktikų sistemas. Tokiu būdu blackboard sistema sprendimus priimtų atsižvelgdama į daugiau žaidimo aspektų ir turėtų daugiau automatizuoto funkcionalumo.

3.5.11. Tyrimo išvados

Atlikus 3 eksperimentus, kurių metu buvo naudojama blackboard architektūra grįstas DI metodas ekonomikos valdymui, empiriškai įvertintos hipotezės ir pateiktos šios išvados:

H1: Esant blogoms ekonominėms aplinkybėms - žemam turimų resursų ir saugojimo talpos kiekiui, metodas prioretizuos resursus generuojančių pastatų statymą.

Ši hipotezė pasitvirtino. Pirmojo scenarijaus metu buvo nuosekliai stebimas resursus generuojančių pastatų augimas, o resursų talpą praplečiančio pastato – sandėlio, statyba nebuvo inicijuota, kadangi nebuvo identifikuota talpos stoka.

H2: Esant geroms ekonominėms aplinkybėms – dideliam turimų resursų kiekiui, tačiau žemam saugojimo talpos kiekiui, metodas prioretizuos ir pradės statyti resursus talpinančius pastatus.

Ši hipotezė pasitvirtino. Antrajame scenarijuje, pirmasis pastatytas pastatas buvo sandėlis, o tai reiškia, kad metodas iškart sureagavo į resursų talpos trūkumą.

H3: Esant labai geroms ekonominėms aplinkybėms – dideliam turimų resursų ir saugojimo talpos kiekiui, metodas subalansuotai paskirstys pastatų statybas, išlaikydamas apytiksliai vienodą visų resursų tipų gamybą.

Ši hipotezė pasitvirtino. Trečiajame scenarijuje metodas demonstravo pakankamai tolygų visų resursų tipų gamybos augimą, o resursų balanso koeficientas išliko 0.0171-0.0305 ribose, taip dar kartą patvirtinant resursų gamybos augimo balansą.

Apibendrinant, atlikus eksperimentinius testavimus ir rezultatų vertinimą, galima teigti, kad blackboard architektūra grįstas DI metodas, skirtas ekonomikos valdymui geba veiksmingai reaguoti į įvairias ekonomines žaidybines situacijas. Metodas ne tik sėkmingai nustatė esamus prioritetus, bet ir pagal situaciją keitė savo veiksmų strategiją – mažėjant laisvos talpos kiekiui vietoj tolimesnės resursų gamybos plėtros, statė sandėlius.

4. Apibendrinimai iš išvados

1. Atlikus literatūros analizę, projektavimą ir empirinius tyrimus, įvertintos DI metodų taikymo galimybės strateginių kompiuterinių žaidimų kontekste.
2. Išanalizavus žaidimų DI metodų istorinę raidą, tipus ir taikymą egzistuojančiuose strateginiuose žaidimuose pastebėta, kad dažniausiai naudojami metodai yra deterministiniai, dėl jų nuspėjamumo, testuojamumo ir patikimumo. Išskirtos pagrindinės problemos, su kuriomis susiduria nepriklausomi žaidimų kūrėjai – labai ribotas priėjimas prie žaidimų kūrimo korporacijų naudojamų pažangių DI metodų ir atvirų pavyzdžių trūkumas. Šių problemų sprendimas tapo pagrindiniu darbo tikslu – pateikti aktualius, aiškiai įgyvendintus, pritaikomus DI metodus ir jų tyrimo rezultatus, kuriais galėtų naudotis nepriklausomi žaidimų kūrėjai.
3. Projektinėje dalyje naudojant Unity žaidimų variklį ir C# programavimo kalbą sukurtas strateginio žaidimo prototipas, įgyvendinantis analizės metu pasirinktus DI metodus – procedūrinį žemėlapių generavimą, kelio radimo metodus ir blackboard architektūra grįstą ekonomikos valdymo sistemą. Prototipas sukurtas naudojantis krioklio kūrimo modeliu, kuris leido nuosekliai pereiti per visus etapus – nuo analizės iki testavimo. Strateginio žaidimo struktūra buvo atvaizduota naudojantis panaudojimo atvejų, paketų, klasių, veiklos, būsenų ir sekų diagramomis.
4. Tiriamojoje dalyje atlikus tris empirinius tyrimus, eksperimentiškai ištestuoti ir įvertinti DI metodai – procedūrinis žemėlapių generavimas, kelio radimo metodai ir blackboard architektūra grįstas ekonomikos valdymo metodas.
5. Procedūrinio žemėlapių generavimo tyrimo metu atlikus 18 eksperimentų su Perlin ir Simplex triukšmo funkcijomis, ir trimis skirtingomis resursų išdėstymo strategijomis, buvo įvertintos šių funkcijų ir strategijų kombinacijų galimybės. Efektyviausios kombinacijos buvo pateikiamos kaip rekomendacijos. Empiriškai patvirtinta 1 iš 3 iškeltų hipotezių.
6. Įvertinus procedūrinio žemėlapių generavimo tyrimo rezultatus paaiškėjo, kad Perlin triukšmas generuoja natūralesnius ir strateginiams žaidimams tinkamesnius reljefus, užtikrindamas ~97% panaudojamo paviršiaus procentą lyginant su Simplex ~74.5%, esant testavimo metu naudotoms sąlygoms.
7. Procedūrinio žemėlapių generavimo tyrimo metu nustatyta, kad tinklelinė resursų išdėstymo strategija turi mažiausią standartinį resursų nuokrypį – 0.26-0.29 ir aukščiausią resursų klasteriškumo koeficientą – 15.8-17.8, o tai reiškia kad ji užtikrina didžiausią resursų balanso lygį.
8. Kelio radimo metodų tyrimo metu atlikus 30 eksperimentų, lyginančių A* algoritmu pagrįstą metodą su Unity NavMesh sistema trijose skirtingose aplinkose, nustatyta, kad Unity NavMesh rado trumpesnius kelius aplinkose su bet kokio tipo kliūtimis ir veikė greičiau. Teikiamas pasiūlymas naudoti Unity NavMesh paprastesnėse, labiau statinėse aplinkose, dėl jo veikimo greičio ir trumpesnių kelių radimo.
9. Atlikus 10 papildomų eksperimentų, skirtų tinklelių atsinaujinimo našumui nustatyti, pastebėta, kad įgyvendinto A* algoritmu pagrįsto metodo tinklelis atsinaujino greičiau – vidutiniškai per 13 milisekundžių, lyginant su Unity NavMesh 18-39 milisekundžių, todėl pasižymėjo didesniu prisitaikymu prie pokyčių.
10. Atlikus eksperimentinius kelio radimo metodų testavimus ir įvertinus jų rezultatus, empiriškai patvirtinta 2 iš 3 iškeltų hipotezių.
11. Atlikus blackboard architektūra grįsto ekonomikos valdymo metodo tyrimą trijuose skirtinguose ekonominiuose scenarijuose pastebėta, kad šis metodas reaguoja į pokyčius – kai trūksta resursų talpos – stato sandėlius, ir priima sprendimus vidutiniškai per 4-6 milisekundes, todėl geba

efektyviai ir tinkamai nustatyti prioritetus pagal ekonominę situaciją,. Empiriškai patvirtinta 3 iš 3 hipotezių.

Dirbtinio intelekto metodų taikymas strateginiuose žaidimuose išlieka aktualia sritimi, o vienas didžiausių iššūkių – metodų prieinamumas nepriklausomiems žaidimų kūrėjams. Šis darbas pateikia veikiančius ir empiriškai įvertintus DI metodų pavyzdžius, kuriais galima remtis kuriant žaidimą neturint didelio biudžeto.

Empiriniai duomenys patvirtino 6 iš 9 iškeltų hipotezių, o nepatvirtintos hipotezės suteikė svarbių įžvalgų apie įgyvendintų metodų ribas ir sritis, kuriose galimos tolimesnės optimizacijos ir plėtra.

Literatūros sąrašas

[1] NASA, „Defining Artificial Intelligence“ (2025) [Tinkle]. Nuoroda: https://www.nasa.gov/what-is-artificial-intelligence/ [Kreiptasi 2025-03-01]

[2] J. Dsouza, „AI in Gaming“ (2025) [Tinkle]. Nuoroda: https://www.engati.com/blog/ai-in-gaming [Kreiptasi 2025-03-01]

[3] Columbia Engineering, „Explore how AI is transforming the gaming industry“ (2025) [Tinkle]. Nuoroda: https://ai.engineering.columbia.edu/ai-applications/ai-video-games/ [Kreiptasi 2025-03-01]

[4] Qubit Labs Team, „How Many Game Developers Are There In the World? Surprising Statistics“ [Tinkle] https://qubit-labs.com/how-many-game-developers-are-there-in-the-world-surprising-statistics/ [Kreiptasi 2025-03-01]

[5] Ian Millington, „Artificial Intelligence for Games“ (2006) [Tinkle]. Nuoroda: https://theswissbay.ch/pdf/Gentoomen%20Library/Game%20Development/Programming/Artificial%20Intelligence%20for%20Games.pdf [43, 242 psl.][Kreiptasi 2025-03-01]

[6] B. Di Stefano, „Artificial Intelligence vs Computational Intelligence“ (2021) [Tinkle]. Nuoroda: https://www.ieeetoronto.ca/june-2021/artificial-intelligence-vs-computational-intelligence/ [Kreiptasi 2025-03-05]

[7] Georgios N. Yannakakis and Julian Togelius, „Artificial Intelligence and Games“, 2018-01-26 [Tinkle]. Nuoroda: https://gameaibook.org/book.pdf [Kreiptasi 2025-03-05]

[8] „History of AI Use in Video Game Design“ (2021) [Tinkle]. Nuoroda: https://bigdataanalyticsnews.com/history-of-artificial-intelligence-in-video-games/ [Kreiptasi 2025-03-05]

[9] A. Schulz, „25 years ago: Deep Blue beats Kasparov“ [Tinkle]. Nuoroda: https://en.chessbase.com/post/25-years-ago-deep-blue-beats-kasparov [Kreiptasi 2025-03-05]

[10] S. Rizzo, „Imagined Thinking Machines: Artificial Intelligence in Video Games“, Full Sail University, bal. 2, 2015 [Tinkle]. Nuoroda: https://hub.fullsail.edu/articles/imagined-thinking machines-artificial-intelligence-in-video-games [Kreiptasi 2025-03-05]

[11] Market Trends, „The Evolution of Artificial Intelligence in Video Games“, Analytics Insight, 12 29 2020 [Tinkle]. Nuoroda: https://www.analyticsinsight.net/the-evolution-of-artificial-intelligence-in video-games/ [Kreiptasi 2025-03-05]

[12] Giuseppe De Giacomo , Bastien Maubert, Aniello Murano, „Nondeterministic Strategies and their Refinement in Strategy Logic“, 2020 [Tinkle]. Nuoroda: https://proceedings.kr.org/2020/30/kr2020-0030-de-giacomo-et-al.pdf [Kreiptasi 2025-03-05]

[13] Lori Dajose, „What Neural Networks Playing Video Games Teach Us About Our Own Brains“, 2021-01-07 [Tinkle]. Nuoroda: https://www.caltech.edu/about/news/neural-networks-playing-video-games-teach-us-about-our-own-brains [Kreiptasi 2025-03-05]

[14] Patrick Kelly, „Genetic Algorithms in Games (Part 1)“, 2018-09-04 [Tinkle]. Nuoroda: https://www.gamedeveloper.com/design/genetic-algorithms-in-games-part-1- [Kreiptasi 2025-03-05]

[15] Michele Pirovano, „The use of Fuzzy Logic for Artificial Intelligence in Games“, 2012-12-07 [Tinkle]. Nuoroda: https://www.michelepirovano.com/pdf/fuzzy_ai_in_games.pdf [2 psl.][Kreiptasi 2025-03-05]

[16] Amit Yadav, „Reinforcement Learning in Games: A Complete Guide“, 2024-06-26 [Tinkle]. Nuoroda: https://medium.com/%40amit25173/reinforcement-learning-in-games-a-complete-guide-24d1cab79317 [Kreiptasi 2025-03-05]

[17] GeeksForGeeks, „Difference between Deterministic and Non-deterministic Algorithms“ (2025) [Tinkle]. Nuoroda: https://www.geeksforgeeks.org/difference-between-deterministic-and-non-deterministic-algorithms/ [Kreiptasi 2025-03-05]

[18] John Laird, Sugih Jamin, „Artificial Intelligence and Computer Games“ [Tinkle]. Nuoroda: https://web.eecs.umich.edu/~sugih/courses/eecs494/fall06/lectures/lecture13-gameai.pdf [Kreiptasi 2025-03-05]

[19] Göksu Deniz, „What is the Blackboard Design Pattern?“, 2022-12-16 [Tinkle]. Nuoroda: https://justgokus.medium.com/what-is-the-blackboard-design-pattern-c834227dc617 [Kreiptasi 2025-03-05]

[20] Aleena Sebastian, „Game Strategy Optimization Using Decision Trees“, 2024-05-25 [Tinkle]. Nuoroda: https://medium.com/@aleena.sebastian/game-strategy-optimization-using-decision-trees-d4067008eed1 [Kreiptasi 2025-03-05]

[21] Jesse Glover, „Building a Rule-Based System in C#“, 2024-04-30 [Tinkle]. Nuoroda: https://medium.com/@kohouri/learning-from-saga-frontier-building-a-rule-based-game-system-in-c-eb338bf6eb9b [Kreiptasi 2025-03-05]

[22] N. Ahamed, „AI Cheating in Games“, Medium, 2022-09-30 [Tinkle]. Nuoroda: https://n-ahamed36.medium.com/ai-cheating-in-games-583e333677ce [Kreiptasi 2025-03-05]

[23] Wikipedia, „Civilization V“ [Tinkle]. Nuoroda: https://en.wikipedia.org/wiki/Civilization_V [Kreiptasi 2025-03-05]

[24] Steam, „Civilization V“ [Tinkle]. Nuoroda: https://store.steampowered.com/app/8930/Sid_Meiers_Civilization_V/ [Kreiptasi 2025-03-05]

[25] Patrick Klepek, „To Modders Who Decided to Overhaul the AI in Civilization V“, 2017-04-13 [Tinkle]. Nuoroda: https://www.vice.com/en/article/the-modders-who-decided-to-overhaul-the-ai-in-civilization-v/ [Kreiptasi 2025-03-05]

[26] Anthony D. Del Monaco, Forrest A. Jones, Seth W. Bruneel, „Trade Secrets: What Game Developers Should Know“, 2021-08-24 [Tinkle]. Nuoroda: https://igda-website.s3.us-east-2.amazonaws.com/wp-content/uploads/2021/08/31095742/Article_Trade-Secrets-What-Game-Developers-Should-Know_8.24.21.pdf [Kreiptasi 2025-03-05]

[27] Wikipedia, „Age of Empires III“ [Tinkle]. Nuoroda: https://en.wikipedia.org/wiki/Age_of_Empires_III [Kreiptasi 2025-03-05]

[28] Age of Empires Fandom Wiki, „Artificial Intelligence“ [Tinkle]. Nuroda: https://ageofempires.fandom.com/wiki/Artificial_intelligence [Kreiptasi 2025-03-05]

[29] ESO Community, „Writing an AI script for Age of Empires III (Part III)“, 2020-03-24 [Tinkle]. Nuoroda: https://eso-community.net/viewtopic.php?t=19970 [Kreiptasi 2025-03-05]

[30] Christian Mills, „Procedural Map Generation Techniques“, 2021-12-09 [Tinkle]. Nuoroda: https://christianjmills.com/posts/procedural-map-generation-techniques-notes/ [Kreiptasi 2025-03-05]

[31] Steve Jones, „Generative AI and Open World Gaming“, 2023-01-10 [Tinkle]. Nuoroda: https://blog.metamirror.io/generative-ai-and-open-world-gaming-4f00e153ea12 [Kreiptasi 2025-03-05]

[32] BIT-101, „Perlin vs. Simplex“, 2021-07-17 [Tinkle]. Nuoroda: https://www.bit-101.com/2017/2021/07/perlin-vs-simplex/ [Kreiptasi 2025-03-05]

[33] Polydin, „Exploring Procedural Generation in Games“, 2024-06-29 [Tinkle]. Nuoroda: https://polydin.com/procedural-generation-in-games/ [Kreiptasi 2025-03-05]

[34] Necmettin Sancak, „GENERATING AND BREAKING THE RADIAL GRID FOR PROCEDURAL CITY LAYOUTS“, (2023) [Tinkle]. Nuoroda: https://www.academia.edu/122687703/GENERATING_AND_BREAKING_THE_RADIAL_GRID_FOR_PROCEDURAL_CITY_LAYOUTS [Kreiptasi 2025-03-05]

[35] Rafi Akhtar, „Search Algorithms in AI“, 2023-03-22 [Tinkle]. Nuoroda: https://www.geeksforgeeks.org/search-algorithms-in-ai/ [Kreiptasi 2025-03-05]

[36] Amit P., „Search Algorithms in AI“, 1997 (atnaujinta 2025) [Tinkle]. Nuoroda: https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html [Kreiptasi 2025-03-05]

[37] Cory Desmond, „The Island – Genre, implementation of AI and decision trees“, 2014-10-01 (atnaujinta 2025) [Tinkle]. Nuoroda: https://corydesmondai.wordpress.com/2014/10/01/the-island-genre-and-implementation-of-ai/ [Kreiptasi 2025-03-05]

[38] Brandon Antonette, „Decision Trees in Video Games“, 2019-12-11 [Tinkle]. Nuoroda: Decision Trees in Video Games. What’s a Decision Tree? | by Brandon Antonette | Medium [Kreiptasi 2025-03-05]

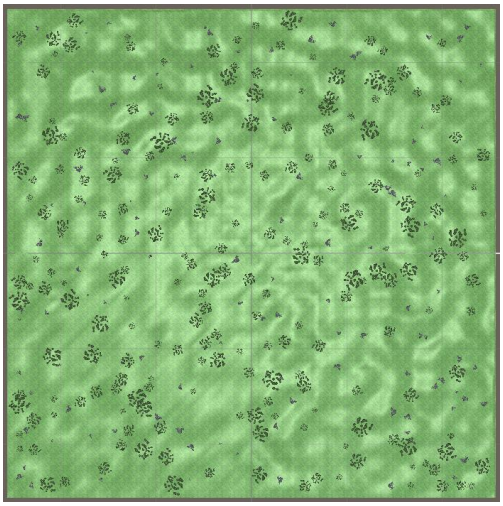
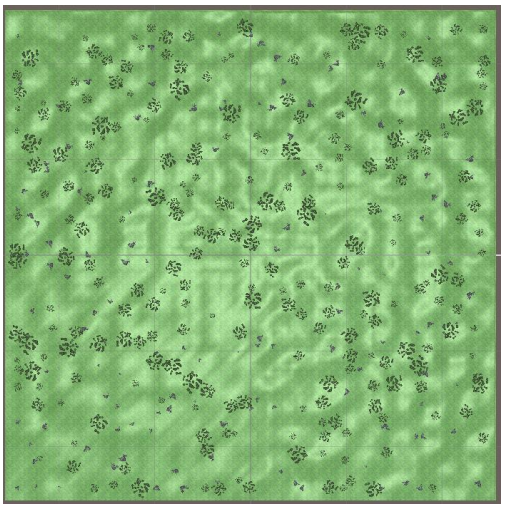
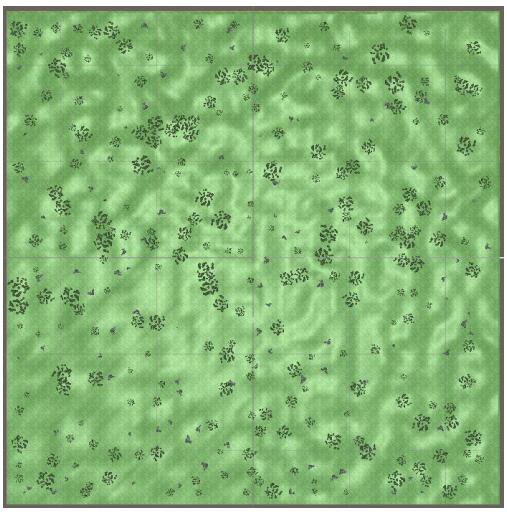
[39] Wikipedia, „Blackboard system“ [Tinkle]. Nuoroda: https://en.wikipedia.org/wiki/Blackboard_system [Kreiptasi 2025-03-05]

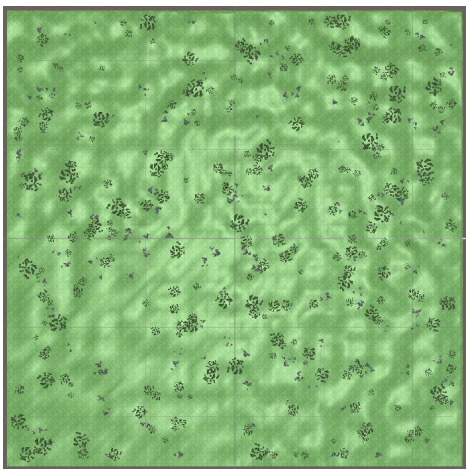
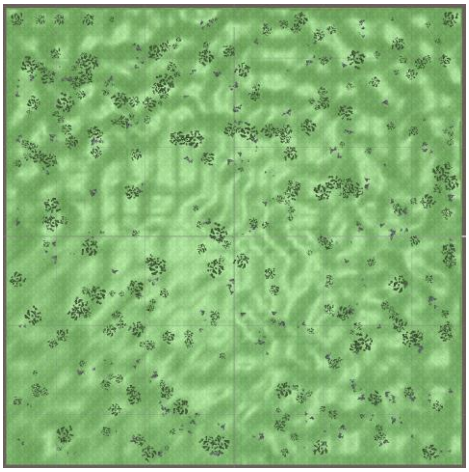
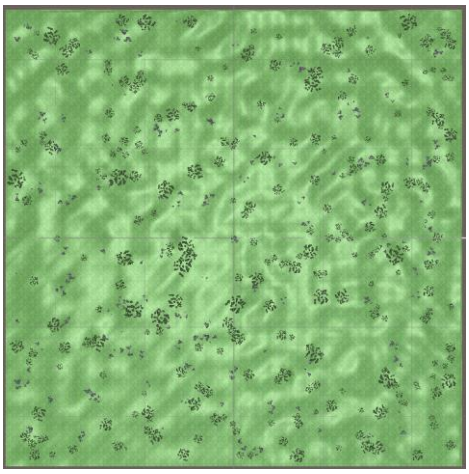
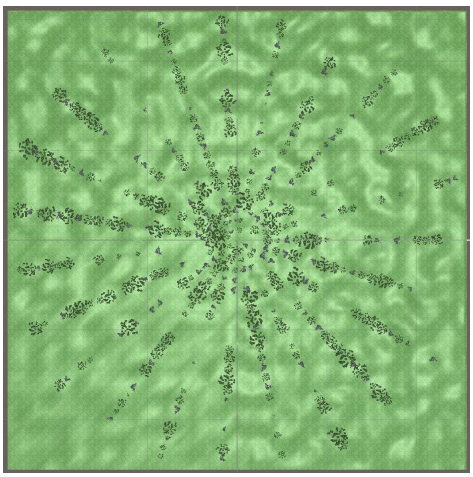
[40] Bruno Poli, „Creating Decoupled Features: The Blackboard System“, 2018-06-19 [Tinkle]. Nuoroda: https://www.gamedeveloper.com/design/creating-decoupled-features-the-blackboard-system [Kreiptasi 2025-03-05]

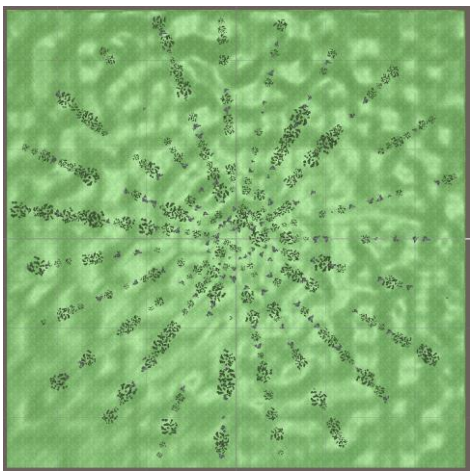
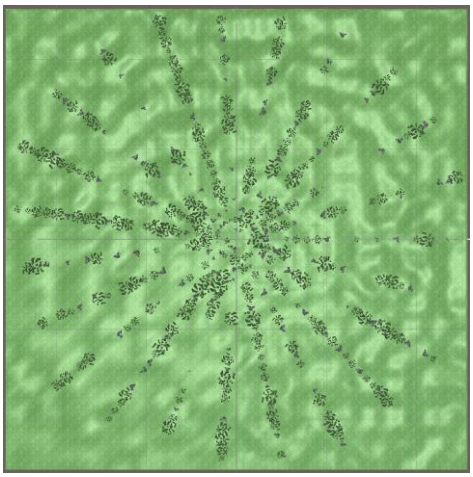
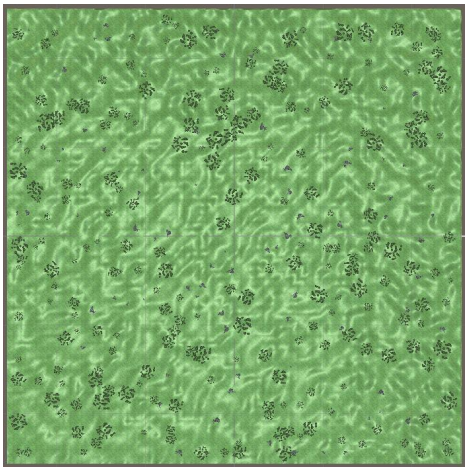
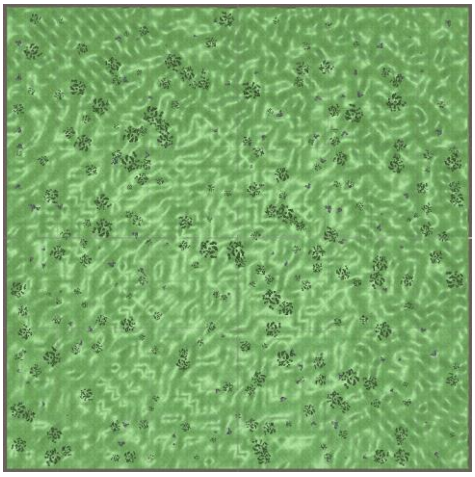
[41] Unity3D docs, „Navigation areas and costs“ (2025) [Tinkle] Nuoroda: https://docs.unity3d.com/Packages/com.unity.ai.navigation@1.1/manual/AreasAndCosts.html#:~:text=Unity%20uses%20A*%20to%20calculate,until%20the%20destination%20is%20reached. [Kreiptasi 2025-05-21]

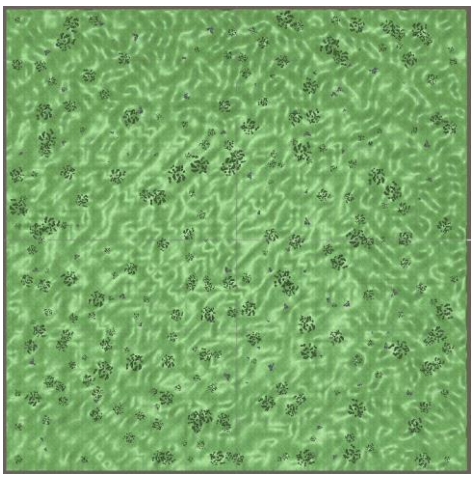
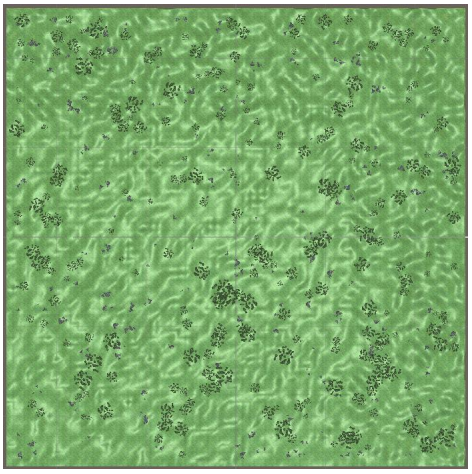
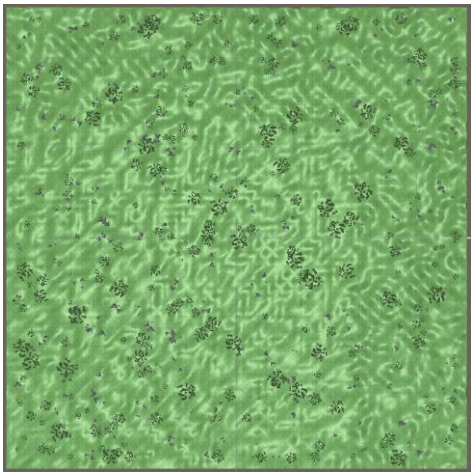
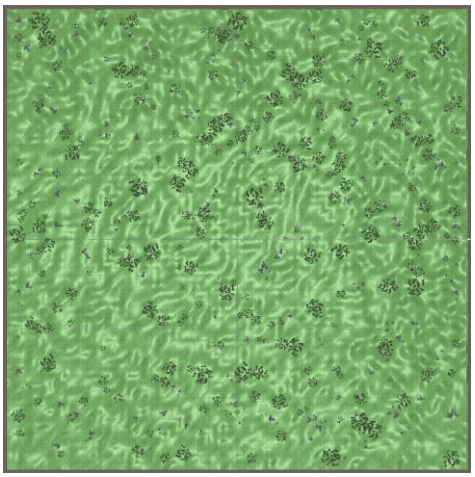
Priedai

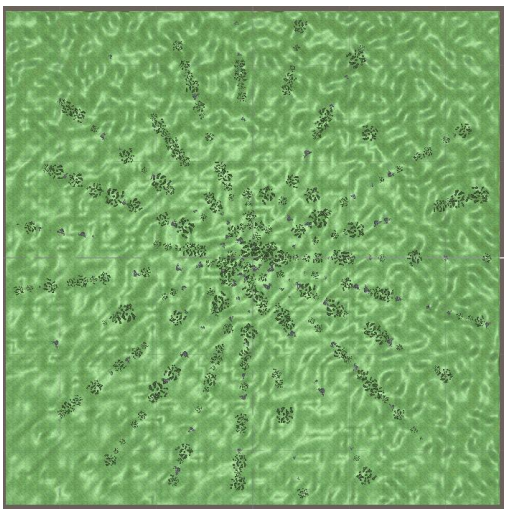
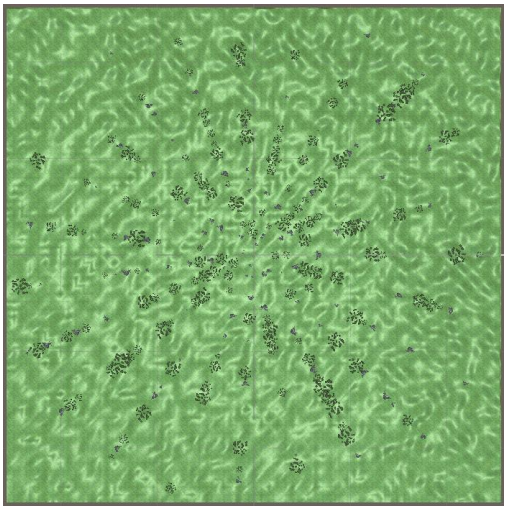
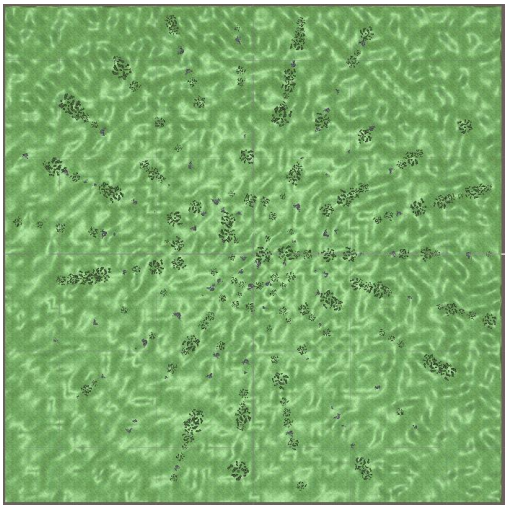
1 priedas. Žemėlapių generavimo eksperimento rezultatai

Nr.	Triukšmo funkcija	Resursų išdėstymo strategija	Sugeneruoto žemėlapio iškarpa	Apskaičiuoti rodikliai neapdorota forma (Unity Debug.Log)
1	Perlin	Tinklelinė		[Perlin][Grid] Map generation time: 26ms; Resource generation time: 11ms; Usable Terrain Percentage: 96,97266, Generated resource count: 339, Standard Resource Deviation: 0,2764094, Resource Clustering Coefficient: 16,36584.
2				[Perlin][Grid] Map generation time: 26ms; Resource generation time: 13ms; Usable Terrain Percentage: 96,94824, Generated resource count: 375, Standard Resource Deviation: 0,2909221, Resource Clustering Coefficient: 15,80621.
3				[Perlin][Grid] Map generation time: 26ms; Resource generation time: 12ms; Usable Terrain Percentage: 96,92383, Generated resource count: 343, Standard Resource Deviation: 0,2778778, Resource Clustering Coefficient: 16,49822.

4	Perlin	Klasterinė		[Perlin][Clusters] Map generation time: 26ms; Resource generation time: 21ms; Usable Terrain Percentage: 97,07031, Generated resource count: 512, Standard Resource Deviation: 0,3657719, Resource Clustering Coefficient: 9,3957.
5				[Perlin][Clusters] Map generation time: 26ms; Resource generation time: 21ms; Usable Terrain Percentage: 96,97266, Generated resource count: 512, Standard Resource Deviation: 0,355619, Resource Clustering Coefficient: 9,77585.
6				[Perlin][Clusters] Map generation time: 27ms; Resource generation time: 25ms; Usable Terrain Percentage: 96,94824, Generated resource count: 512, Standard Resource Deviation: 0,3542432, Resource Clustering Coefficient: 9,766783.
7	Perlin	Spindulinė		[Perlin][Radial] Map generation time: 26ms; Resource generation time: 12ms; Usable Terrain Percentage: 96,94824, Generated resource count: 354, Standard Resource Deviation: 0,2818593 , Resource Clustering Coefficient: 11,55995.

8				[Perlin][Radial] Map generation time: 26ms; Resource generation time: 12ms; Usable Terrain Percentage: 96,99707, Generated resource count: 353, Standard Resource Deviation: 0,2823667 , Resource Clustering Coefficient: 11,49004.
9				[Perlin][Radial] Map generation time: 26ms; Resource generation time: 15ms; Usable Terrain Percentage: 96,99707, Generated resource count: 350, Standard Resource Deviation: 0,2830208 , Resource Clustering Coefficient: 11,60795.
10	Simplex	Tinklelinė		[Simplex][Grid] Map generation time: 82ms; Resource generation time: 10ms; Usable Terrain Percentage: 74,31641, Generated resource count: 303, Standard Resource Deviation: 0,2617296 , Resource Clustering Coefficient: 17,65723.
11				[Simplex][Grid] Map generation time: 82ms; Resource generation time: 10ms; Usable Terrain Percentage: 73,55957, Generated resource count: 304, Standard Resource Deviation: 0,2630563 , Resource Clustering Coefficient: 17,34804.

12				[Simplex][Grid] Map generation time: 82ms; Resource generation time: 13ms; Usable Terrain Percentage: 75,1709, Generated resource count: 301, Standard Resource Deviation: 0,2609331 , Resource Clustering Coefficient: 17,85492.
13	Simplex	Klasterinė		[Simplex][Clusters] Map generation time: 83ms; Resource generation time: 24ms; Usable Terrain Percentage: 75,04883, Generated resource count: 512, Standard Resource Deviation: 0,3528622, Resource Clustering Coefficient: 10,14212.
14				[Simplex][Clusters] Map generation time: 82ms; Resource generation time: 21ms; Usable Terrain Percentage: 73,63281, Generated resource count: 512, Standard Resource Deviation: 0,3528622, Resource Clustering Coefficient: 10,26391.
15				[Simplex][Clusters] Map generation time: 83ms; Resource generation time: 20ms; Usable Terrain Percentage: 74,04785, Generated resource count: 512, Standard Resource Deviation: 0,3576726, Resource Clustering Coefficient: 10,2578.

16	Simplex	Spindulinė		[Simplex][Radial] Map generation time: 83ms; Resource generation time: 10ms; Usable Terrain Percentage: 75,04883, Generated resource count: 299, Standard Resource Deviation: 0,2629338 , Resource Clustering Coefficient: 12,5044.
17				[Simplex][Radial] Map generation time: 83ms; Resource generation time: 8ms; Usable Terrain Percentage: 74,4873, Generated resource count: 238, Standard Resource Deviation: 0,2370529 , Resource Clustering Coefficient: 14,56123.
18				[Simplex][Radial] Map generation time: 82ms; Resource generation time: 8ms; Usable Terrain Percentage: 74,92676, Generated resource count: 242, Standard Resource Deviation: 0,2368114 , Resource Clustering Coefficient: 13,80543.

2 Priedas. Pilni kelio radimo tyrimo rezultatai

Kelio radimo metodas	**Žemėlapio scenarijus**	**Apskaičiuoti rezultatai neapdorota forma (Unity Debug.Log)**
A* algoritmu grįstas metodas	Tuščias žemėlapis	[A*][EmptyMap] From (-11,9, -2,5) to (13,0, 16,1) | Straight: 31,05m | Path: 35,10m | Time: 17,31ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (-11,9, -2,5) to (13,0, 16,1) | Straight: 31,05m | Path: 38,57m | Time: 9,26ms | Success: True

A* algoritmu grįstas metodas		[A*][EmptyMap] From (-10,0, -19,2) to (15,4, -16,4) | Straight: 25,58m | Path: 28,79m | Time: 17,13ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (-10,0, -19,2) to (15,4, -16,4) | Straight: 25,58m | Path: 30,53m | Time: 6,63ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (15,7, -2,2) to (-12,8, 1,1) | Straight: 28,69m | Path: 32,19m | Time: 16,94ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (15,7, -2,2) to (-12,8, 1,1) | Straight: 28,69m | Path: 35,16m | Time: 9,75ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (6,3, -14,6) to (14,0, -8,6) | Straight: 9,74m | Path: 10,88m | Time: 8,84ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (6,3, -14,6) to (14,0, -8,6) | Straight: 9,74m | Path: 11,70m | Time: 8,30ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (3,2, 2,2) to (-16,9, -18,8) | Straight: 29,04m | Path: 35,33m | Time: 15,75ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (3,2, 2,2) to (-16,9, -18,8) | Straight: 29,04m | Path: 35,68m | Time: 6,00ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (8,0, 16,0) to (-16,5, 15,7) | Straight: 24,48m | Path: 29,09m | Time: 12,80ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (8,0, 16,0) to (-16,5, 15,7) | Straight: 24,48m | Path: 28,48m | Time: 9,39ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (13,4, -11,4) to (16,3, 16,5) | Straight: 28,05m | Path: 32,32m | Time: 14,39ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (13,4, -11,4) to (16,3, 16,5) | Straight: 28,05m | Path: 31,68m | Time: 12,19ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (-5,1, -0,9) to (-8,1, -8,6) | Straight: 8,24m | Path: 9,77m | Time: 8,12ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (-5,1, -0,9) to (-8,1, -8,6) | Straight: 8,24m | Path: 10,05m | Time: 10,76ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (14,4, 16,5) to (-13,9, -2,4) | Straight: 33,96m | Path: 41,78m | Time: 14,44ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (14,4, 16,5) to (-13,9, -2,4) | Straight: 33,96m | Path: 37,67m | Time: 11,19ms | Success: True
A* algoritmu grįstas metodas		[A*][EmptyMap] From (-18,5, 8,8) to (4,4, -13,9) | Straight: 32,27m | Path: 39,13m | Time: 10,03ms | Success: True
Unity NavMesh		[NavMesh][EmptyMap] From (-18,5, 8,8) to (4,4, -13,9) | Straight: 32,27m | Path: 38,53m | Time: 10,80ms | Success: True
A* algoritmu grįstas metodas	Žemėlapis su statine kliūtimi	[A*][ObstacleMap] From (-0,7, 11,1) to (-19,9, -5,5) | Straight: 25,33m | Path: 34,19m | Time: 19,70ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (-0,7, 11,1) to (-19,9, -5,5) | Straight: 25,33m | Path: 33,54m | Time: 11,09ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (5,4, 2,8) to (-11,5, -7,0) | Straight: 19,50m | Path: 24,61m | Time: 17,83ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (5,4, 2,8) to (-11,5, -7,0) | Straight: 19,50m | Path: 25,39m | Time: 12,34ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (17,9, -8,4) to (14,0, 19,3) | Straight: 28,03m | Path: 39,20m | Time: 16,63ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (17,9, -8,4) to (14,0, 19,3) | Straight: 28,03m | Path: 35,71m | Time: 11,06ms | Success: True

A* algoritmu grįstas metodas		[A*][ObstacleMap] From (2,4, -6,7) to (-5,8, -8,0) | Straight: 8,31m | Path: 10,93m | Time: 19,50ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (2,4, -6,7) to (-5,8, -8,0) | Straight: 8,31m | Path: 10,78m | Time: 15,98ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (9,0, -2,5) to (7,9, -11,7) | Straight: 9,28m | Path: 11,92m | Time: 16,18ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (9,0, -2,5) to (7,9, -11,7) | Straight: 9,28m | Path: 12,17m | Time: 12,50ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (-12,5, -3,2) to (-8,4, -16,7) | Straight: 14,09m | Path: 19,68m | Time: 14,90ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (-12,5, -3,2) to (-8,4, -16,7) | Straight: 14,09m | Path: 19,12m | Time: 14,95ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (-9,1, 17,3) to (4,9, -10,1) | Straight: 30,77m | Path: 38,97m | Time: 20,34ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (-9,1, 17,3) to (4,9, -10,1) | Straight: 30,77m | Path: 42,85m | Time: 12,08ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (5,2, -14,4) to (2,8, -13,6) | Straight: 2,55m | Path: 3,57m | Time: 17,46ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (5,2, -14,4) to (2,8, -13,6) | Straight: 2,55m | Path: 3,39m | Time: 12,24ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (13,8, -15,7) to (-3,7, 19,2) | Straight: 38,99m | Path: 54,44m | Time: 15,91ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (13,8, -15,7) to (-3,7, 19,2) | Straight: 38,99m | Path: 53,81m | Time: 15,61ms | Success: True
A* algoritmu grįstas metodas		[A*][ObstacleMap] From (-1,8, 18,1) to (-5,5, 16,6) | Straight: 3,99m | Path: 5,41m | Time: 19,08ms | Success: True
Unity NavMesh		[NavMesh][ObstacleMap] From (-1,8, 18,1) to (-5,5, 16,6) | Straight: 3,99m | Path: 5,39m | Time: 13,90ms | Success: True
A* algoritmu grįstas metodas	Žemėlapis su dinamiškai atsirandančia kliūtimi	[A*][DynamicObstacle] From (14,8, -11,8) to (7,7, 19,0) | Straight: 31,61m | Path: 43,26m | Time: 25,39ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (14,8, -11,8) to (7,7, 19,0) | Straight: 31,61m | Path: 39,48m | Time: 15,29ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (7,3, -14,6) to (16,2, 10,7) | Straight: 26,85m | Path: 33,55m | Time: 23,49ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (7,3, -14,6) to (16,2, 10,7) | Straight: 26,85m | Path: 31,36m | Time: 15,90ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (13,2, -7,4) to (-3,0, -8,1) | Straight: 16,23m | Path: 21,73m | Time: 19,91ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (13,2, -7,4) to (-3,0, -8,1) | Straight: 16,23m | Path: 19,28m | Time: 13,82ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (5,4, 14,9) to (-11,3, -1,2) | Straight: 23,21m | Path: 31,72m | Time: 18,23ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (5,4, 14,9) to (-11,3, -1,2) | Straight: 23,21m | Path: 27,05m | Time: 14,92ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (18,4, -5,1) to (-2,7, 13,7) | Straight: 28,28m | Path: 36,82m | Time: 26,00ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (18,4, -5,1) to (-2,7, 13,7) | Straight: 28,28m | Path: 34,18m | Time: 16,32ms | Success: True

A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (10,5, -14,3) to (11,6, -16,4) | Straight: 2,44m | Path: 3,29m | Time: 19,26ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (10,5, -14,3) to (11,6, -16,4) | Straight: 2,44m | Path: 2,97m | Time: 14,32ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (16,0, -14,9) to (-14,8, 13,7) | Straight: 41,95m | Path: 56,79m | Time: 22,20ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (16,0, -14,9) to (-14,8, 13,7) | Straight: 41,95m | Path: 51,13m | Time: 14,01ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (7,4, 10,0) to (-4,4, -5,1) | Straight: 19,13m | Path: 23,15m | Time: 17,61ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (7,4, 10,0) to (-4,4, -5,1) | Straight: 19,13m | Path: 21,80m | Time: 13,14ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (12,5, 12,4) to (-19,4, -2,8) | Straight: 35,27m | Path: 44,56m | Time: 21,58ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (12,5, 12,4) to (-19,4, -2,8) | Straight: 35,27m | Path: 43,97m | Time: 11,53ms | Success: True
A* algoritmu grįstas metodas		[A*][DynamicObstacle] From (9,6, -5,1) to (-1,2, -7,5) | Straight: 11,03m | Path: 14,70m | Time: 18,50ms | Success: True
Unity NavMesh		[NavMesh][DynamicObstacle] From (9,6, -5,1) to (-1,2, -7,5) | Straight: 11,03m | Path: 13,24m | Time: 11,44ms | Success: True

3 Priedas. Pilni kelio radimo metodų tinklelių eksperimentiniai rezultatai.

Žemėlapio scenarijus	**Kelio radimo algoritmas**	**Eksperimento numeris**	**Pradinio taško koordinatės**	**Galutinio taško koordinatės**	**Tiesus atstumas**	**Apskaičiuoto kelio ilgis**	**Kelio radimo laikas**
Tuščias žemėlapis	A*	1	(-11,9, -2,5)	(13,0,16,1)	31,05m	35,10m	17,31ms
	NavMesh					38,57m	9,26ms
	A*	2	(-10,0, -19,2)	(15,4,-16,4)	25,58m	28,79m	17,13ms
	NavMesh					30,53m	6,63ms
	A*	3	(15,7, -2,2)	(-12,8,1,1)	28,69m	32,19m	16,94ms
	NavMesh					35,16m	9,75ms
	A*	4	(6,3, -14,6)	(14,0,-8,6)	9,74m	10,88m	8,84ms
	NavMesh					11,70m	8,30ms
	A*	5	(3,2, 2,2)	(-16,9,-18,8)	29,04m	35,33m	15,75ms
	NavMesh					35,68m	6,00ms
	A*	6	(8,0, 16,0)	(-16,5,15,7)	24,48m	29,09m	12,80ms
	NavMesh					28,48m	9,39ms
	A*	7	(13,4, -11,4)	(16,3,16,5)	28,05m	32,32m	14,39ms
	NavMesh					31,68m	12,19ms
	A*	8	(-5,1, -0,9)	(-8,1,-8,6)	8,24m	9,77m	8,12ms
	NavMesh					10,05m	10,76ms

	A*	9	(14,4, 16,5)	(-13,9,-2,4)	33,96m	41,78m	14,44ms
	NavMesh					37,67m	11,19ms
	A*	10	(-18,5, 8,8)	(4,4,-13,9)	32,27m	39,13m	10,03ms
	NavMesh					38,53m	10,80ms
Žemėlapis su statine kliūtimi	A*	1	(-0,7, 11,1)	(-19,9,-5,5)	25,33m	34,19m	19,70ms
	NavMesh					33,54m	11,09ms
	A*	2	(5,4, 2,8)	(-11,5,-7,0)	19,50m	24,61m	17,83ms
	NavMesh					25,39m	12,34ms
	A*	3	(17,9, -8,4)	(14,0,19,3)	28,03m	39,20m	16,63ms
	NavMesh					35,71m	11,06ms
	A*	4	(2,4, -6,7)	(-5,8,-8,0)	8,31m	10,93m	19,50ms
	NavMesh					10,78m	15,98ms
	A*	5	(9,0, -2,5)	(7,9,-11,7)	9,28m	11,92m	16,18ms
	NavMesh					12,17m	12,50ms
	A*	6	(-12,5, -3,2)	(-8,4,-16,7)	14,09m	19,68m	14,90ms
	NavMesh					19,12m	14,95ms
	A*	7	(-9,1, 17,3)	(4,9,-10,1)	30,77m	38,97m	20,34ms
	NavMesh					42,85m	12,08ms
	A*	8	(5,2, -14,4)	(2,8,-13,6)	2,55m	3,57m	17,46ms
	NavMesh					3,39m	12,24ms
	A*	9	(13,8, -15,7)	(-3,7,19,2)	38,99m	54,44m	15,91ms
	NavMesh					53,81m	15,61ms
	A*	10	(-1,8, 18,1)	(-5,5,16,6)	3,99m	5,41m	19,08ms
	NavMesh					5,39m	13,90ms
Žemėlapis su dinamiškai atsirandančia kliūtimi	A*	1	(14,8, -11,8)	(7,7,19,0)	31,61m	43,26m	25,39ms
	NavMesh					39,48m	15,29ms
	A*	2	(7,3, -14,6)	(16,2,10,7)	26,85m	33,55m	23,49ms
	NavMesh					31,36m	15,90ms
	A*	3	(13,2, -7,4)	(-3,0,-8,1)	16,23m	21,73m	19,91ms
	NavMesh					19,28m	13,82ms
	A*	4	(5,4, 14,9)	(-11,3,-1,2)	23,21m	31,72m	18,23ms
	NavMesh					27,05m	14,92ms
	A*	5	(18,4, -5,1)	(-2,7,13,7)	28,28m	36,82m	26,00ms
	NavMesh					34,18m	16,32ms
	A*	6	(10,5, -14,3)	(11,6,-16,4)	2,44m	3,29m	19,26ms

	NavMesh					2,97m	14,32ms
	A*	7	(16,0, -14,9)	(-14,8,13,7)	41,95m	56,79m	22,20ms
	NavMesh					51,13m	14,01ms
	A*	8	(7,4, 10,0)	(-4,4,-5,1)	19,13m	23,15m	17,61ms
	NavMesh					21,80m	13,14ms
	A*	9	(12,5, 12,4)	(-19,4,-2,8)	35,27m	44,56m	21,58ms
	NavMesh					43,97m	11,53ms
	A*	10	(9,6, -5,1)	(-1,2,-7,5)	11,03m	14,70m	18,50ms
	NavMesh					13,24m	11,44ms

Kelio radimo metodo tinklelis	**Eksperimento numeris**	**Apskaičiuoti rezultatai neapdorota forma (Unity Debug.Log)**
A*	1	[A*][GridUpdate] Update at (17,0, -1,6) | Time: 12,64ms
Unity NavMesh		[NavMesh][Update] Update at (17,0, -1,6) | Time: 23,50ms
A*	2	[A*][GridUpdate] Update at (-4,5, -7,7) | Time: 15,47ms
Unity NavMesh		[NavMesh][Update] Update at (-4,5, -7,7) | Time: 37,35ms
A*	3	[A*][GridUpdate] Update at (10,2, -16,6) | Time: 13,13ms
Unity NavMesh		[NavMesh][Update] Update at (10,2, -16,6) | Time: 27,90ms
A*	4	[A*][GridUpdate] Update at (-12,7, 14,6) | Time: 13,48ms
Unity NavMesh		[NavMesh][Update] Update at (-12,7, 14,6) | Time: 18,43ms
A*	5	[A*][GridUpdate] Update at (-19,4, 17,3) | Time: 10,33ms
Unity NavMesh		[NavMesh][Update] Update at (-19,4, 17,3) | Time: 21,05ms
A*	6	[A*][GridUpdate] Update at (-6,2, -13,7) | Time: 15,62ms
Unity NavMesh		[NavMesh][Update] Update at (-6,2, -13,7) | Time: 21,10ms
A*	7	[A*][GridUpdate] Update at (-0,4, 19,8) | Time: 11,62ms
Unity NavMesh		[NavMesh][Update] Update at (-0,4, 19,8) | Time: 37,73ms
A*	8	[A*][GridUpdate] Update at (-15,3, 4,5) | Time: 15,32ms
Unity NavMesh		[NavMesh][Update] Update at (-15,3, 4,5) | Time: 23,24ms
A*	9	[A*][GridUpdate] Update at (-18,2, -15,0) | Time: 10,38ms
Unity NavMesh		[NavMesh][Update] Update at (-18,2, -15,0) | Time: 24,09ms
A*	10	[A*][GridUpdate] Update at (-17,7, -17,1) | Time: 15,58ms
Unity NavMesh		[NavMesh][Update] Update at (-17,7, -17,1) | Time: 39,43ms

4 Blackboard sistemos eksperimentinių testavimų neapdoroti rezultatai

Scenarijus	**Laikas (ms)**	**Neapdoroti rezultatai (Unity Debug.Log)**

Bloga ekonominė situacija	30	Total Buildings Constructed: 2 Average Warehouse Fill Level: 31,4% Resource Production Balance Coefficient: 0,07 - Resource: Gold, Production This Interval: 5 units - Resource: Wood, Production This Interval: 12 units - Resource: Stone, Production This Interval: 5 units - Resource: Iron, Production This Interval: 5 units - Resource: Food, Production This Interval: 5 units
	60	Total Buildings Constructed: 4 Average Warehouse Fill Level: 45,3% Resource Production Balance Coefficient: 0,0436 - Resource: Gold, Production This Interval: 5 units - Resource: Wood, Production This Interval: 14 units - Resource: Stone, Production This Interval: 12 units - Resource: Iron, Production This Interval: 12 units - Resource: Food, Production This Interval: 12 units
	90	Total Buildings Constructed: 4 Average Warehouse Fill Level: 41,4% Resource Production Balance Coefficient: 0,0845 - Resource: Gold, Production This Interval: 5 units - Resource: Wood, Production This Interval: 19 units - Resource: Stone, Production This Interval: 12 units - Resource: Iron, Production This Interval: 12 units - Resource: Food, Production This Interval: 5 units
	120	Total Buildings Constructed: 5 Average Warehouse Fill Level: 46,1% Resource Production Balance Coefficient: 0,0933 - Resource: Gold, Production This Interval: 5 units - Resource: Wood, Production This Interval: 26 units - Resource: Stone, Production This Interval: 12 units - Resource: Iron, Production This Interval: 12 units - Resource: Food, Production This Interval: 5 units
	150	Total Buildings Constructed: 4 Average Warehouse Fill Level: 54,0% Resource Production Balance Coefficient: 0,0395 - Resource: Gold, Production This Interval: 10 units - Resource: Wood, Production This Interval: 12 units - Resource: Stone, Production This Interval: 17 units - Resource: Iron, Production This Interval: 17 units - Resource: Food, Production This Interval: 17 units
	180	Average Decision Time: 5,02 milliseconds Total Buildings Constructed: 4 Average Warehouse Fill Level: 51,7% Resource Production Balance Coefficient: 0,0257 - Resource: Gold, Production This Interval: 15 units - Resource: Wood, Production This Interval: 17 units - Resource: Stone, Production This Interval: 15 units - Resource: Iron, Production This Interval: 15 units - Resource: Food, Production This Interval: 22 units
Gera ekonominė situacija	30	Total Buildings Constructed: 7 Average Warehouse Fill Level: 33,9% Resource Production Balance Coefficient: 0,0185 - Resource: Gold, Production This Interval: 20 units - Resource: Wood, Production This Interval: 29 units - Resource: Stone, Production This Interval: 27 units - Resource: Iron, Production This Interval: 27 units - Resource: Food, Production This Interval: 27 units * Warehouse constructed to prevent overflow.

	60	Total Buildings Constructed: 5 Average Warehouse Fill Level: 44,6% Resource Production Balance Coefficient: 0,0933 - Resource: Gold, Production This Interval: 5 units - Resource: Wood, Production This Interval: 26 units - Resource: Stone, Production This Interval: 12 units - Resource: Iron, Production This Interval: 12 units - Resource: Food, Production This Interval: 5 units
	90	Total Buildings Constructed: 4 Average Warehouse Fill Level: 52,5% Resource Production Balance Coefficient: 0,0257 - Resource: Gold, Production This Interval: 15 units - Resource: Wood, Production This Interval: 17 units - Resource: Stone, Production This Interval: 15 units - Resource: Iron, Production This Interval: 15 units - Resource: Food, Production This Interval: 22 units
	120	Total Buildings Constructed: 6 Average Warehouse Fill Level: 53,7% Resource Production Balance Coefficient: 0,0255 - Resource: Gold, Production This Interval: 15 units - Resource: Wood, Production This Interval: 29 units - Resource: Stone, Production This Interval: 22 units - Resource: Iron, Production This Interval: 22 units - Resource: Food, Production This Interval: 22 units
	150	Total Buildings Constructed: 8 Average Warehouse Fill Level: 47,1% Resource Production Balance Coefficient: 0,0155 - Resource: Gold, Production This Interval: 25 units - Resource: Wood, Production This Interval: 34 units - Resource: Stone, Production This Interval: 32 units - Resource: Iron, Production This Interval: 32 units - Resource: Food, Production This Interval: 32 units
	180	Total Buildings Constructed: 7 Average Warehouse Fill Level: 50,2% Resource Production Balance Coefficient: 0,0195 - Resource: Gold, Production This Interval: 25 units - Resource: Wood, Production This Interval: 27 units - Resource: Stone, Production This Interval: 32 units - Resource: Iron, Production This Interval: 32 units - Resource: Food, Production This Interval: 32 units * Warehouse constructed to prevent overflow.
Labai gera ekonominė situacija	30	Total Buildings Constructed: 10 Average Warehouse Fill Level: 32,1% Resource Production Balance Coefficient: 0,0305 - Resource: Gold, Production This Interval: 25 units - Resource: Wood, Production This Interval: 41 units - Resource: Stone, Production This Interval: 39 units - Resource: Iron, Production This Interval: 39 units - Resource: Food, Production This Interval: 32 units
	60	Total Buildings Constructed: 6 Average Warehouse Fill Level: 35,5% Resource Production Balance Coefficient: 0,0255 - Resource: Gold, Production This Interval: 15 units - Resource: Wood, Production This Interval: 29 units - Resource: Stone, Production This Interval: 22 units - Resource: Iron, Production This Interval: 22 units - Resource: Food, Production This Interval: 22 units

	90	Total Buildings Constructed: 9 Average Warehouse Fill Level: 28,0% Resource Production Balance Coefficient: 0,0294 - Resource: Gold, Production This Interval: 20 units - Resource: Wood, Production This Interval: 29 units - Resource: Stone, Production This Interval: 34 units - Resource: Iron, Production This Interval: 34 units - Resource: Food, Production This Interval: 27 units
	120	Total Buildings Constructed: 7 Average Warehouse Fill Level: 37,5% Resource Production Balance Coefficient: 0,0185 - Resource: Gold, Production This Interval: 20 units - Resource: Wood, Production This Interval: 29 units - Resource: Stone, Production This Interval: 27 units - Resource: Iron, Production This Interval: 27 units - Resource: Food, Production This Interval: 27 units
	150	Total Buildings Constructed: 6 Average Warehouse Fill Level: 41,9% Resource Production Balance Coefficient: 0,017 - Resource: Gold, Production This Interval: 25 units - Resource: Wood, Production This Interval: 32 units - Resource: Stone, Production This Interval: 25 units - Resource: Iron, Production This Interval: 25 units - Resource: Food, Production This Interval: 25 units
	180	Total Buildings Constructed: 10 Average Warehouse Fill Level: 42,4% Resource Production Balance Coefficient: 0,0171 - Resource: Gold, Production This Interval: 35 units - Resource: Wood, Production This Interval: 42 units - Resource: Stone, Production This Interval: 42 units - Resource: Iron, Production This Interval: 42 units - Resource: Food, Production This Interval: 35 units