KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Audrius Kulikajevas

KELIO PAIEŠKOS ALGORITMŲ TAIKYMŲ ROBOTIKOJE TYRIMAS

Magistro baigiamasis projektas

Vadovas
lekt. dr. Mikas Binkis

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

KELIO PAIEŠKOS ALGORITMŲ TAIKYMŲ ROBOTIKOJE TYRIMAS

Magistro baigiamasis projektas
Programų sistemų inžinerija (621E16001)

Vadovas
(parašas) lekt. dr. Mikas Binkis
(data)

Recenzentas
(parašas) prof. dr. Alfonsas Misevičius
(data)

Projektą atliko
(parašas) Audrius Kulikajevas
(data)

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos

(Fakultetas)

Audrius Kulikajevas

(Studento vardas, pavardė)

Programų sistemų inžinerija, 621E16001

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Vizualaus robotų programavimo ir robotų sąveikos simuliacinės aplinkos kūrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužės 18 d.

Kaunas

Patvirtinu, kad mano, **Audriaus Kulikajevo**, baigiamasis projektas tema „Kelio paieškos algoritmų taikymų robotikoje tyrimas“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka) *(parašas)*

Audrius, K. „Kelio paieškos algoritmų taikymų robotikoje tyrimas“. Magistrinio baigiamasis projektas / vadovas lekt. dr. Mikas Binkis; Kauno technologijos universitetas, Informatikos fakultetas. Kaunas, 2017. 56 p.

SANTRAUKA

Darbo metu sukurta sistema vizualiam robotų programavimui ir robotų sąveikos modeliavimo tyrimui – *RobotVL*. Sukurta sistema jos vartotojui leidžia panaudojus grafiškai kuriamus algoritmus sukurti algoritmus robotams bei išbandyti tuos algoritmus virtualioje aplinkoje.

Tyrimo pagrindinis tikslas – ištirti kelio planavimo algoritmus bei atrinkti tinkamiausią (ar jų kombinaciją) algoritmą kuris bus įdiegiamas sukurtoje sistemoje. Parinktas algoritmas yra pradedantiesiems palengvinti darbą su mobiliaisiais robotais. Taip pat tai yra kaip mokomoji medžiaga kurią vartotojas gali išanalizuoti bei pakeitus stebėti kaip keičiasi roboto elgsena.

Tyrimas buvo atliekamas su pasirinktais universaliais algoritmais galinčiais dirbti tiek fizinėje, tiek realioje aplinkoje. Algoritmai kurie būtų neįgyvendami fizinėje aplinkoje buvo atmetami kadangi sistema turi leisti vartotojui tą patį sukurtą roboto algoritmą pakartotinai panaudoti ir tikrame robote.

Algoritmai buvo analizuojami pagal jų vykdymo greitį, operatyviosios atminties sąnaudos reikalingas darbo metu bei rasto kelio optimalumą. Tyrimo rezultatų metu buvo atrinkti du algoritmai kurių kombinacija yra tinkamiausia kuriamoje sistemoje.

Audrius, K. “Research of pathfinding algorithms application in field of robotics”. Master’s final project / supervising lect. ph. d. Mikas Binkis; Kaunas University of Technology, Faculty of Informatics.

Kaunas, 2017. 56 p.

SUMMARY

In this work, we have created software for visual robot software development and their interactions observation – *RobotVL*. Developed software allows its user to graphically create algorithms for robots and test them inside a virtual environment.

The purpose of the research is to examine path planning algorithms and to propose the most suited (or a combination of) algorithm to implement in the developed software. Selected algorithm allows a beginner user to get into mobile robotics easier alongside being a tool for learning by modifying it and observing how changes to the algorithm impact the behavior of a mobile robot.

The research was conducted with universal algorithms that are not tied to specific environment. Algorithms that are impossible to implement in physical environment were rejected due to requirement that the system needs to be able to reuse the same graphical algorithm for both virtual and physical environments.

Selected algorithms were analyzed by their performance, operating memory usage and whether they could find the most optimal path or not. Research result suggests a combination of two algorithms to implement in the system which would be the most optimal and universal solution for the task at hand.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

Terminas ar santrumpa	Aprašymas
Asemblis	Tai *.NET* karkaso kalbų komponentas. Asemblis – logiškai susietų tipų bei išteklių visuma skirta suteikti kokiam nors funkcionalumui.
Binarinis operatorius	Operatorius priimantis du operandus, paprastai vieną iš kairės, kitą iš dešinės. Pavyzdžiui sudėtis, daugyba.
Liuksmetras	Aparatūrinė įranga parodanti aplinkos apšvietą apšvietos vienetais – liuksais.
CLI (angl. *command line interface*)	Tai viena primityviausių sąsajų tarp vartotojo bei kompiuterio. Vartotojui suteikiama komandinė eilutė kurioje vartotojas gali nurodyti komandas kurias turėtų atlikti kompiuteris.
Mobilieji robotai	Tai robotų rūšis kuri gali judėti aplinkoje. Toki robotai gali turėti įvairius judėjimo laisvės laipsnius. Pavyzdžiui: judėti erdvėje tik plokštuma ant kurios stovima arba skraidyti.
Kelio planavimas	Tai tinkamo kelio iš taško A į tašką B radimo problema.
Aplinkos tyrimas/tyrinėjimas	Tai robotikoje aktuali problema. Aplinkos tyrinėjimas yra būdas supažindinti robotą su jo judėjimo aplinka. Šios problemos sprendinys – dalinai arba pilnai ištirtos aplinkos vaizdas atvaizduojamas grafo pavidalu.
API (angl. *application programming interface*)	Sąsaja, kurią suteikia kompiuterinė sistema, biblioteka ar programa tam, kad programuotojas per kitą programą galėtų pasiekti jos funkcionalumą ar apsikeistų su ja duomenimis.
Statinė analizė	Tai būdas aptikti galimas klaidas programiniame kode dar prieš jį paleidžiant.
Refleksija	Būdas programos metu, pačiai programai išanalizuoti savo pačios programinį kodą, duomenų struktūras, bei jas modifikuoti vykdymo metu.

1. ĮVADAS

1.1. Darbo problematika ir aktualumas

Programinė įranga – neatsiejama mūsų gyvenimo dalis, tačiau retas asmuo išmano kaip yra kuriama programinė įranga. Egzistuoja daugybė programavimo mokymuisi skirtų įrankių – knygos, video gidai ir kt. Tačiau žmones nuo savarankiško programavimo mokymosi atgraso storos knygos, internete pilna vaizdo pamokų mokančių programuoti, tačiau tokios pamokos dažnai užima labai daug laiko ir siūlo labai prastą laiko-žinių santykį. O mokyklose su programavimo disciplina supažindinama labai vėlai, taipogi CLI programavimas daugelio nesudomina. Šioms problemoms spęsti labai tinka robotai, kadangi galima ištirti peržiūrėti kaip programinis kodas sąveikauja su realiu pasauliu. Tokis mokymosi būdas gali būti kaip savotiškas žaidimas, tačiau robotai gali būti brangūs. Todėl šiai problemai spręsti siūloma sukurti sistemą kuri imituos fizinį pasaulį bei leis joje modeliuoti robotus.

Kuriama sistema skirta sudominti bei supažindinti vartotoją su programavimo disciplina. Dėl šios priežasties šis projektas gali būti labai aktualus ir ateityje. Programinei įrangai tampant vis didesne mūsų gyvenimo dalimi didės ir programuotojų paklausa. Taip pat programavimas kaip įgūdis gali praversti ir kasdieniniame gyvenime.

1.2. Darbo tikslas ir uždaviniai

Siekiant išspręsti ankščiau minėtą problemą siūlome sprendimą – *RobotVL* programinę įrangą. Tai sistema leidžianti vartotojams išmokti programuoti bei taikyti tuos pačius algoritmus robotų valdymui tiek virtualiems, tiek fiziniams robotams. Šis sprendimas leis vartotojams neturintiems jokių išankstinių programavimo bei robotikos žinių sukurti bei išbandyti algoritmus virtualioje aplinkoje. Šiam tikslui pasiekti reikia įvykdyti šias užduotis:

1. Išanalizuoti egzistuojančius panašius sprendimus.
2. Išanalizuoti kelio planavimo bei aplinkos tyrinėjimo algoritmus.
3. Suprojektuoti bei sukurti grafinio programavimo posistemę.
4. Suprojektuoti bei sukurti vizualaus kodo transformavimo posistemę.
5. Suprojektuoti bei sukurti virtualią aplinką kurioje būtų galima išbandyti (modeliuoti) sukurtus algoritmus.
6. Suprojektuoti bei sukurti sąsają per kurią robotai gali komunikuoti su modeliavimo posisteme. Sąsaja taip pat reikalingą tam jog robotus būtų galima įkrauti dinamiškai.
7. Ištirti kelio planavimo algoritmų veikimą sukurtoje aplinkoje.
8. Apibendrinti gautus tyrimo rezultatus. Pasiūlyti algoritmą kurį reikia įgyvendinti grafinio programavimo aplinkoje kaip sintaksės mazgą.

Atlikus šias užduotis bus sukurta programinė įranga skirta supažindinti jos vartotojus su programavimu bei robotika.

1.3. Darbo struktūra

Darbas susideda iš šių dalių:

1. Vizualaus kodo transformavimo bei modeliavimo aplinkos analizės – analizės metu aptariama tyrimo sritis bei objektas. Apibūdinamos pagrindinės vartotojo charakteristikos. Pateikiama kelio planavimo bei aplinkos tyrimo algoritmų literatūros analizė.
2. Reikalavimų specifikacijos – reikalavimų specifikacijos metu pateikiami tiek funkciniai, tiek nefunkciniai sistemos reikalavimai. Apibūdinama aplinka kurioje veiks sistema.
3. Sistemos projekto – skyriuje pateikiamas sistemos pagrindimas bei jos architektūra. Taip pat pateikiamas sistemos elgsenos modelis modeliavimo posistemei.
4. Sistemos realizacijos – pateikiami sistemos funkcionalumo įgyvendinimui priimti sprendimai. Pateikiamas bei detalizuojamas testavimo planas.
5. Eksperimentinio algoritmų tyrimo – skyriuje pateikiami tiriami kelio planavimo algoritmai. Pateikiama eksperimento planas, eiga bei rezultatai. Pateikiami kriterijai pagal kuriuos analizuojami algoritmai. Pagal tyrimo rezultatus pasiūlomi kelio planavimo algoritmai kuriuos reikia įgyvendinti sistemoje.

2. VIZUALAUS KODO TRANSFORMAVIMO BEI MODELIAVIMO APLINKOS ANALIZĖ

2.1. Analizės tikslas

Analizės metu bus pateikiami kelio planavimo algoritmų apžvalga kurie bus įtraukiami į kuriamą sistemą. Taip pat analizės metu bus apibūdinama tyrimo sritis bei jos objektas, aptariama iškilusi problema. Nurodomi sistemos vartotojai bei pateikiamos pagrindinės jų charakteristikos. Analizėje taip pat pateikiamas aplinkos tyrimo algoritmas skirtas aplinkos grafo sudarymui.

Pateikiami kokybės kriterijai, nurodantys, kokį kokybės lygį reikia pasiekti tam, kad sistemą, būtų galima klasifikuoti kaip išbaigtą.

2.2. Tyrimo sritis, objektas ir problema

2.2.1. Tyrimo sritis bei tikslas

Eksperimentinio tyrimo metu bus tiriami įvairūs kelio planavimo algoritmai. Pagal gautus tyrimo rezultatus, tinkamiausias (arba keli sujungti į vieną) taikymo sričiai algoritmas bus naudojamas robotų modeliavimo aplinkoje. Šį algoritmą sistema pateiks kaip vienetinį mazgą, kurį vartotojas gali panaudoti savo kuriamoje vizualioje programoje.

Pasirinktas(-i) algoritmas turėtų veikti realiu laiku bei būti nepririštas prie virtualaus pasaulio realizacijų, kurių nebūtų galima pritaikyti realaus pasaulio aplinkoms. Pavyzdžiui – robotas negali žinoti apie pasikeitimus pasaulyje realiu laiku, jeigu tam nėra naudojamas specialus daviklis.

2.2.2. Tyrimo objektas

Magistrinio darbo tyrimo objektas – kelio planavimo algoritmai bei jų pritaikymas robotams. Šie robotai yra realaus pasaulio robotų atitikmuo virtualioje erdvėje. Šie robotai imituoja visas realaus pasaulio fizikines savybes, virtualūs robotai neturi jokių išankstinių žinių apie juos supantį pasaulį – visą informaciją gauna kaip ir fiziniai robotai – prijungus jutiklius. Šie jutikliai padės robotams išvengti kliūčių bei atlikti jiems paskirtas užduotis.

2.2.3. Tiriama problema

Siekiant sukurti lengvai prieinamą kiekvienam vartotojui robotų programavimui skirtą sistemą, reikia vartotojui pasiūlyti jau paruoštus algoritmus. Viena iš bene svarbiausių mobiliųjų robotų problemų – kelio planavimas. Mobilieji robotai – tai bet kokie robotai galintys judėti erdvėje. Šiai problemai išspręsti sistema turi pasiūlyti pradedančiajam vartotojui paruoštus algoritmų mazgus. Realizuoti standartiniai algoritmai padės vartotojui greičiau perprasti darbo eigą, neatgrasant jo iš anksto reikalingų turėti žinių kiekiu, o vartotojui panorėjus, praplėsti žinias, šis bet kada galės algoritmus įgyvendinti pats, taip praplėsdamas savo žinias dar giliau. Dėl šios priežasties darbe bus tiriami įvairūs kelio planavimo algoritmai.

Trumpiausio kelio paieška (kelio planavimas) yra labai svarbus uždavinys tiek realiame, tiek virtualiame pasaulyje. Šių uždavinių sprendimai gali būti naudojami maršrutų optimizavimui, integruotų lustų testavimui, labirinto kelio radimui [1] ir net vaizdo žaidimų dirbtinio intelekto valdomų agentų valdymui [2]. Šiems uždaviniams spręsti yra įvairiausių būdų, skirtų skirtingiems scenarijams bei aplinkoms. Šie algoritmai dažniausiai yra pagrįsti grafų teorija.

Kelio paieška taip pat neatsiejama nuo robotikos. Šis uždavinys yra vienas iš fundamentalių mobiliųjų robotų keliamų uždavinių. Dėl to šio uždavinio sprendimas yra labai aktualus kuriant mobilius robotus – tiek realius, tiek virtualius.

2.3. Vartotojų analizė

2.3.1. Pagrindinės vartotojų charakteristikos

Vartotojas, į kurį sutelkiamas dėmesys kuriant sistemą bei jos posistemes – mokyklinio amžiaus asmuo, neturintis jokio arba turintis tik minimalų suvokimą apie programavimą. Nustatomos šios pagrindinės vartotojo charakteristikos:

1. tai yra jaunas žmogus, orientuojamasi į vidurinių klasių amžiaus grupę (4–10kl.);
2. jokių arba minimalios programavimo bei algoritmų sudarymo žinios;
3. žemo lygio kompiuterinio raštingumo žinios.

2.3.2. Vartotojo problemos

Programavimo sintaksė bei įprastiniai programavimo įrankiai yra ganėtinai bauginantys neturintiems patirties šioje srityje. Egzistuoja labai daug literatūros programavimo tematika, tačiau neretam šiuolaikiniam jaunuoliui knygos yra nuobodu. Dėl šios priežasties gali būti ganėtinai sunku sudominti vaikus šia mokslo sritimi.

Kuriamas produktas sieks išspręsti šias problemas tokiais būdais:

1. pateikiant paprastą, aiškią bei lengvai suprantamą sąsają;
2. paslepiant kodo rašymą po loginiais ryšių blokais;
3. pasiūlant standartinius algoritmus robotų valdymui, taip sumažinant žinių poreikį;
4. patiekiant programavimą kaip žaidimą.

Vizualaus kodo transformavimo posistemė sprendžia problemą, kaip pateikti standartinius algoritmus robotų valdymui, siekiant sumažinti išankstinių žinių poreikį galutiniam vartotojui.

Modeliavimo posistemė sprendžia programavimo pateikimo kaip žaidimo problemą.

2.4. Kelio planavimo algoritmai

Kelio planavimas yra labai sena problema, kuriai spręsti yra sukurta daugybė įvairių sprendimų. Šiuos algoritmus būtų galima skirstyti į dvi pagrindines dalis – deterministiniai bei tikimybiniai. Egzistuoja ypač daug šios problemos sprendimo būdų, kiekvienas iš jų turi savo

pranašumų bei trūkumų. Vieni labai specializuoti, kaip pavyzdžiui, pririštasis robotas. Toks algoritmas leidžia robotui kirsti savo pačio giją, bet nei gija, nei pats robotas niekada negali kirsti kliūties [3].

Kiti yra bendriniai, kuriuos galima pritaikyti bet kokiai aplinkai. Bene pati populiariausia trumpiausio kelio problemos sprendimo algoritmų rūšis yra deterministiniai.

Dažniausiai pasitaikantys kelio paieškos algoritmai yra pagrįsti grafų teorija. Tokie algoritmai ieško geriausio kelio grafe. Geriausias kelias nėra būtinai trumpiausias, kartais reikalingi algoritmai, kurie suranda kelią per tam tikrą kritinį laiką ar pagal kitas aplinkybes.

Tyrimo metu bus analizuojami grafų teorija pagrįsti algoritmai. Darbo siekis yra išanalizuoti galimus kelio paieškos algoritmus bei pasirinkti algoritmą, labiausiai tinkantį kuriamai sistemai. Šie algoritmai turėtų būti bendriniai, jie negali būti susieti su specifine situacija – t. y. veikti tik virtualioje aplinkoje. Šiuos algoritmus perkėlę į realųjį pasaulį turėtumėme gauti tokius pačius arba labai panašius algoritmus.

2.4.1. Dijkstros algoritmas

Šis „godus“ algoritmas yra skirtas trumpiausių kelių radimui tarp svorinio grafo viršūnių. Algoritmas buvo pasiūlytas [4] kaip sprendimas šioms problemoms:

1. Medžio, turinčio mažiausią ilgį tarp *n* viršūnių, radimui;
2. trumpiausio kelio tarp dviejų viršūnių *P* ir *Q* radimui.

Egzistuoja įvairių šio algoritmo adaptacijų, tačiau pirmasis pasiūlytas algoritmas buvo skirtas rasti keliui tarp dviejų viršūnių. Originalus algoritmas nenaudojo prioritetų eilės, dėl šios priežasties jo asimptotinis laikas buvo gana prastas. Šio algoritmo asimptotinis laikas yra:

$O(n^2)$,

kur n yra viršūnių skaičius. Tačiau egzistuoja ir šio algoritmo variantas prioritetų eilės įgyvendinimui, naudojantis *Fibonačio* kupetą. Šis algoritmo variantas yra įrodytas kaip veikiantis greičiausiu asimptotiniu laiku. Jo asimptotinis laikas blogiausiu atveju yra:

$$O(e + nlogn),$$

kur n yra viršūnių skaičius, o e yra grafo briaunų skaičius. Specialus *Dijkstros* atvejis yra paieška platyn, kai visų grafo briaunų svoriai yra lygūs vienam.

2.4.2. A* algoritmas

*A** (arba *A-Star*) algoritmas [5], jo variantai [6] [7] yra vienas iš paprasčiausių, bet praktiškų kelio planavimo algoritmų. Šiam algoritmui naudojama euristinė metodologija, siekiant paiešką vykdyti link tikslo [8]. *A** algoritmas pagrįstas prioritetų eile, kai parenkamas pirmas geriausias paieškos rezultatas. Algoritmui keliaujant grafu, kuriamas grafo dalinių ryšių medis [9]. Kelio planavimas yra pagrįstas *Dijkstros* algoritmu. Tačiau dėl naudojamos euristikos klasikinio *A-Star* algoritmo kainos aproksimacija yra daug geresnė nei klasikinio *Dijkstros*. Šio algoritmo kainos aproksimacija yra:

$$f(u) \equiv g(u) + h(u),$$

kur u yra pasirinkta viršūnė, $g(u)$ yra žinoma kaina patekti iš pradinio taško į u, o $h(u)$ yra euristinis įvertinimas kainos, norint patekti iš u į bet kurį kitą viršūnę. Dėl algoritmo tikslumo bei ganėtinai geros greitaveikos jis tapo vienu iš labiausiai naudojamų vaizdo žaidimų dirbtinio intelekto algoritmų [10].

2.4.3. *D** algoritmas

*D** (arba *D-Star*) algoritmas [11] yra prielaidomis pagrįstas kelio planavimo algoritmas. Šis algoritmas kilo iš *A-Star* algoritmo, siekiant jį paversti dinaminiu. Dėl šios priežasties abiejų šeimų algoritmai elgiasi panašiai, tačiau *D-Star* algoritmo grafų briaunų svoriai gali kisti. Tiek *A-Star*, tiek *D-Star* algoritmai gali veikti nežinomame grafe, tačiau jei ieškoma viršūnė keis poziciją, *D-Star* visais atvejais veiks tokiu pačiu arba didesniu greičiu. Klasikinis *D-Star* algoritmas gerokai lenkia *A-Star* algoritmą, kai grafo viršūnių skaičius didelis. Dėl šios priežasties jis tapo vienu populiariausių algoritmų mobiliems robotams kurti.

D-Star algoritmas taip pat turi supaprastintą versiją. *D-Star Lite* [12] [6] nėra pagrįstas klasikine versija, tačiau jo našumas yra toks pat ar net geresnis, o pati realizacija yra žymiai paprastesnė. Ši algoritmo versija yra dažniausiai naudojama mobiliuosiuose robotuose.

D-Star algoritmas veikia „bangavimo“ principu. Kai kiekviena mazgo viršūnė nustato savo būseną pagal tai, kokioje būsenoje yra jos kaimynai. Kaip ir *Dijkstros* bei *A-Star*, šis algoritmas palaiko aplankytų viršūnių sąrašą. Tačiau kitaip nei kiti algoritmai, *D-Star* taip pat turi ir būseną. Naudojant šį algoritmą viršūnės gali būti vienoje iš šių būsenų:

- nauja – reiškia, jog ši viršūnė nei karto nebuvo aplankyta, ši būsena suteikiama visoms viršūnėms jų sukūrimo metu;
- atvira – ši būsena suteikiama viršūnei, kuri pateko į aplankytų sąrašą;
- uždaryta – algoritmas suteikia šią būseną visoms viršūnėms, kai jos panaikinamos iš aplankytųjų sąrašo;
- pakilus – ši būsena nurodo, jog viršūnės kaina padidėjo nuo paskutinio karto, kai ji buvo aplankytų sąraše;
- sumažėjus – atvirkštinė būsena pakilusiai. Šioje būsenoje viršūnės būna tada, kai jų kaina sumažėja nuo paskutinio aplankymo.

Pakilusios būsena suteikiama visoms viršūnėms, į kurias patekti reikės daugiau pastangų (ilgiau užtruks) nei praeitos iteracijos metu. Tai dažniausiai atsitinka atsiradus kliūčiai kelyje. Dingus kliūčiai, būsena pakeičiama į sumažėjusią. Viršūnės aptikusius pokyčius kaimynuose perskaičiuoja į savo kainas ir atitinkamai nustato savo būsenas.

Dėl savo veikimo principo D* gali veikti tik su 45 arba 90 laipsnių tinkleliu. Todėl tai nėra tinkamas sprendimas kuriamai sistemai. Taip pat šis algoritmas turi nuolatos žinoti kiekvienos iš grafo viršūnių būseną, kas nėra realistiška aplinkoje kurioje naudojama sistema.

2.5. Aplinkos tyrimo algoritmai

Grafo sudarymas (tyrinėjimas) yra atskira problema, tik iš dalies susijusi su pačiu kelio planavimu. Dėl šios priežasties numatome, jog robotas jau yra ištyrinėjęs aplinką. Vadinasi, robotas bent iš dalies jau žino reliatyvias grafo viršūnių pozicijas. Nors kai kuriems algoritmams, kurie bus aptariami, ši prielaida nėra būtina, tačiau turint bendrą grafą roboto atmintyje elgsena bus optimalesnė.

2.5.1. *RRT** algoritmas

RRT [13] – greitos paieškos atsitiktiniai medžiai. Tai paieškos algoritmas, skirtas neišgaubtoms erdvėms. *RRT* medis kuriamas iteraciniu būdu, atsitiktinai užpildant neaplankytas vietas. *RRT* tinka kliūčių aptikimo algoritmams, dėl šios priežasties jį galima panaudoti kaip grafo sudarymo algoritmą. *RRT** (arba *RRT-Star*) algoritmas [14] yra praplėsta *RRT* versija, veikianti atrankos principu. Pagrindinis skirtumas tarp jų yra tas, jog *RRT-Star* papildomai išlaiko aplankytų kelių kainas bei atlieka optimalų grafų briaunų perkonfigūravimą, tam kad būtų išlaikomas optimaliausias kelias iš bradinio taško į bet kurią kitą viršūnę [9].

Įrodyta, jog *RRT-Star* algoritmas yra asimptotiškai optimalus. Kadangi pritaikius šį algoritmą galima gauti visą grafą, kuris apeina visas kliūtis, jį galima panaudoti su kelio planavimo algoritmais kaip *Sampling-based A-Star*, kuris adaptuotas įprasto *A-Star* algoritmo variantas, tačiau jam pritaikytas *RRT-Star* bei *RRG* algoritmas. Taip pat *Sampling-based* A-Star nėra deterministinis, o tikimybių teorija pagrįstas algoritmas.

2.6. Kokybės kriterijai faktoriai

Kokybišką algoritmą bei jo realizaciją galima traktuoti įvairiai. Egzistuoja daugelis kelio planavimo algoritmų, kuriuos galime panaudoti kuriamoje sistemoje. Kiekvienas jų turi savo privalumų bei trūkumų. Tyrimo metu ištirtų algoritmų kokybė yra vertinama pagal tai, kaip jie sugebėjo susidoroti su pateikta užduotimi.

Algoritmų kokybei nustatyti buvo panaudotos šios metrikos:

1. Algoritmo greitaveika.
2. Kiek pasirinktas kelias nuklydo nuo trumpiausio kelio.
3. Kiek algoritmui reikia prisiminti informacijos, t. y. – kokios yra atminties sąnaudos.

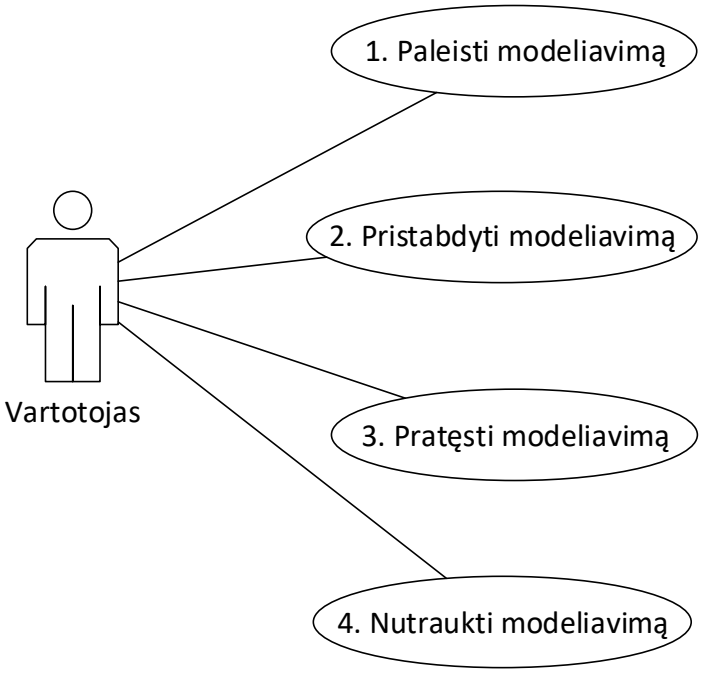
Atsižvelgiant į šiuos kriterijus tinkamiausias algoritmas kuriamai sistemai bus įgyvendinamas kaip programinis vienetas *RobotVL* sistemoje. Šis mazgo komponentas bus atsparos taškas vartotojams pradedantiems darbą su mobiliaisiais robotais.

2.7. Išvados

1. Buvo apžvelgtas grafų sudarymo algoritmas – *RRT**. Šis algoritmas yra tinkamas sprendimas sudaryti grafams trimatėje aplinkoje kurioje robotas gali judėti tik dvejomis ašimis. Dėl šios priežasties tai yra tinkamas aplinkos analizės algoritmas kuriamai sistemai
2. Analizės metu buvo apžvelgti egzistuojantys kelio planavimo sprendimai. Tačiau analizės metu pastebėta jog ne visi egzistuojantys yra tinkai kuriamai sistemai. Dėl šios priežasties algoritmai kurie nėra universalūs arba nepritaikomi kuriamiems robotams priskiriami kaip netinkami įdiegti kuriamoje sistemoje.
3. Visi analizuoti algoritmai buvo pagrįsti grafų teorija. Taip pat pastebėta jog didelė dalis trumpiausio kelio radimo algoritmų yra paremti *Dijkstros* algoritmo pagrindu, dėl jo sugebėjimo rasti globaliai optimalų sprendimą visam grafui.

3. REIKALAVIMŲ SPECIFIKACIJA IR ANALIZĖ

3.1. Reikalavimų specifikacija



3.1 pav. Robotų modeliavimo aplinkos posistemės panaudos atvejų diagrama

3.1 lentelė. Panaudos atvejų lentelė.

PA Nr.	PA Pavadinimas	Aktorius	Efektas
1	Paleisti modeliavimą	Vartotojas	Paleidžiamas virtualaus roboto modeliavimas naudojant sukurtą vartotojo kodą.
2	Pristabdyti modeliavimą	Vartotojas	Laikinai pristabdomas roboto modeliavimas.
3	Pratęsti modeliavimą	Vartotojas	Pratęsiamas laikinai sustabdytas modeliavimas su (jeigu tokių būta) kodo pakeitimais.
4	Nutraukti modeliavimą	Vartotojas	Nutraukiamas roboto modeliavimas.

3.2 lentelė. PA „1 Paleisti modeliavimą“

PA „1 Paleisti modeliavimą“	
Aktorius	Vartotojas
Aprašas	Šiame PA pradedamas virtualaus roboto modeliavimas.
Prieš sąlyga	Vartotojas yra roboto programos kūrimo lange, bei nori pradėti jo modeliavimą virtualioje aplinkoje.
Sužadinimo sąlyga	Vartotojas paspaudė modeliavimo pradžios mygtuką.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia pradėti modeliavimą.	1.1. Sistema pradeda modeliuoti roboto veikimą pagal vartotojo pateiktas instrukcijas.
2. Baigiamas PA.	
Po sąlyga	Pradėtas roboto modeliavimas.

3.3 lentelė. PA „2 Pristabdyti modeliavimą“

PA „2 Pristabdyti modeliavimą“	
Aktorius	Vartotojas
Aprašas	Vartotojas gali laikinai sustabdyti roboto modeliavimą, pakeisti kodą ir tęsti darbą. Šis PA leidžia vartotojui pristabdyti modeliavimo vykdymą.
Prieš sąlyga	Vartotojas yra pradėjęs modeliavimą ir nori laikinai sustabdyti kodo vykdymą.
Sužadinimo sąlyga	Vartotojas paspaudė sustabdymo mygtuką.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia pradėti modeliavimą.	1.1. Sistema įjungia modeliavimą.
2. Vartotojas paspaudžia pristabdyti modeliavimą.	2.1. Sistema laikinai pristabdo roboto modeliavimą.
3. Baigiamas PA.	
Po sąlyga	Modeliavimas laikinai sustabdyta.

3.4 lentelė. PA 3 „Pratęsti modeliavimą“

PA „3 Pratęsti modeliavimą“	
Aktorius	Vartotojas
Aprašas	Vartotojas gali laikinai pristabdyti virtualaus roboto modeliavimą, pakeisti kodą ir tęsti darbą. Šis PA leidžia vartotojui pratęsti modeliavimo vykdymą.
Prieš sąlyga	Vartotojos nori pratęsti roboto modeliavimą virtualioje aplinkoje.
Sužadinimo sąlyga	Vartotojas paspaudė modeliavimo pratęsimo mygtuką.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**

1. Vartotojas paspaudžia pradėti modeliavimą.	1.1. Sistema pradeda roboto modeliavimą virtualioje aplinkoje.
2. Vartotojas paspaudžia pristabdyti modeliavimą.	2.1. Sistema laikinai pristabdo modeliavimo vykdymą.
3. Vartotojas paspaudžia pratęsti modeliavimą.	3.1. Sistema vėl pradeda modeliavimą, jeigu yra kodo pakeitimų vykdomas kodas yra atnaujinamas.
4. Baigiamas PA.	
Po sąlyga	Modeliavimas tęsiamas nuo ten kur buvo sustabdytas.

3.5 lentelė. PA „4 Nutraukti modeliavimą“

PA „4 Nutraukti modeliavimą“	
Aktorius	Vartotojas
Aprašas	Šiame PA nutraukiamas modeliavimo vykdymas visiškai.
Prieš sąlyga	Vartotojos nori nutraukti vykdomą roboto modeliavimą.
Sužadinimo sąlyga	Vartotojas paspaudė roboto modeliavimo nutraukimo mygtuką.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia pradėti modeliavimą.	1.1. Sistema įjungia modeliavimą.
2. Vartotojas paspaudžia nutraukti modeliavimą.	2.1. Sistema išjungia modeliavimo veikimą.
3. Baigiamas PA.	
Po sąlyga	Modeliavimas nutrauktas.

3.1.1. Funkciniai reikalavimai

Reikalavimo Nr.	1
Aprašymas	
Įgyvendinti standartinius robotų algoritmus grafinėje aplinkoje kaip loginius mazgus.	
Atitikimo kriterijus	
Įgyvendintas bent vienas kliūčių išvengimo, kelio paieškos algoritmas kaip loginis mazgas kurį galima įsidėti į savo kuriamą robotą.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	5

Reikalavimo Nr.	2
Aprašymas	
Vizualios kalbos vykdymas modeliavimo posistemėje.	
Atitikimo kriterijus	
Vizualia programavimo kalba sukurti programų kodų algoritmai vykdomi modeliavimo posistemėje.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	5

Reikalavimo Nr.	3
Aprašymas	
Galimybė valdyti modeliavimo procesą pagal poreikį.	
Atitikimo kriterijus	
Vartotojas gali pradėti, sulaikyti, tęsti bei atšaukti roboto modeliavimą bet kuriuo modeliavimo etapu.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	5

Reikalavimo Nr.	4
Aprašymas	
Galimybė peržiūrėti bei redaguoti kodą sukurtą vizualia kalba imperatyvia programavimo kalba.	
Atitikimo kriterijus	
Vartotojas sukurtas vizualus programinis kodas yra transformuojamas į imperatyvų programinį kodą, kurį vartotojas gali peržiūrėti bei pakeisti pagal poreikį.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	4

3.2. Apribojimai reikalavimams

3.2.1. Reikalavimas sprendimui

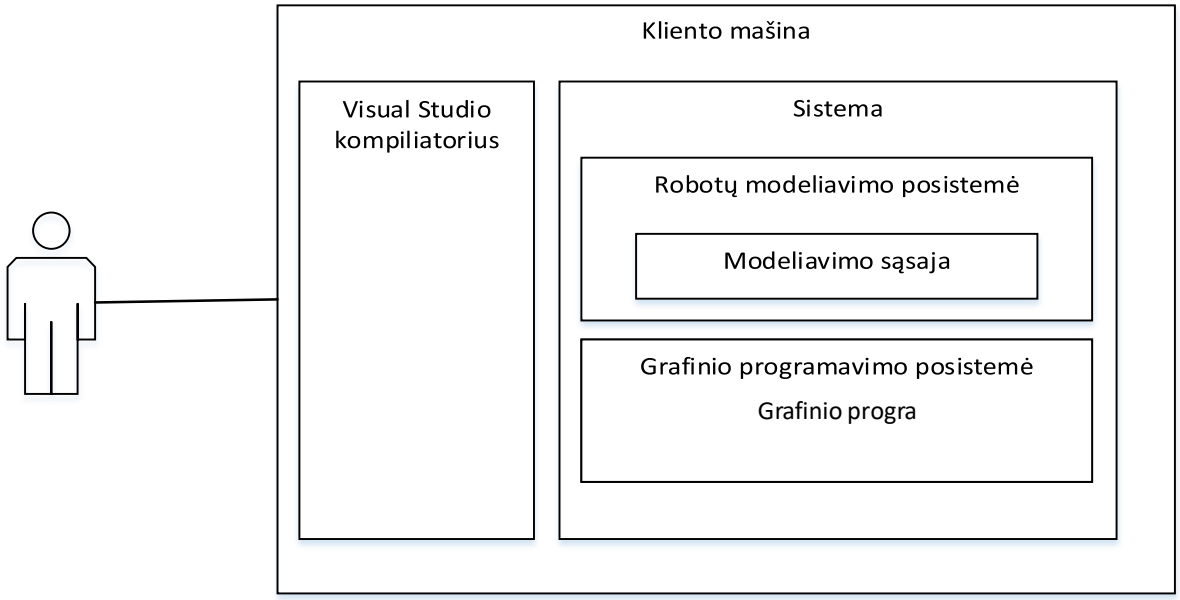
Reikalavimas:	Sistema turi veikti *Windows* operacinėje aplinkoje.
Aprašymas:	Kuriama sistema yra mokomoji priemonė skirta tiek namų aplinkai tiek mokymo įstaigoms. Lietuvoje populiariausia operacinė sistema tiek namuose, tiek mokymo įstaigose yra *Windows*. Dėl šios priežasties pirminė PĮ iteracija privalo veikti šioje aplinkoje.
Atlikimo kriterijus:	Sistema veikia *Windows 7*, *Windows 8*, *Windows 8.1* bei *Windows 10* aplinkose.

Reikalavimas:	Sistemai reikalingas gatavas 3D variklis.
Aprašymas:	3D variklio kūrimui reikalinga daug laiko ir pastangų. Siekiant sutaupyti lėšas, bei sumažinti laiko sąnaudas reikalingas gatavas 3D variklis veikiantis *Window*s aplinkoje.
Atlikimo kriterijus:	Pasirinktas variklis atitinkantis specifikacijai keliamus reikalavimus. Pirminei iteracijai naudojamas variklis – *Unity 5*.

Reikalavimas:	Vizualios kalbos universalumas.
Aprašymas:	Vizuali programavimo kalba turi būti universali. Šiam tikslui pasiekti ji turi būti konvertuojama į pasirinktos imperatyvios programavimo kalbos kodą.

Atlikimo kriterijus:	Vizuali programavimo kalba transformuojama į *C#* programinį kodą su galimybe pritaikyti transformavimui į kitas kalbas kaip *Java*.
Reikalavimas:	Modeliavimo posistemės našumas.
Aprašymas:	Modeliavimo posistemė turi būti pakankamai naši.
Atlikimo kriterijus:	Modeliavimo posistemė turi įvykdyti ne mažiau 30 roboto instrukcijų per sekundę.
Reikalavimas:	Transformacijos bei modeliavimo posistemių modulių pakartotinis panaudojimas.
Aprašymas:	Moduliai turi būti lengvai panaudojami kitose sistemose.
Atlikimo kriterijus:	Sistemos moduliai turi veikti kaip nepriklausomos posistemės kurias būtų galima pakartotinai panaudoti kituose projektuose.
Reikalavimas:	Atvira modeliavimo posistemės sąsaja.
Aprašymas:	Modeliavimo posistemė turi turėti atvirą sąsają kurią būtų galima naudoti sukūrus programinį kodą kitais būdais.
Atlikimo kriterijus:	Modeliavimo posistemė turi sąsają kuria galima pilnai išnaudoti visą virtualių robotų funkcionalumą nekuriant jų sistema.

3.2.2. Diegimo aplinka



3.2 pav. Diegimo aplinkos (išdėstymo) diagrama

3.2.3. Komunikuojančios sistemos

Kuriama sistemai reikalinga transformacija iš grafine kalba sukurto kodo į vykdomąjį kodą. Kadangi sistema transformuos kodą į C# programinį kodą reikalingas ir atitinkamas kompiliatorius. Pirmai iteracijai naudojamas *Unity* variklis. Šis variklis turi savo *.NET* analogą vadinamą *Mono*. Jis palaiko *.NET* asemblius iki 2.0 versijos. Dėl šios priežasties bus naudojamas kartu su *Visual Studio* pridedamas kompiliatorius.

3.2.4. Prieinama specializuota PĮ

Sistemai sukurti reikalingas grafinis variklis. Egzistuoja įvairių siūlomų sprendimų šiam keliamam reikalavimui išspręsti. Šiam reikalavimui išpildyti pasirinktas *Unity 5* variklis. Pasirinktas specializuota PĮ leidžia naudojant nemokamą programinę versiją ją naudoti komerciniams, asmeniniams tiek moksliniams tikslams.

3.2.5. Numatoma darbo vietos aplinka

Sistema yra kuriama asmeniniams kompiuteriams bei fokusuojama į dvi tikslines vartotojų auditorijas:

- Mokiniai – šių vartotojų darbo vieta yra kompiuterių klasės. Vienu kompiuteriu gali naudotis daugiau nei vienas mokinys.
- Namų vartotojai – tai tie vartotojai, kurie naudojasi programos versija skirta namams. Šių vartotojų darbo aplinka bus jų namai. Dažniausia aplinka – darbastaliai.

3.3. Reikalavimų analizės apibendrinimas

1. Atlikta funkcinių bei nefunkcinių reikalavimų analizė. Iš šių reikalavimų galime paruošti sistemos projektą. Taip pat funkciniai reikalavimai parodys atliktos sistemos kokybę, kadangi sistemos kokybę galima matuoti pagal patenkintus funkcinių bei nefunkcinius reikalavimus.
2. Taip pat buvo numatyta diegimo aplinka kurioje bus vykdoma programinė įranga. Žinodami diegimo aplinką galime daryti mažiau prielaidų projektuojant sistemą.

4. SISTEMOS PROJEKTAS

4.1. Sistemos pagrindimas ir esmės išdėstymas

Siekiama, jog „Vizualios robotų programavimo ir robotų sąveikos simuliacinės aplinkos“ architektūrą būtų lengvai išplečiama. Dėl šios priežasties architektūrą yra komponentinė t.y. posistemės nėra tarpusavyje kietai surištos, jas galima lengvai pakeisti. Todėl sistema yra skiriama į tris pagrindines posistemes:

1. vizualaus programavimo;
2. vizualaus kodo transformavimo;
3. robotų modeliavimo.

Šie komponentai veikia kaip atskiros paprogramės, dėl šios priežasties juos lengva pakeisti kitais, nerizikuojant pažeisti sistemos integralumą. Tokį architektūrinį pasirinkimą įtakojo, tai jog *C#* kompiliatorius kartu su *Visual Studio* paketu yra licencijuoti. Todėl negalima jo komplektuoti kartu su kuriama sistema. Ši problema išsprendžiama kompiliavimo posistemėje, kuri yra izoliuota paprogramė kuri naudojama iškviečiant kompiliavimo komandą.

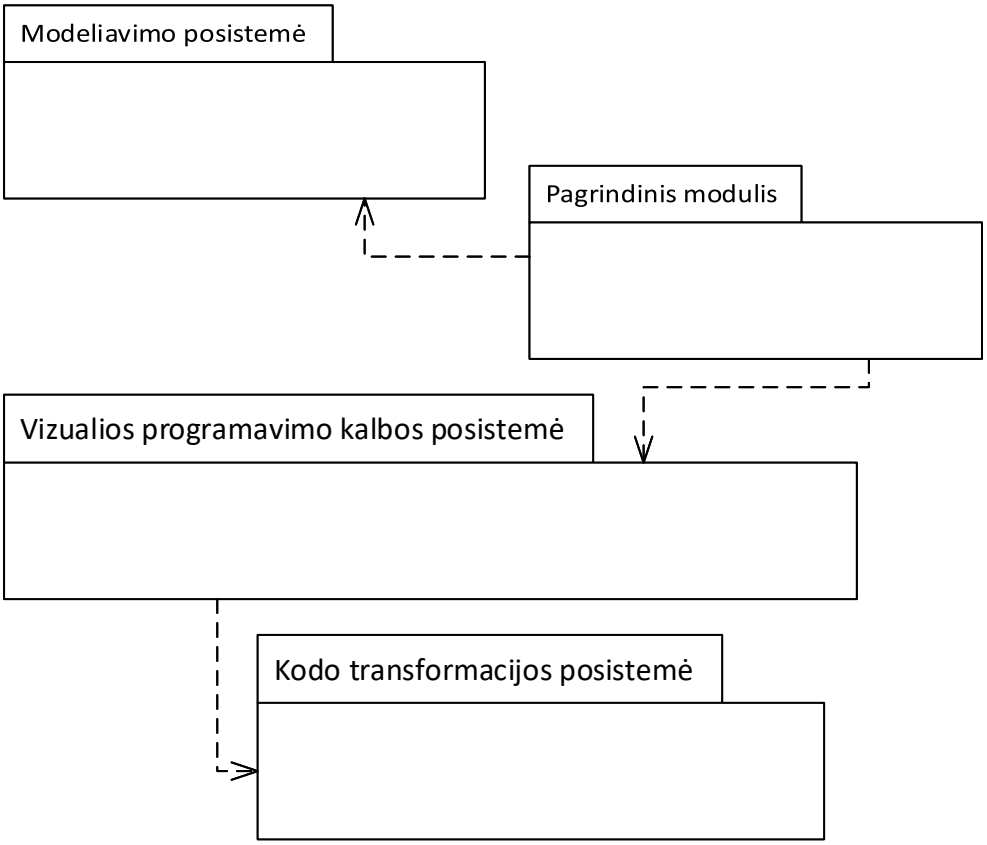
Abstraktaus sintaksės medžio struktūra kompiliavimo transformavimo posistemei parinkta siekiant padidinti išplečiamumą ateityje, kadangi tokis architektūrinis sprendimas leidžia vykdyti ir atvirkštinę kodo transformaciją.

4.2. Sistemos architektūra

Kuriama sistema turi keturias tarpusavyje sąveikaujančias posistemes:

- Vizuali robotų programavimo posistemė.
- Virtualių robotų modeliavimo posistemė.
- Mazgų transformavimo į kodą posistemė.

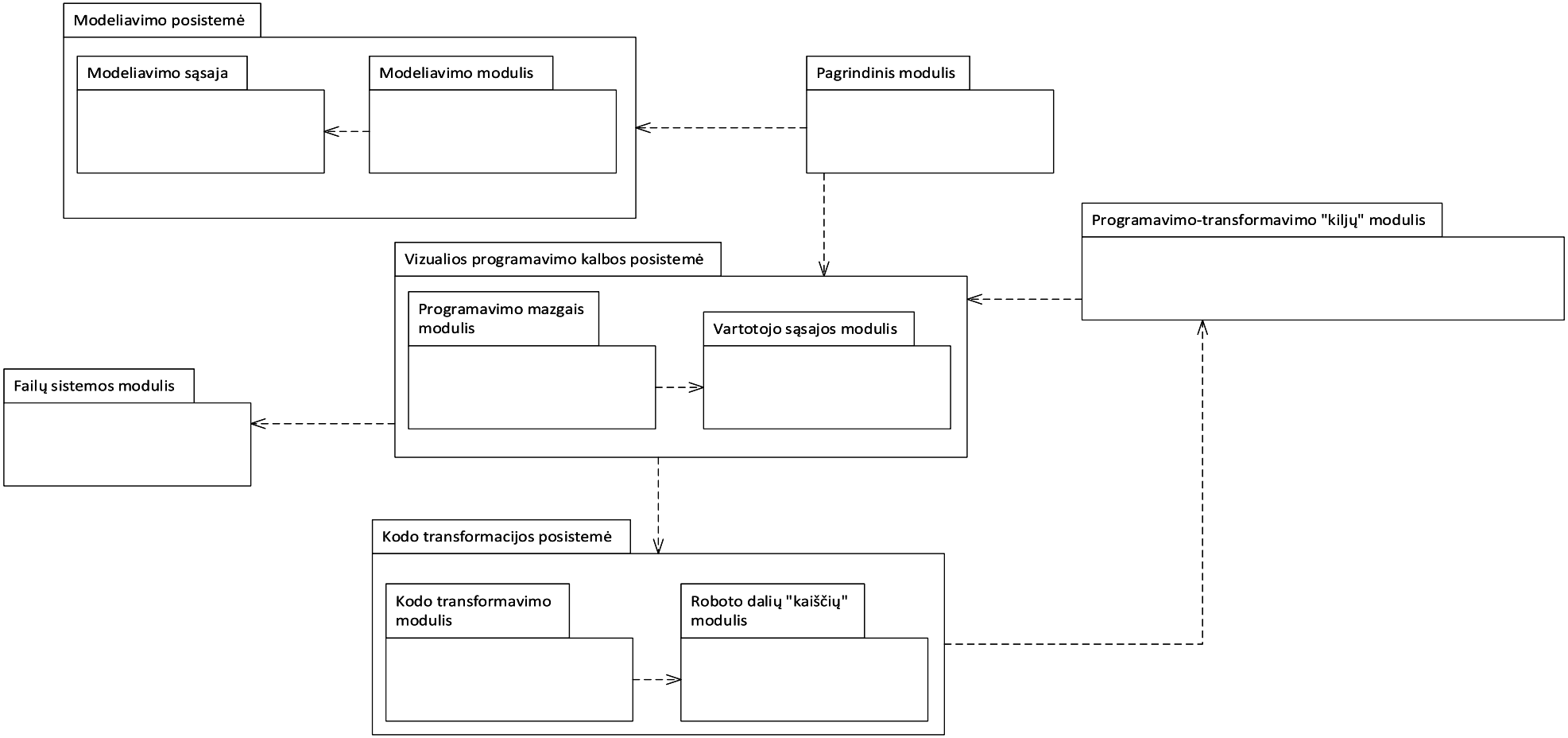
Komponentų sąveikos diagrama pateikiama 4.1 diagramoje.



4.1 pav. Pagrindinių komponentų paketų diagrama

4.3. Detalus projektas

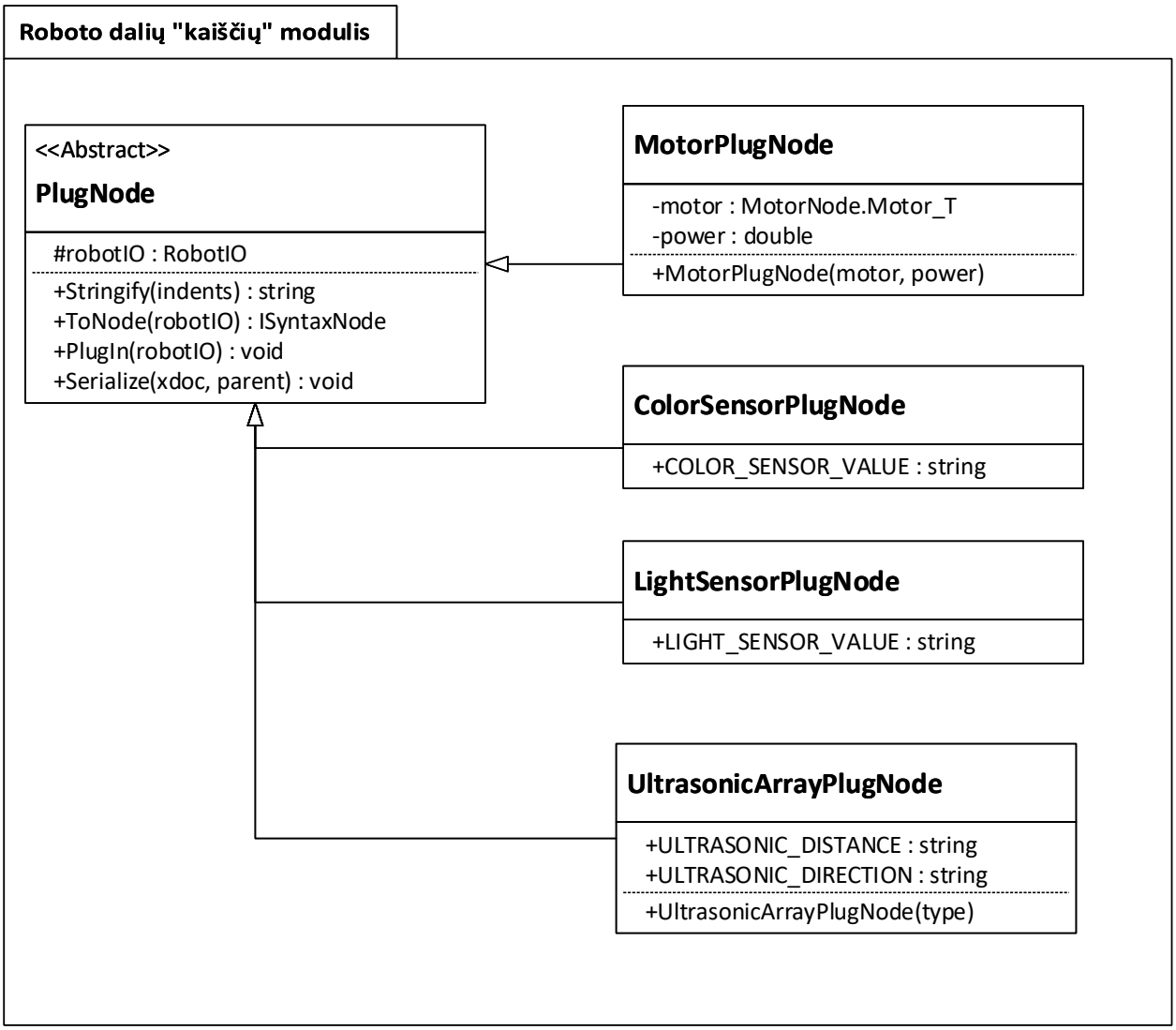
Detalizuota visos sistemos paketų diagrama pateikiama 4.2 paveiksle.



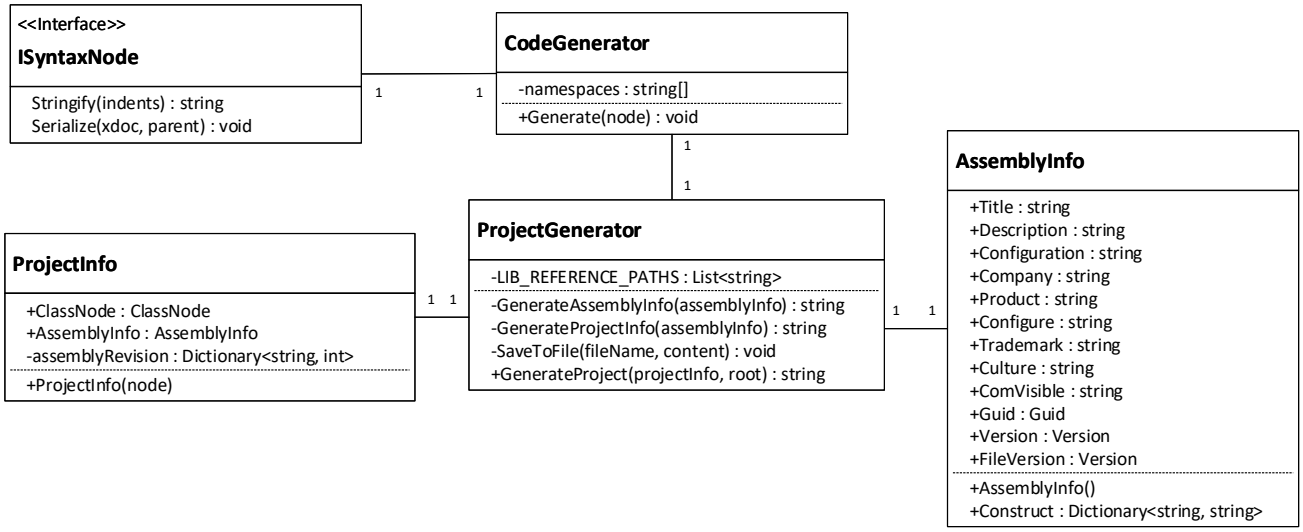
4.2 pav. Detalizuoti visos sistemos paketai.

Paketas **„Vizualios programavimo kalbos posistemė“** atsakingas už vizualios robotų programavimo kalbos sintaksės implementaciją ir vartotojo sąsajos atvaizdavimą ekrane. Programavimo-transformavimo „klijų“ modulis atsakingas už duomenų pateikimą transformavimo posistemei.

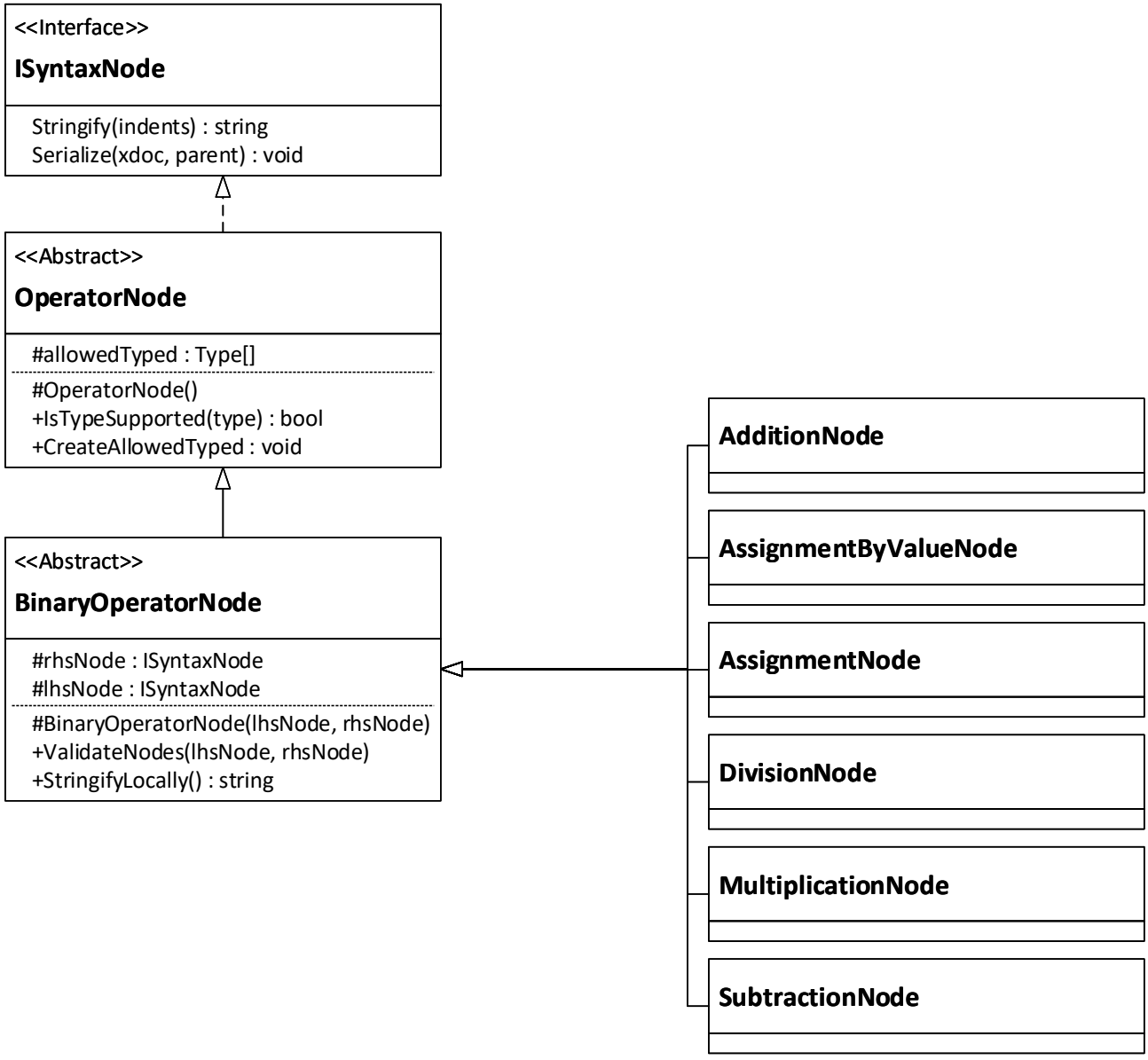
Paketas **„Kodo transformavimo posistemė“** atsakingas už vizualios robotų programavimo kalbos sintaksės transformavimą į abstraktų sintaksės medį ir šio medžio transformavimą į C# programavimo kalbą. Šios posistemės klasių tarpusavio sąveikų diagramos vaizdas pateikiamas 4.9 paveikslėlyje, o jos duomenų žodynas pateikiamas 4.1 lentelėje. Paveiksluose 4.4-4.8 matomas detalesnis 4.9 vaizdas. Diagrama buvo suskaldyta į penkias atskiras dalis siekiant aiškumo atspausdintame dokumente.



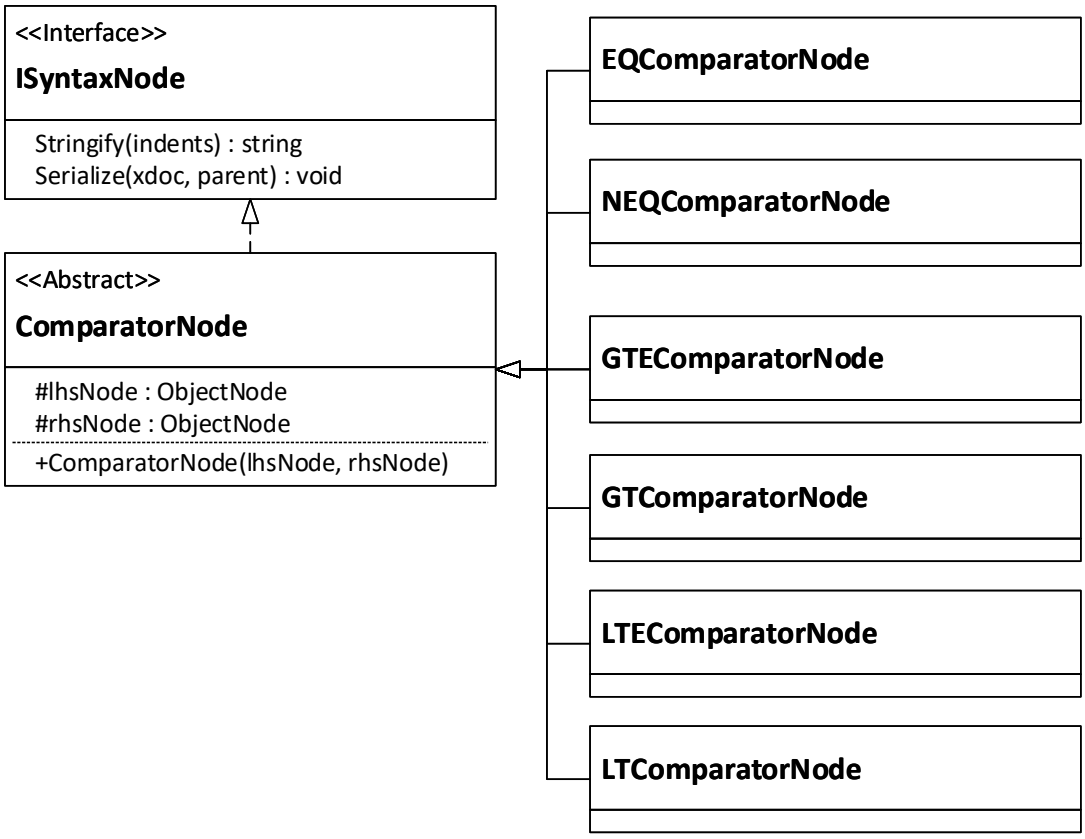
4.3 pav. Roboto dalių „kaiščių“ modulio klasių diagrama



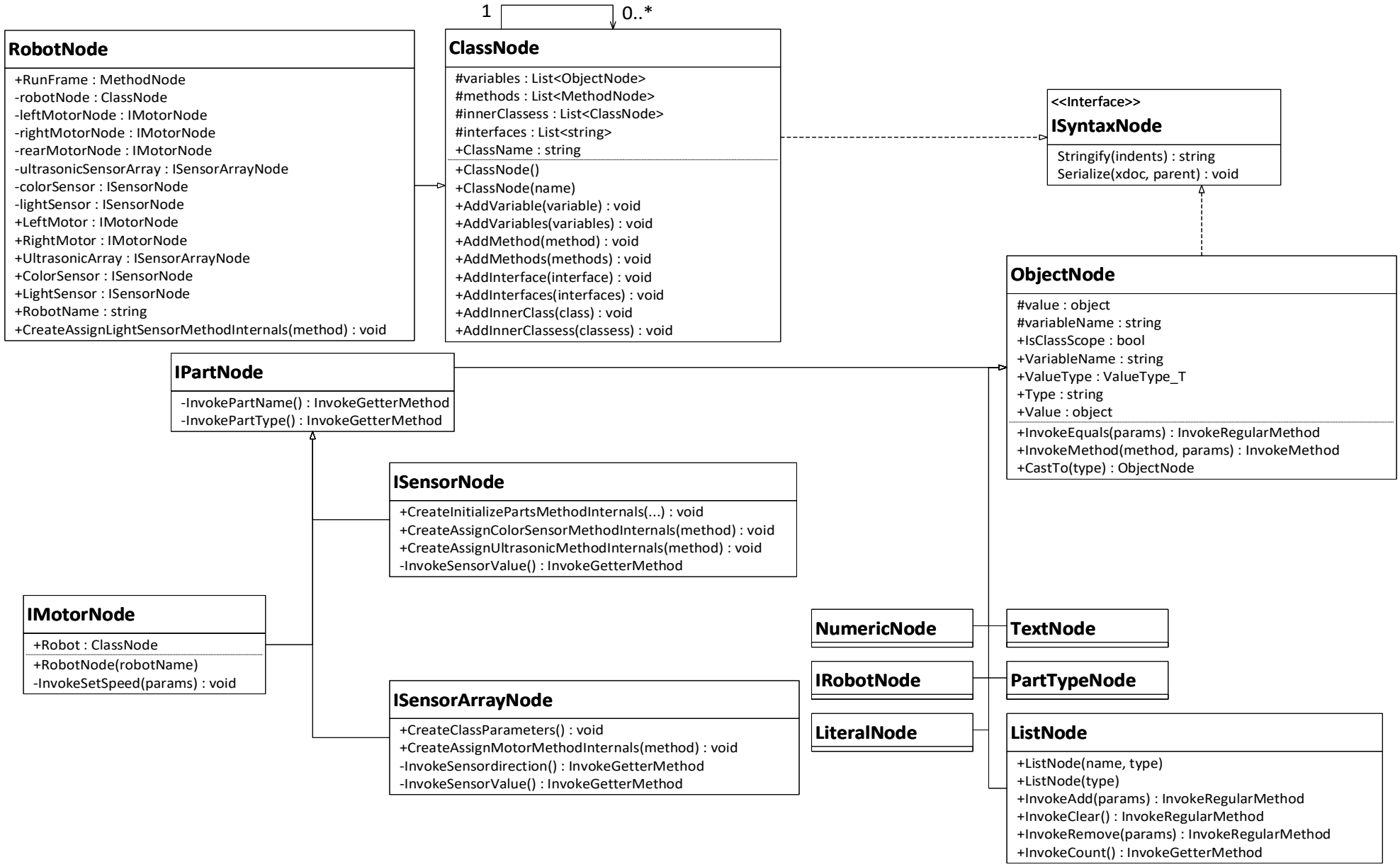
4.4 pav. Kodo transformacinės posistemės klasių diagrama (1)



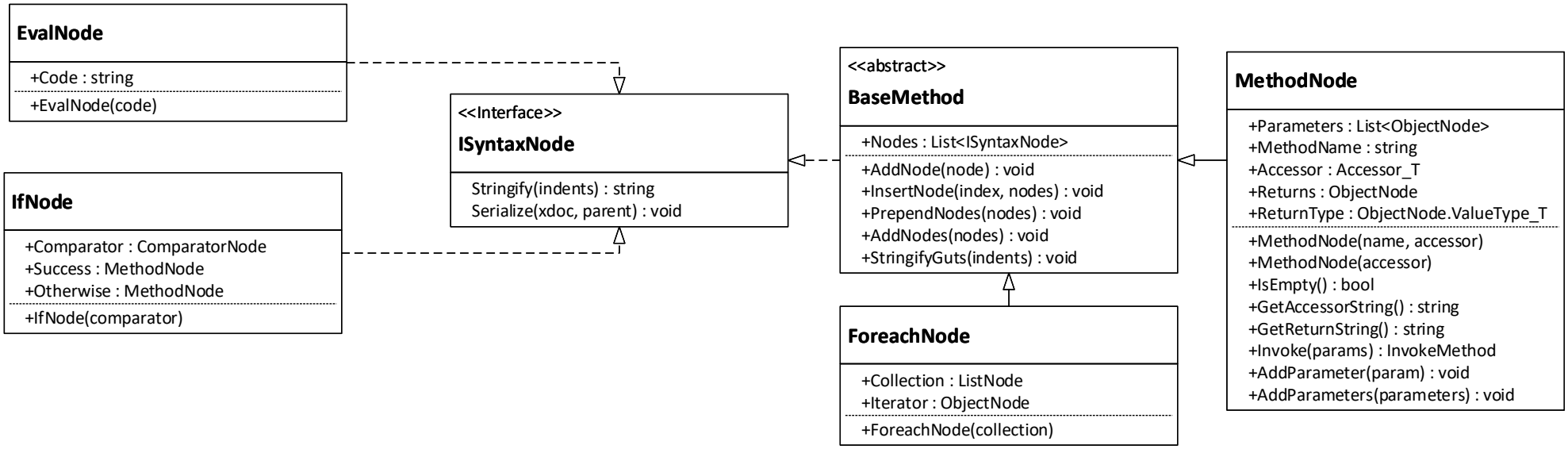
4.5 pav. Kodo transformacinės posistemės klasių diagrama (2)



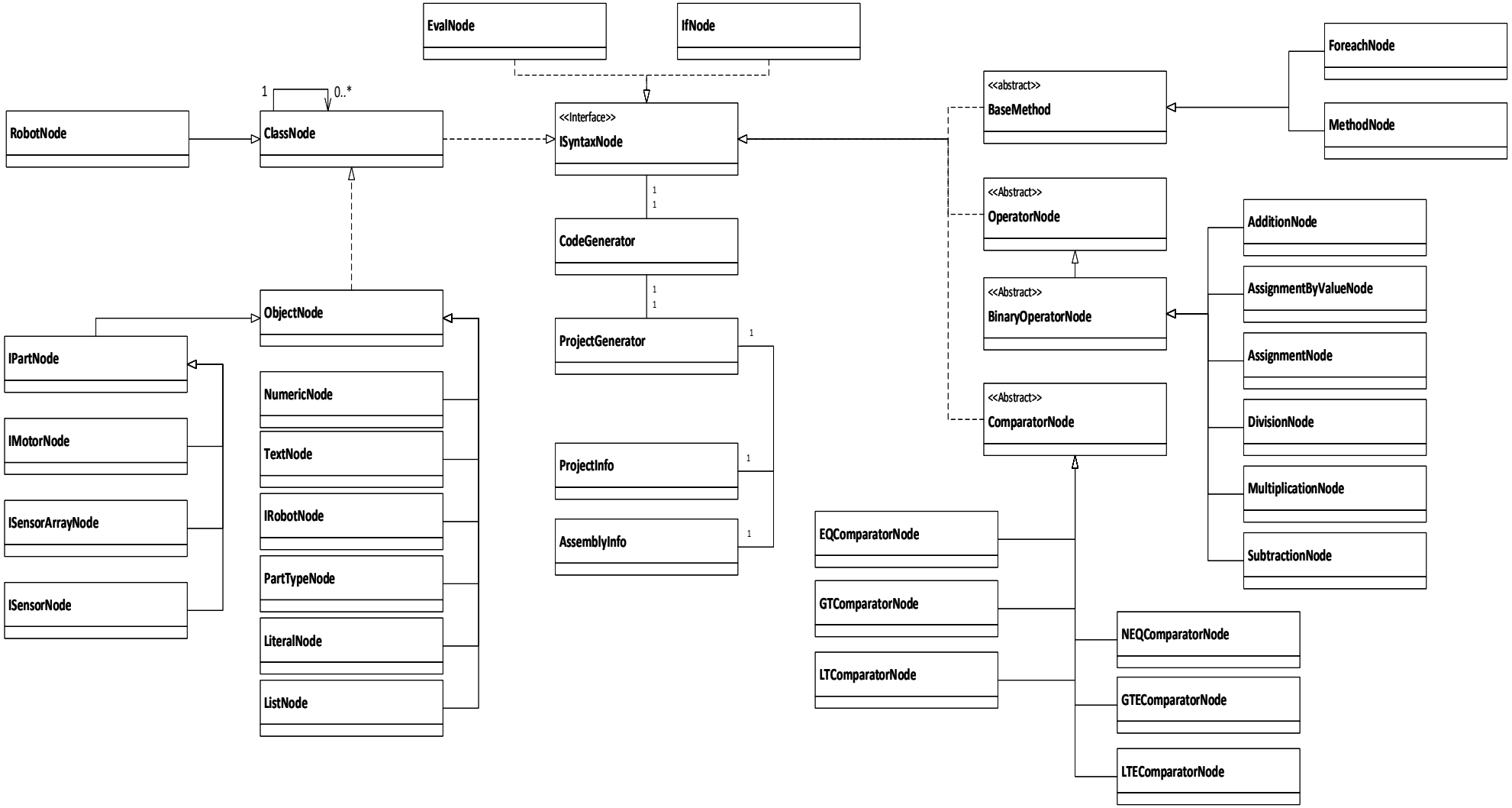
4.6 pav. Kodo transformacinės posistemės klasių diagrama (3)



4.7 pav. Kodo transformacinės posistemės klasių diagrama (4)



4.8 pav. Kodo transformacinės posistemės klasių diagrama (5)



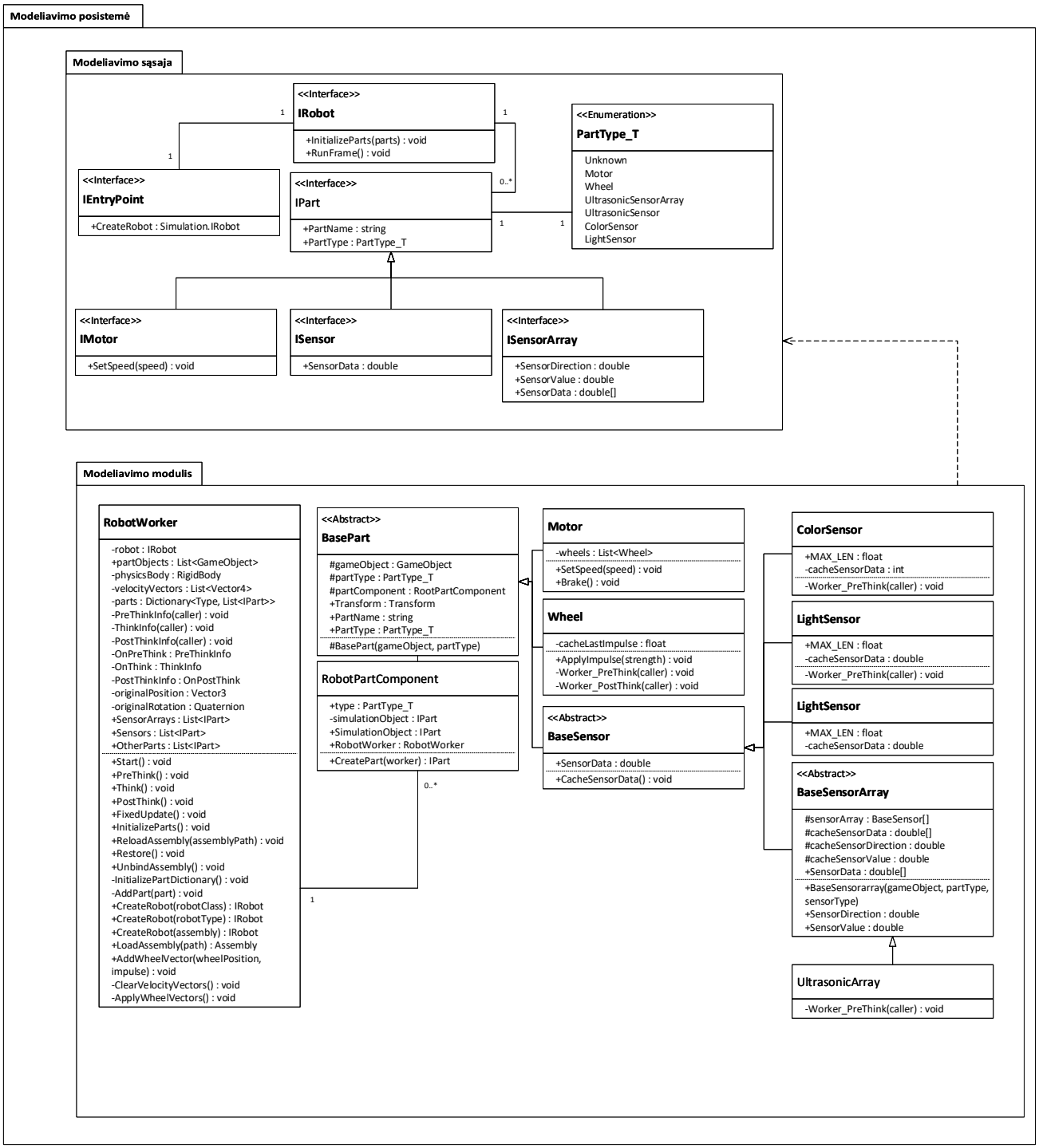
4.9 pav. Kodo transformacinės posistemės bendra supaprastinta klasių diagrama

4.1 lentelė. Kodo transformavimo posistemės duomenų žodynas

Pavadinimas	Aprašymas	Tipas
ISyntaxNode	Bazinė sintaksinio vieneto sąsaja kuri paveldi visi sintaksiniai vienetai.	Sąsaja
BaseMethod	Abstrakti klasė apibūdinanti metodus generuojančius sintaksinius vienetus.	Abstrakti klasė
AtrributeValue	Tai atributas kuris suteikiamas duomenų tipų klasėms tam kad būtų galima refleksijos metu dinamiškai sukurti reikiamo tipo objektą.	Atributas
ComparatorNode	Abstrakti klasė aprašanti palyginimo sintaksinius vienetus.	Abstrakti klasė
EQComparatorNode	Sintaksinis vienetas palyginantis ar du elementai yra lygūs.	Klasė
GTComparatorNode	Sintaksinis vienetas palyginantis ar vienas elementas yra didesnis už kitą.	Klasė
GTEComparatorNode	Sintaksinis vienetas palyginantis ar vienas elementas yra didesnis arba lygus už kitą.	Klasė
LTComparatorNode	Sintaksinis vienetas palyginantis ar vienas elementas yra mažesnis už kitą.	Klasė
LTEComparatorNode	Sintaksinis vienetas palyginantis ar vienas elementas yra mažesnis ar lygus kitam.	Klasė
NEQComparatorNode	Sintaksinis vienetas palyginantis ar vienas elementas nelygus kitam.	Klasė
InvokeMethod	Abstrakti klasė nurodanti aprašanti metodų iškvietimo sintaksinius vienetus.	Abstrakti klasė
InvokeRegularMethod	Sintaksinis elementas iškviečiantis standartinį objekto metodą.	Klasė
InvoekGetterMethod	Sintaksinis vienetas iškviečiantis *getter* tipo objekto metodą.	Klasė
InvokeSetterMethod	Sintaksinis vienetas iškviečiantis *setter* tipo objekto metodą.	Klasė
InvokeStaticRegularMethod	Sintaksinis vienetas iššviečiantis standartinį statinį metodą.	Klasė
InvokeStaticGetterMethod	Sintaksinis vienetas iškviečiantis *getter* tipo statinį metodą.	Klasė
IfNode	Sintaksinis vienetas aprašantis *if-else* sąlygos sakinį.	Klasė
ForeachNode	Sintaksinis vienetas aprašantis *foreach* ciklą.	Klasė
ClassNode	Sintaksinis vienetas aprašantis klasę.	Klasė
RobotNode	Sintaksinis vienetas aprašantis klasę su robotui specifiniais kintamaisiais bei metodais.	Klasė
EvalNode	Sintaksinis vienetas generuojantis nurodytą kodo fragmentą.	Klasė
MethodNode	Sintaksinis vienetas aprašantis metodą.	Klasė
ObjectNode	Sintaksinis vienetas aprašantis objektą. Ši klasė taip pat yra bazė visiems kintamųjų tipams.	Klasė
IMotorNode	Sintaksinis vienetas aprašantis *IMotor* kintamojo tipą.	Klasė
IPartNode	Sintaksinis vienetas aprašantis *IPart* kintamojo tipą.	Klasė
IRobotNode	Sintaksinis vienetas aprašantis *IRobot* kintamojo tipą.	Klasė
ISensorNode	Sintaksinis vienetas aprašantis *ISensor* kintamojo tipą.	Klasė
ISensorArrayNode	Sintaksinis vienetas aprašantis *ISensorArray* kintamojo tipą.	Klasė
ListNode	Sintaksinis vienetas aprašantis sąrašo kintamojo tipą.	Klasė
LiteralNode	Sintaksinis vienetas aprašantis tipą kurio reikšmei netaikoma tipo konversija.	Klasė
NumericNode	Sintaksinis vienetas aprašantis *double* kintamojo tipą.	Klasė
PartTypeNode	Sintaksinis vienetas aprašantis *PartType* kintamojo tipą.	Klasė

Pavadinimas	Aprašymas	Tipas
TextNode	Sintaksinis vienetas aprašantis *string* kintamojo tipą.	Klasė
OperatorNode	Bazinė klasė aprašantis operatoriaus tipo sintaksinius vienetus.	Abstrakti klasė
BinaryOperatorNode	Bazinė klasė aprašantis binarinių operatorių sintaksinius vienetus.	Abstrakti klasė
AdditionNode	Sintaksinis vienetas aprašantis binarinę sudėtį.	Klasė
AssignmentNode	Sintaksinis vienetas aprašantis priskyrimo operatorių.	Klasė
DivisionNode	Sintaksinis vienetas aprašantis binarinį dalybos operatorių.	Klasė
MultiplicationNode	Sintaksinis vienetas aprašantis binarinį daugybos operatorių.	Klasė
SubtractionNode	Sintaksinis vienetas aprašantis binarinį atimties operatorių.	Klasė
CodeGeneration	Klasė sugeneruojanti programinį kodą iš nurodyto sintaksės medžio.	Klasė
ProjectGeneration	Klasė sugeneruojanti papildomus projekto failus reikalingus kompiliavimo metu.	Klasė
ProjectInfo	Klasė aprašanti projekto informaciją.	Klasė
AssemblyInfo	Klasė aprašanti sukompiliuoto *asemblio* informaciją.	Klasė
Extensions	Klasių plėtinių rinkinys.	Klasė
Utilities	Pagalbinių metodų rinkinys.	Klasė
StringWriterWithEncoding	Praplečiamas *StringWriter* klasės funkcionalumas tam kad būtų palaikomas skirtingas teksto kodavimas.	Klasė
PlugNode	Abstrakti klasė aprašanti kaištį. Kaiščiai naudojami kaip laikini elementai kodo generavime kaip pakaitalas roboto dalims.	Abstrakti klasė
MotorPlugNode	Kaiščio elementas motorui.	Klasė
ColorSensorPlugNode	Kaiščio elementas spalvų davikliui.	Klasė
LightSensorPlugNode	Kaiščio elementas liuksmetrui.	Klasė
UltrasonicSensorArrayPlugNode	Kaiščio elementas ultragarso daviklių masyvui.	Klasė

Paketas „**Robotų modeliavimo posistemė**“ ir jo sąsaja atsakinga už robotų elgsenos, aprašytos vizualia robotų programavimo kalba, modeliavimą virtualioje erdvėje. Šiai posistemei taip pat priklauso bendrinė sąsaja, kuri sistemoje dinamiškai paleidžia vartotojo sukurtus robotų asemblius. Posistemės tarpusavio sąveikų diagramos vaizdas matomas 4.10 paveikslėlyje, duomenų žodynas pateikiamas 4.2 lentelėje.



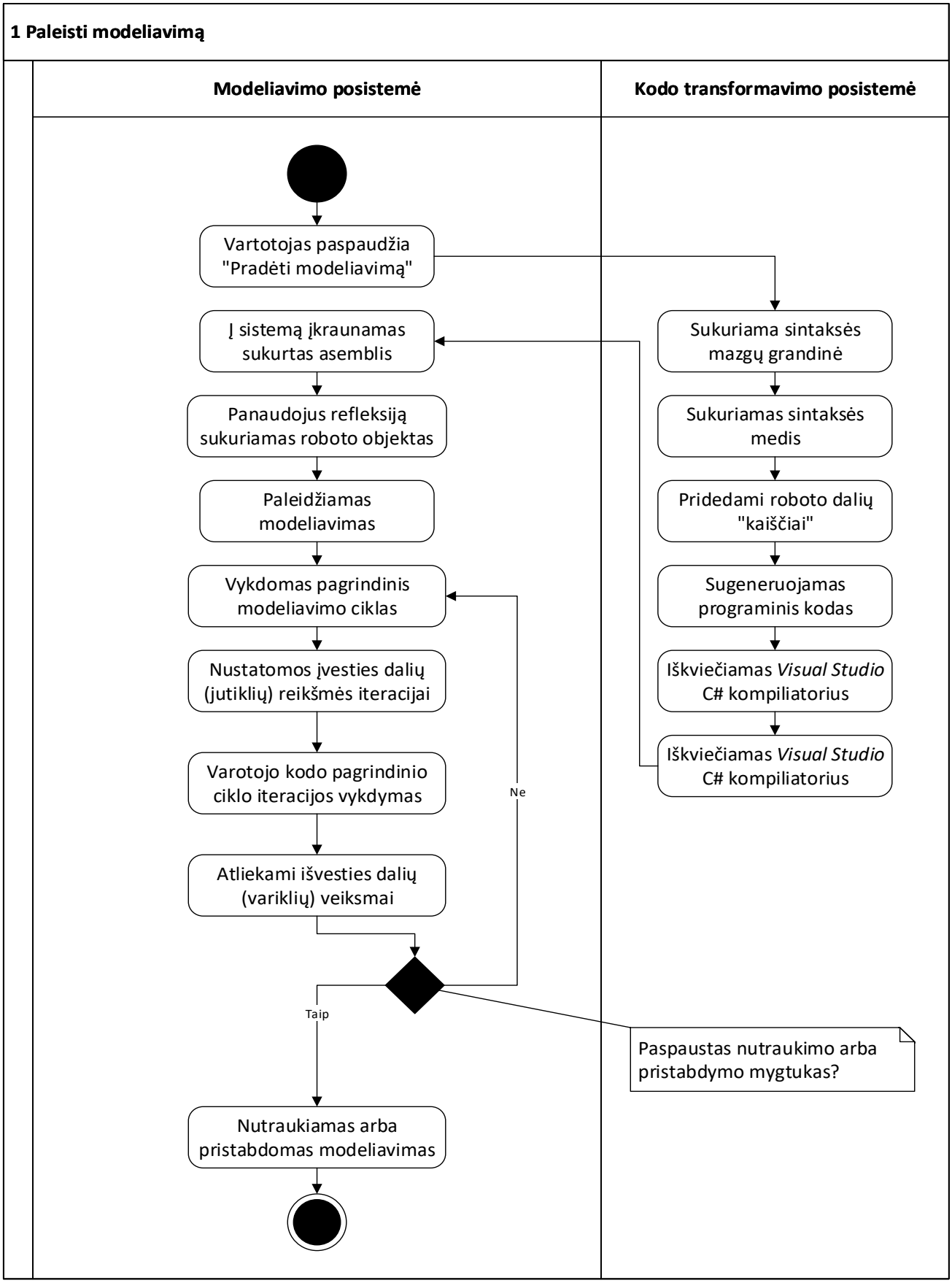
4.10 pav. Modeliavimo posistemės klasių diagrama

4.2 lentelė. Modeliavimo posistemės duomenų žodynas.

Pavadinimas	Aprašymas	Tipas
RobotWorker	Klasė atsakinga už dinaminį robotų *asemblio* įkrovimą į sistemą ir roboto modeliavimo vykdymą.	Klasė
RobotSpawn	Klasė nurodanti pradinį roboto padėtį.	Klasė
BasePart	Dalį aprašanti bazinė klasė.	Abstrakti klasė
Motor	Variklį aprašanti klasė.	Klasė
RobotPartComponent	Sąsają tarp *Unity* variklio objekto ir roboto dalių aprašanti klasė.	Klasė

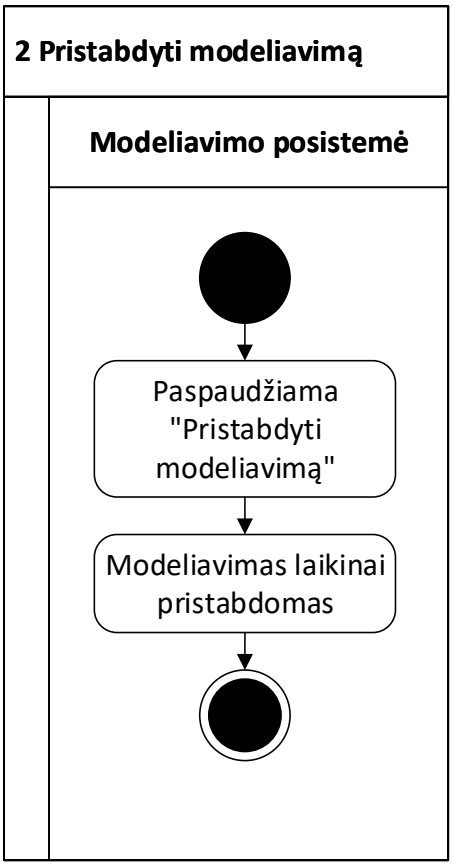
Pavadinimas	Aprašymas	Tipas
Wheel	Ratą aprašanti klasė.	Klasė
BaseSensor	Bazinį jutiklių veikimą aprašanti klasė.	Abstrakti Klasė
ColorSensor	Spalvų jutiklio veikimą aprašanti klasė.	Klasė
LightSensor	Šviesos jutiklio veikimą aprašanti klasė.	Klasė
UltrasonicSensor	Ultragarso jutiklio veikimą aprašanti klasė.	Klasė
BaseSensorArray	Bazinį jutiklių masyvų veikimą aprašanti klasė.	Abstrakti klasė
UltrasonicSensorArray	Ultragarso jutiklių masyvo veikimą aprašanti klasė.	Klasė
IRobot	Metodus, kuriuos turi įgyvendinti robotų klasės, nurodanti sąsaja.	Sąsaja
IEntryPoint	Sąsaja paveldi visos *RobotVL* sugeneruotos robotų programos. Sąsaja naudojama dinamiškai užkrauti robotų asemblius ir paleisti vykdymą.	Sąsaja
IPart	Visų tipų dalis aprašanti sąsaja.	Sąsaja
IMotor	Variklių tipo dalis aprašanti sąsaja.	Sąsaja
PartType_T	Sąsajos palaikomų elementų sąrašas.	Išvardijimas
ISensor	Jutiklių tipo dalis aprašanti sąsaja.	Sąsaja
ISensorArray	Jutiklių rinkinių tipo dalis aprašanti sąsaja.	Sąsaja

4.4. Sistemos elgsenos modelis



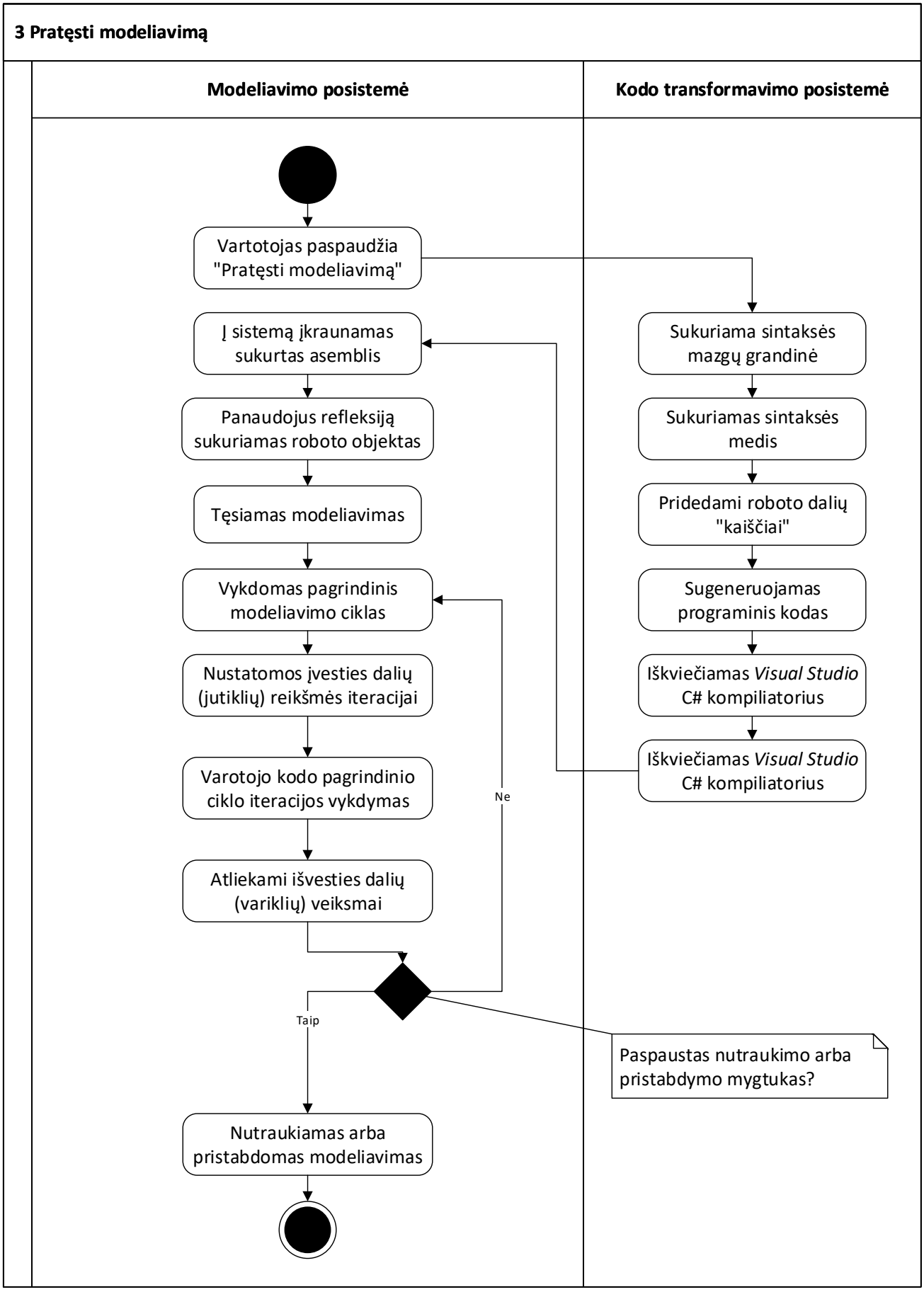
4.11 pav. Sistemos elgsenos pradėjus modeliavimą veiklos diagrama

Sistemos elgsena paspaudus „Pradėti modeliavimą“ pavaizduota 4.11 paveiksle.

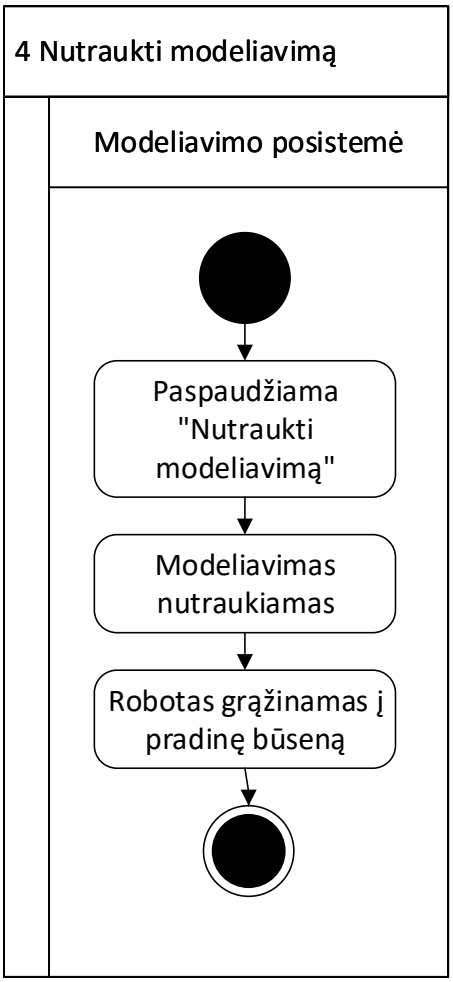


4.12 pav. Sistemos elgsenos nutraukus modeliavimą veiklos diagrama

Sistemos elgsena vartotojui paspaudus „Pristabdyti modeliavimą“ mygtuką pavaizduota 4.12 paveiksle. Sistemos būsena vartotojui paspaudus „Pratęsti modeliavimą“ pavaizduota 4.13 paveiksle.



4.13 pav. Sistemos elgsenos tęsiant modeliavimą veiklos diagrama



4.14 pav. Sistemos elgsenos nutraukus modeliavimą veiklos diagrama

Sistemos būsena vartotojui paspaudus nutraukti modeliavimą pavaizduota 4.14 paveiksle.

5. SISTEMOS REALIZACIJA

5.1. Sistemos funkcionalumo ir realizacijos aprašymas

5.1.1. Apie sistemą

Vizualaus robotų programavimo ir robotų sąveikos modeliavimo aplinkos sistema suteikia galimybę sukurti riboto funkcionalumo vizualų programinį kodą, kurio pagalba būtų galima valdyti robotą, tam jog šis atliktų jam paskirtas užduotis. Pateikiama programavimo sąsaja (*API*) kuri leidžia vartotojui lengvai prijungti savo sukurtus robotus į sukurtą modeliavimo aplinką. Vartotojui suteikiama galimybė aprašyti roboto elgseną naudojantis roboto valdymui pritaikyta vizualia programavimo kalba.

Sukurta vizuali programavimo kalba leidžia atlikti pagrindinius programavimo veiksmus ir realizuoti esamus algoritmus. Šiuos algoritmus galima išbandyti virtualioje aplinkoje. Vartotojo suprogramuota logika (algoritmai) atvaizduojami roboto judėjime virtualioje aplinkoje. Robotas (bei vartotojas) su aplinka sąveikauja naudojant robotų komponentus. Šie komponentai yra standartizuoti, todėl roboto funkcijos bus klasikinės kaip judėjimas, atstumo fiksavimas iki kliūties, spalvos atpažinimas, aplinkos apšvietos matavimas ir kita.

5.1.2. Pagrindinės funkcijos

Vizualios programavimo sąsajos pagrindinės funkcijos:

- Sukurti mazgą, galimi mazgų tipai:
    - motoras;
    - ultragarso sensorių masyvas;
    - metodas;
    - kintamasis;
    - sąlyga;
    - aritmetinis veiksmas.
- Keisti mazgų reikšmes.
- Ištrinti mazgus.
- Peržiūrėti mazgo panaudojimo pavyzdį.

Robotų modeliavimo posistemės pagrindinės funkcijos:

- Pradėti modeliavimą.
- Pristabdyti modeliavimą.
- Pratęsti modeliavimą.
- Nutraukti modeliavimą.
- Dinaminis kodo įkrovimas bei atnaujinimas.

5.2. Kodo transformavimo sąsajos realizacija

Vizualaus kodo transformavimo į imperatyvinį programinį kodą posistemė yra realizuojama trimis moduliais:

1. Kodo transformavimo modulis.
2. Roboto dalių „kaiščių“ modulis.
3. Programavimo-transformavimo klijų modulis.

Kodo transformavimo modulis yra skirtas kodo generavimui. Šis modulis iš nurodytos įvesties sugeneruoja *C#* programinį kodą bei jį sukompiliuoja panaudojus kartu su *Visual Studio* pridedamą šios kalbos kompiliatorių.

Kiekvienas sintaksės vienetas yra paveldimas iš *ISyntaxNode* sąsajos. Ši sąsaja nurodo du metodus kuriuos privalo įgyvendinti kiekvienas sintaksės elementas:

1. *Stringify* – metodas kuris iškviečiamas tam kad sugeneruoti sintaksinio vieneto kodo fragmentą.
2. *Serialize* – metodas kuris grąžina šio elemento *XML* reprezentaciją.

Šiame modulyje yra įgyvendinamos šios *C#* kalbos struktūros:

1. Palyginimo operatoriai.
2. *ForEach* ciklas.
3. Binariniai operatoriai skirti aritmetinėms operacijoms.
4. Prilyginimo sakinys.
5. Klasės bei vidinės klasės.
6. Statiniai bei objekto parametrizuoti metodai.
7. *Getter*‘iai.
8. Sąlygos sakiniai.
9. Įvairūs kintamųjų tipai.
10. Tipų keitimą (*type casting*).

Kiekvienas kintamasis yra laikomas objektu. Dėl šios priežasties visi kintamieji paveldi *ObjectNode* klasę. Siekiant palengvinti kodo palaikymą, bei įvesti statinę analizę kompiliavimo metu (kai naudojama refleksija) visi tipai turi atributą nurodantį tipą.

Siekiant paprastumo vartotojui visi skaitiniai kintamieji yra saugomi dvigubu slankaus kablelio skaičiumi (*double*). Dėl šios priežasties ne tik sumažinama kodo apimtis kadangi nereikalingas tipų keitimas bei neatsiranda bėdų dėl konvertavimo tarp tipų.

Kaiščių modulis skirtas sąsajai tarp programinio kodo bei roboto jutiklių, motorų bei kitų dalių. Šie kaiščiai kodui perduoda arba nustato dalių reikšmes.

Programavimo-transformavimo „klijų“ modulis yra skirtas konvertuoti vizualia programavimo kalba sukurtą kodą į tokią formą kurią priimtų transformavimo modulis. Iš pradžių sugeneruojama mazgų grandinė, ši grandinė transformuojama į sintaksinį medį. Medyje roboto dalių mazgams priskiriami kaiščiai. Sukonstruotas medis siunčiamas į transformavimo modulį kur sugeneruojamas programinis kodas.

5.3. Modeliavimo posistemės realizacija

Modeliavimo posistemė susideda iš dalių:

1. Modeliavimo sąsajos modulio.
2. Modeliavimo sąveikos modulio.

Modeliavimo sąsajos modulis nurodo sąsają kuria robotai gali komunikuoti su modeliavimo sąveikos moduliu. Ši sąsaja yra būtina, kadangi visi robotai yra įkraunami į sistemą dinamiškai t.y. robotų *asembliai* yra įkraunami panaudojant refleksiją. Modeliavimo sąsaja susideda iš dviejų pagrindinių sąsajų:

1. *IEntryPoint* – tai yra pagrindinė sąsaja kurią turi paveldėti kiekviena programa kuri bus dinamiškai prijungta prie sistemos. Tai yra roboto programos įėjimo taškas (*main*) nurodantis jog kiekviena programa turi sukurti roboto elementą.
2. *IRobot* – tai kiekvieno roboto kuris naudojamas sistemoje sąsaja. Sąsajoje nurodomi metodai interakcijai su sistema. Sistema robotui perduoda prijungtų dalių sąrašą roboto sukūrimo metu bei iškviečia roboto pagrindinį ciklą 30 iteracijų per sekundę dažniu.

Kitos modeliavimo sąsajos yra susijusios su dalimis, tai yra sąsaja jutikliams, jutiklių masyvams bei motorams. Šie nurodo kokias įvestis gauna robotas iš pasaulio bei kokias išvestis gali robotas suteikti modeliavimo sistemai.

Modeliavimo sąveikos modulis atlieka robotų modeliavimą virtualioje aplinkoje. Pagrindinė robotų modeliavimo sąveikos komponentė yra *RobotWorker* klasė. Ši klasė yra atsakinga už pagrindinius su robotu susijusius veiksmus. Tai yra:

1. *Asemblių* dinaminis įkėlimas į sistemą.
2. Roboto sukūrimas refleksijos būdu.
3. Jutiklių informacijos atnaujinimas.
4. Motorų valdymas pagal gaunamą įvestį iš roboto programos.
5. Roboto pagrindinio ciklo vykdymą.
6. Fizikinę interakciją ir kita...

Taip pat šis modulis įgyvendina visų daviklių (bei jų masyvų) funkcionalumą, bei daviklių reikšmių imitavimą pagal dizainerio kuriančio virtualią aplinką poreikius. Pavyzdžiui galima pridėti zoną imituojančią šešėlį, kitu atveju tam reikalingas *raycast* atsižvelgiant į visų šviesos šaltinių skleidžiamą šviesą, tai yra labai brangi operacija laiko atžvilgiu.

5.4. Testavimo modelis

5.4.1. Testavimo tikslai ir objektai

Svarbiausias testavimo tikslas yra pateikti įtikinamus įrodomus, jog sukurta programinė įranga veikia tinkamai (be klaidų) bei atitinka keliamus kokybės bei funkcionalumo reikalavimus. Programinė įranga be defektų – neegzistuoja, tačiau galima pateikti ją su kiek įmanoma mažesniu defektų kiekiu.

Šiam tikslui pasiekti yra sukuriamas testavimo planas kuriame išskiriami pagrindiniai testavimo objektai. Šie objektai yra vartotojo sąsaja, atskirti programinės įrangos moduliai (vienetai) bei skirtingų sistemos komponentų bendravimas tarpusavyje. Taip pat, šiuo testavimu siekiama užtikrinti jog sukurta programinė įranga atitinka sistemos funkcionalumui keliamus reikalavimus.

5.4.2. Pagrindiniai apribojimai

Šiame skyrelyje aprašome organizacinius, techninius ir programinius testavimo apribojimus.

1. Programinės įrangos kūrėjai neturi reikalingų resursų tam jog būtų atliekamas nuoseklus programinės įrangos testavimas su įvairia technine įranga. Todėl bus daroma prielaida – veikiant mažiau resursų turinčiuose įrenginiuose veiks ir daugiau resursų turinčiuose.
2. Vizualioje robotų programavimo sistemoje leidžiama sukurti iki 5000 skirtingų mazgų.

5.4.3. Testavimo procedūra

5.4.3.1. Vienetų testavimas

Vienetų testavimo etape bus tikrinami atskirų tiek vizualaus robotų programavimo, tiek robotų sąveikos simuliacinės aplinkos ir šias atskiras sistemas apjungianti sistemos modulio dalis. Šiam testavimui bus taikomas baltos dėžės testavimo metodas. Tai reiškia, kad testavimo metu testuotojas pilnai žino programos architektūrą bei gali ja naudotis. Todėl testuotojas gali lengvai susikurti duomenų rinkinį reikalingą paduoti į tam tikrą sistemos modulio dalį, siekiant aptikti defektus testuojamoje programinėje įrangoje.

Šie testai bus automatizuojami paduodant testinius duomenis ir tikrinant gautus rezultatus ar šie sutampa su tikimomis reikšmėmis. Testavimo rezultatai bus kaupiami. Šio testavimo metu užtikrinsime, jog pakeitus vieną sistemos modulio dalį nesugadinome visos sistemos veikimo.

Testai kuriami programuotojų programos kūrimo metu, pasitelkiant *Test Driven Development* metodologiją. Bent vienam testui nepraėjus testavimo tai skaitoma kaip atsiradusi sistemos klaida ir ji turi būti pašalinta prieš pradedant integracinį testą.

5.4.3.2. Integravimo testavimas

Integravimo testavimas atliekamas tarp skirtingų sistemos komponentų tam jog būtų užtikrintas taisyklingas komponentų tarpusavio sąveikavimas.

Testavimo metu parengiami testai palaipsniui apjungiantys sistemos komponentus. Integraciniam testavimui bus sukuriami fiktyvūs objektai imituotuojantys dar neprijungtų komponentų sąsajas. Esant sėkmingam testui prijungiamas naujas komponentas. Šis procesas tęsiamas iki kol visi sistemos komponentai yra suintegruoti į sistemą. Integracinis testavimas yra sėkmingas jeigu atitinka testų scenarijų reikalavimus. Integraciniu testavimu siekiama rasti defektus atsirandančius dėl komponentų tarpusavio sąveikavimo.

5.4.3.3. Priėmimo testavimas

Priėmimo testavimo etapo metu pagrindinis tikslas patikrinti ar sistema atitinka jos specifikaciją. Šiam testavimui bus taikomas juodos dėžės testavimo metodas, todėl žinoma kokius duomenis reikia paduoti ir kokių reikia tikėtis rezultatų. Visą tai žinome iš duotos specifikacijos. Šis testavimas prasideda tik kai turime jau visą programą su visais jos moduliais. Šiame testavime tikrinama kaip programa atitinka specifikacijai, todėl šis testavimas visą savo dėmesį atiduoda programos funkcionalumui.

Priėmimo testavimas atliekamas dviem etapais. Pirmuoju testavimas atliekamas be vartotojo, juo metu patikrinami visi vartotojo sąsajos elementai ar jų atliekamos funkcijos pagal duotą specifikaciją. Antruoju etapu testavimas atliekamas su vartotoju leidžiant jam daryti ką jis nori, tokiu būdų stengiamasi surasti nenumatytas klaidas.

5.4.4. Testavimo resursai ir jų paskirstymas

5.4.4.1. Resursai

Programiniai testavimo resursai:

- Vizuali robotų programavimo sąsajos sistema.
- Robotų sąveikos simuliacinės aplinkos sistema.
- Vizualios programavimo ir simuliacinės robotų aplinkos apjungimo sistema.
- Techniniai resursai:
- Specialių techninių resursų programa nėra.
- Žmogiškieji resursai:
- Programuotojas / testuotojas.

Pagrindinis testavimo resursas – vizuali robotų programavimo ir robotų sąveikos modeliavimo aplinkos sistema. Programinę įrangą sudaro 3 pagrindiniai sistemos komponentai. Jie testuojami atskirai ir juos apjungiant. Šie komponentai yra:

- vizualaus robotų programavimo posistemė,
- robotų sąveikos modeliavimo posistemė
- vizualaus kodo transformavimo posistemė.

Mažesni komponentai bei moduliai yra skirti posistemių tarpusavio komunikacijai bei duomenų struktūrų suvienodinimui.

5.4.4.2. Personalas

Kokybės užtikrinimo personalas testuoja sukurtos sistemos:

- atskirus modulius,
- sąveiką tarp atskirų modulių,
- sistemos funkcionalumą.

5.4.4.3. Testavimo aplinka

Programinės įrangos vienetų ir integracijos testavimas bus atliekamas naudojantis *Visual Studio 2015*, *Unity 5.4* įrankiais.

Reikalavimai aplinkai yra šie:

Operacinė sistema – *Windows 8, Windows 8.1* ir *Windows 10*.

5.4.5. Testavimo rezultatų kaupimas

Testų aprašymai ir rezultatai bus saugomi *Microsoft Excel* faile sukurtoje lentelėje. Lentelę sudarys stulpeliai: Testo numeris, Testo pavadinimas, Įėjimo duomenys, Išėjimo duomenys, Pastabos.

5.1 lentelė. Testavimo rezultatų kaupimo lentelės struktūra

Testo numeris	Testo pavadinimas	Įėjimo duomenys	Išėjimo duomenys	Pastabos
...	...	...	...	...

5.5. Testavimo rezultatai

Naudojant *test driven development* sumažintas kodo klaidų skaičius dar programos kūrimo ciklo metu. Vienetų testai padeda aptikti klaidas iš kart joms įsivėlus, tai leido sumažinti kodo defektų skaičių bei padidino programos kokybę. Programiniam kodui atlikti pakeitimai įjungiami į pačią sistemą tik tada kai visi vienetų testai yra sėkmingi.

6. EKSPERIMENTINIS ALGORITMŲ TYRIMAS

6.1. Eksperimento planas bei eiga

Eksperimento metu buvo siekiama išanalizuoti kelio planavimo algoritmus norint palyginti jų greitaveikas bei parinktus kelius. Šiam eksperimentui buvo įgyvendinti 3 parinkti algoritmai. Šie algoritmai buvo pasirinkti pagal tai ar juos galima pritaikyti realiame pasaulyje, tinkamumą kuriamai sistemai bei jų tinkamumą įvairioms situacijoms. Palyginimui pasirenkamos algoritmų versijos be optimizacijų.

Eksperimentui buvo parinkti šie algoritmai:

1. *Dijkstros* algoritmas.
2. *A** algoritmas.
3. *Uniform-cost search (UCS)* algoritmas.

Skyriuje 2.4.3 minėtas *D** algoritmas nepateko į eksperimento metu tiriamus algoritmus kadangi šis algoritmas yra pririštas prie tinklelio ir turi maksimaliai 6 laisvės laipsnius kuriais gali judėti bei turi nuolatos žinoti kiekvieno mazgo esančio grafe būseną.

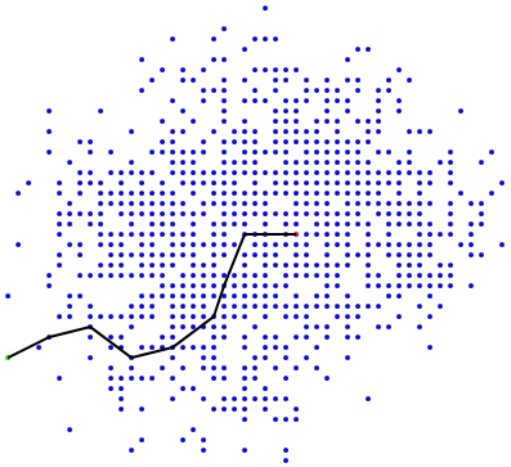
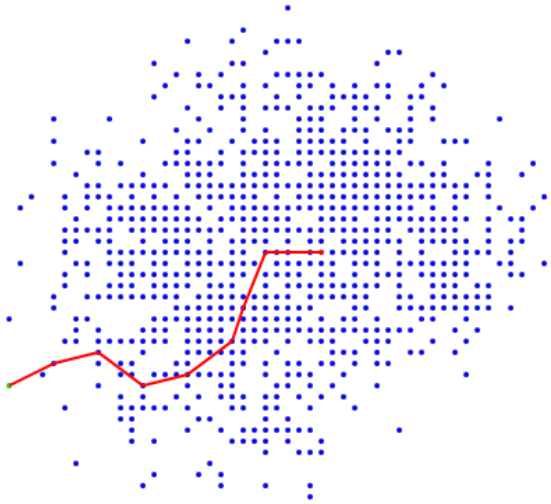
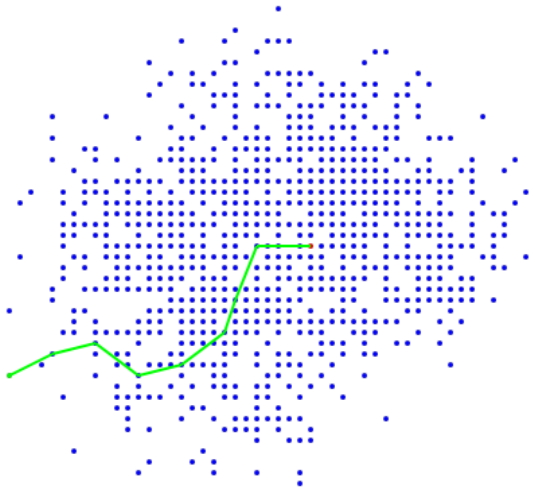
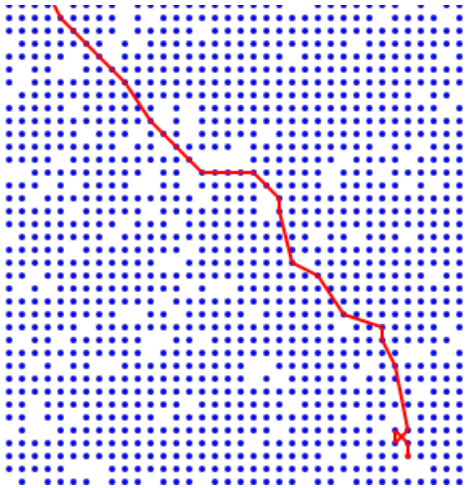
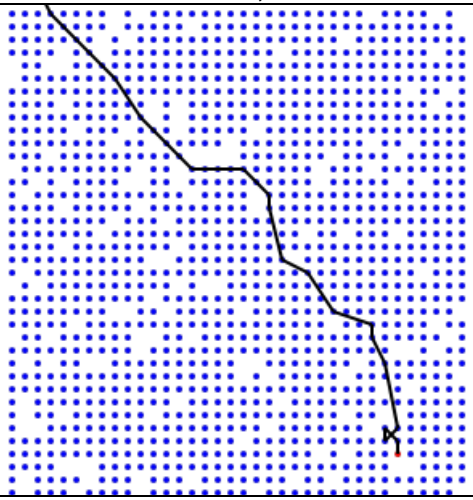
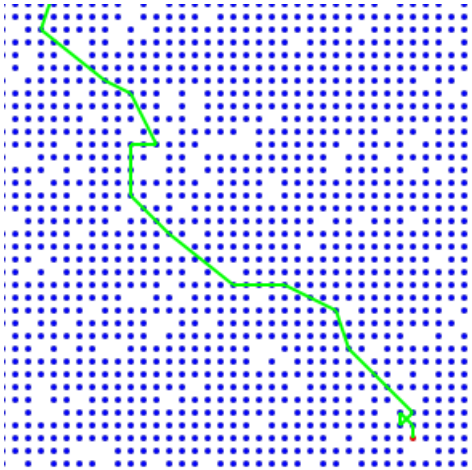
Algoritmų tyrimui buvo generuojami atsitiktiniai grafai su nurodytu viršūnių skaičiumi. Kiekviena viršūnė gali jungtis su mazgais ne tolesniais nei 5 vienetų atstumu. Tokia eksperimentinio grafo konfigūracija leidžia turėti nevientiso sudėtingumo grafus. Dėl šios priežasties, esant nevienodam grafo atstumų pasiskirstymui algoritmai kurie galime rasti grafikus kurie nėra trumpiausi, tačiau yra optimalesni greičio atžvilgiu.

Algoritmai buvo vertinami pagal šiuos kriterijus:

1. Ar buvo rastas trumpiausias kelias.
2. Laikas per kurį buvo rastas kelias.
3. Atminties sąnaudos kelio paieškos metu.

6.2. Eksperimento rezultatai

Eksperimento metu nustatėme jog Dijkstros, bei A* algoritmai visada randa trumpiausią kelią. Tačiau *Uniform-cost search* (UCS) ne visada randa globalų trumpiausią kelią, o randa lokalų minimumą atstumo atžvilgiu. Rezultatų pavyzdžiai matomi žemiau pateiktame 6.1 lentelėje.

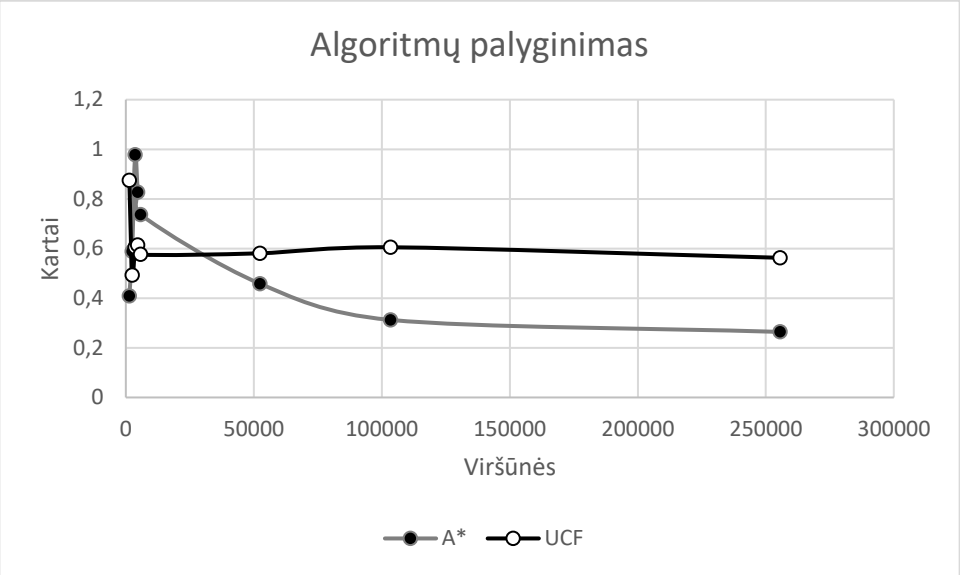
	Dijkstros algoritmas	**A***	***Uniform-cost search* (UCS)**
753 viršūnės / 1258 briaunos			
Ilgiai	≈ 36,3	≈ 36,3	≈ 36,3
5795 viršūnės / 12391 briaunos			
Ilgiai	≈ 80	≈ 80	≈ 84

6.1 lentelė. Algoritmų rastų kelių palyginimai.

Atlikti greitaveikos rezultatai pateikiami 6.2 lentelėje bei 6.1 paveiksle. Šie rezultatai yra pateikti normalizuota forma, kur bazė yra Dijkstros algoritmo rezultatai. *A** algoritmo euristikai buvo naudojamas atstumas tarp grafo viršūnes kurioje esame bei išėjimo viršūnės.

6.2 lentelė. Normalizuota rezultatų lentelė.

VIRŠŪNĖS	1352	2469	3608	4665	5758	52432	103336	255593
DIJKSTRA	1	1	1	1	1	1	1	1
A*	0,409	0,588	0,978	0,828	0,737	0,459	0,313	0,264
UCF	0,875	0,492	0,606	0,614	0,576	0,581	0,605	0,563

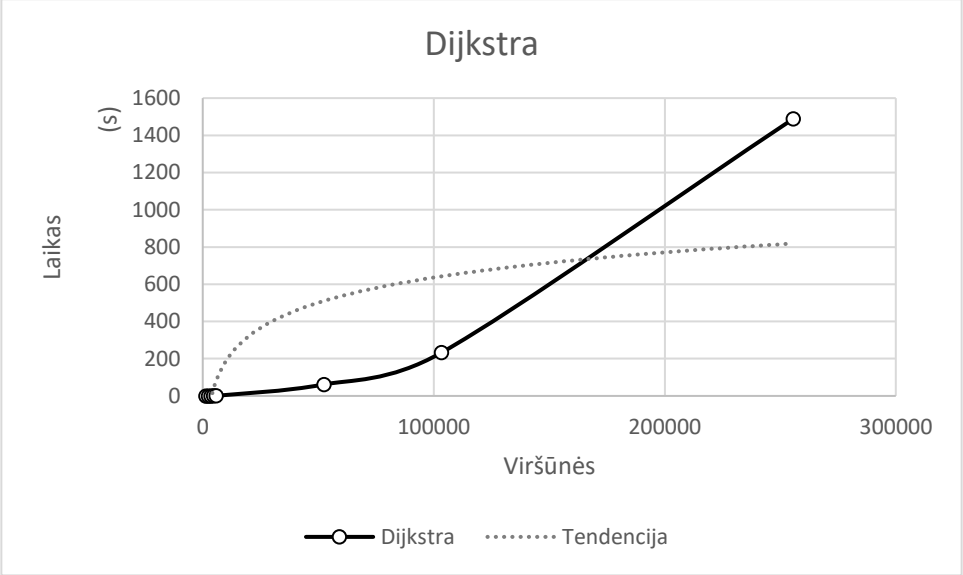


6.1 pav. Normalizuotų rezultatų grafikas.

Kaip matome iš Dijkstros algoritmas yra visais atvejais pranašesnis greitaveikos atžvilgiu. Žemiau, 6.3 lentelėje pateikiami nenormalizuoti Dijkstros rezultatai palyginimui.

6.3 lentelė. Nenormalizuoti Dijkstros algoritmo rezultatai.

VIRŠŪNĖS	1352	2469	3608	4665	5758	52432	103336	255593
DIJKSTRA	63	127	264	483	696	61812	232663	1488737



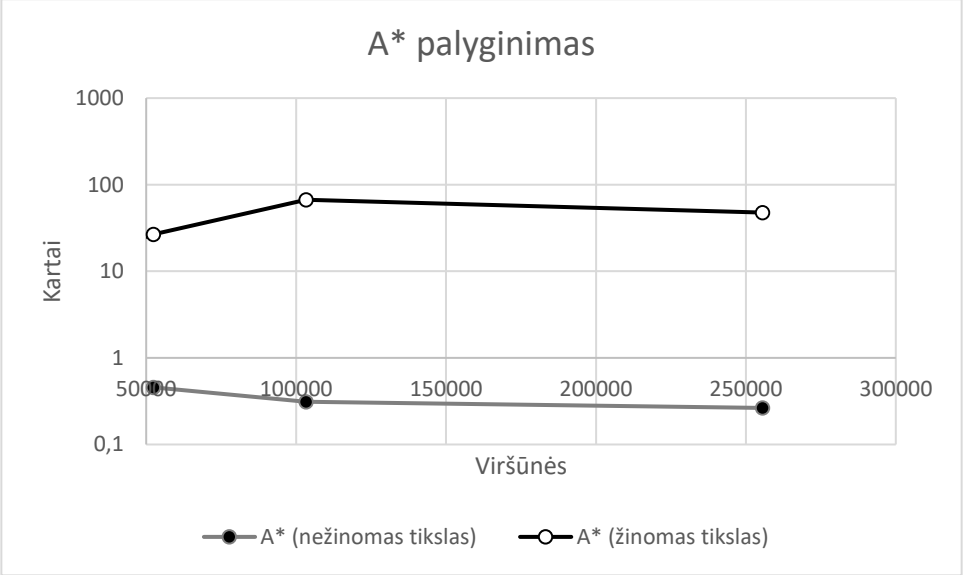
6.2 pav. Nenormalizuotų Dijkstros rezultatų grafikas.

Kaip matome iš grafiko Dijkstros algoritmo asimptotinis laikas yra $O(|E| + |V|log|V|)$ net blogiausiu atveju.

Lentelėje 6.2 pateikiama *A** algoritmo euristika iš anksto nežinojo kurioje vietoje yra išėjimo taškas, tam kad atlikti euristinius skaičiavimus *A** turi žinoti kurioje vietoje yra algoritmo ieškomas tikslas. Šiai problemai išspręsti buvo naudojamas paieškos gilyn algoritmas kuris neatsižvelgiant į grafų svorius suranda, jeigu tai padaryti įmanoma, siekiamą tikslą. Taip pat eksperimento metu išmatavome ir *A** algoritmo greitaveiką kai siekiamas tikslas jau yra žinomas, šį sprendimą galėjome priimti iš to, jog jau darėme prielaidą jog aplinka jau buvo ištirta. Žemiau esančioje lentelėje 6.4 pateiktas palyginimas tarp jau iš anksto žinomo tikslo bei tikslo kurio nežinome iš anksto. Duomenys taip pat pateikiami normalizuota forma pagal Dijkstros algoritmą. Grafiškai duomenys pateikti 6.3 paveiksle.

6.4 lentelė. Normalizuotas *A** palyginimas

VIRŠŪNĖS	1352	2469	3608	4665	5758	52432	103336	255593
A* (NEŽINOMA)	0,409	0,588	0,9778	0,827	0,737	0,458	0,313	0,264
A* (ŽINOMA)	6,300	4,097	264,000	7,5469	7,909	26,632	66,840	47,582

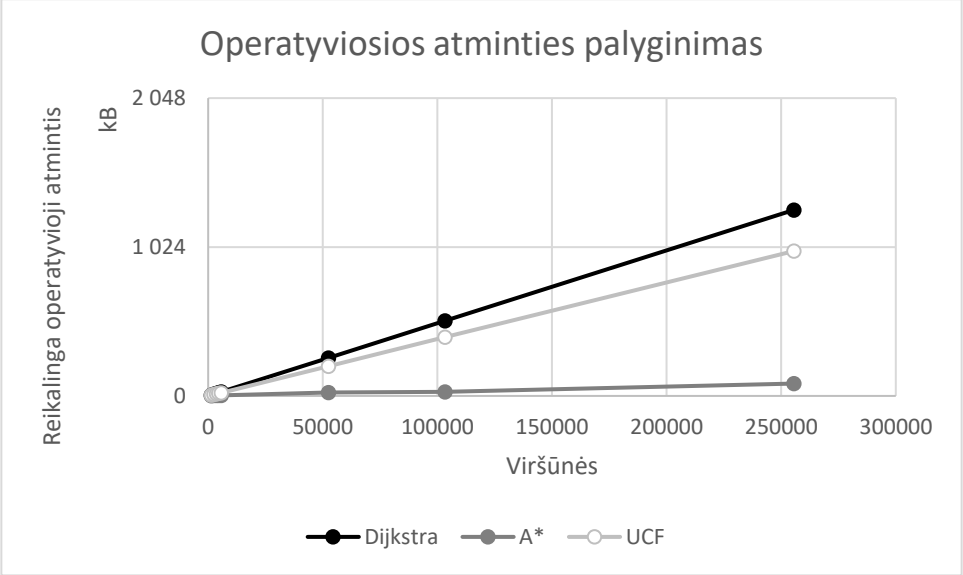


6.3 pav. Normalizuotas *A** palyginimas.

Žemiau pateiktoje lentelėje pateikiamas operatyviosios atminties sąnaudų reikalingų algoritmams rasti trumpiausią kelią palyginimas. Kaip matome iš 6.4 grafiko Dijkstros algoritmui reikalingos didžiausios sąnaudos iš tiriamų algoritmų. Dijkstros algoritmas dėl savo veikimo principo visada išskirs operatyviosios atminties kiekvienai viršūnei. Dėl šios priežasties darbas su begaliniais grafais naudojant Dijkstros algoritmą yra neįmanomas. Taip pat matome jog *A** yra pranašiausias atminties atžvilgiu. Tai daro algoritmą itin patrauklų įterptinėse sistemose kuriose operatyviosios atminties kiekis yra ribotas. Abu *A** algoritmo atvejai yra atvaizduojami tuo pačiu grafiku. Taip yra todėl, nes paieškos platyn algoritmui, kuriuo randamas euristinis tikslas, operatyviosios atminties reikia mažiau nei *A** algoritmui jo vykdymo metu.

6.5 lentelė. Reikalingos operatyviosios atminties (baitais) palyginimas.

VERTICES	1352	2469	3608	4665	5758	52432	103336	255593
DIJKSTRA	6929	12653,63	18491	23908,13	29509,75	268714	529597	1309914
A*	1288	3008	3008	4336	4948	23180	29868	86988
UCF	5408	9876	14432	18660	23032	209328	413332	1020324



6.4 pav. Operatyviosios atminties sąnaudų palyginimas.

6.3. Eksperimento išvados

1. Eksperimento metu ištyrėme ir palyginome *Dijkstros*, *A** bei *Uniform-cost search* algoritmus tarpusavyje atsižvelgiant į greitaveikos, rasto kelio ilgį bei operatyviosios atminties sąnaudas veikimo metu. Apžvelgę tyrimo rezultatus pastebėjome jog visi šie algoritmai turi sritį kurioje jie yra pranašesni už kitus.
2. Tyrimo metu nustatėme jog iš tirtų algoritmų *Dijkstros* algoritmas turi didžiausią greitaveiką kai nežinome kur yra tikslo viršūnė. Tačiau jeigu *A** algoritmas iškarto žino kur yra siekiamas tikslas t.y. jo nereikia ieškoti paieškos gilyn algoritmo pagalba, šis algoritmas greitaveikos atžvilgiu yra kelis kartus greitesnis už *Dijkstros* algoritmą. Turint geresnes euristines reikšmes yra įmanoma gauti dar geresnius rezultatus.
3. Eksperimento metu nustatėme *Dijkstros* algoritmas yra sparčiausias kai nežinome kur yra mūsų tikslas, tačiau šis algoritmas netinka aplinkoms kurių nesam pilnai ištyrinėję t.y. dėl savo veikimo principo netinka begaliniams grafams.
4. Tiek *A**, tiek *Uniform-cost search* algoritmai tinka panaudojimui „Vizualaus robotų programavimo ir simuliacinės aplinkos“ (*RobotVL*) sistemoje. Sistemoje siūloma šį algoritmą įgyvendinti kaip tas patį mazgo elementą esant parametrui kuris nurodo kur yra tikslas naudojamas *A** algoritmas, kai šis parametras nenurodomas naudojamas *UCS* algoritmas. Tokis sprendimas leis turėti optimalų sprendimą greičio atžvilgiu visais atvejais.

7. IŠVADOS

1. Suprojektuota bei sukurta programinė įranga leidžianti vartotojams kurti algoritmus mobiliesiems robotams. Kitaip nei daugelis kitų tokio pobūdžio programinės įrangos paketų – *RobotVL* leidžia kurti universalius algoritmus. Šiuos algoritmus galima pritaikyti ne tik bet kokiai aparatūrinei įrangai tačiau ji taip pat juos galima išbandyti ir virtualioje aplinkoje. Dėl to vartotojui besidominčiam robotų programavimu nereikia jokių papildomų išlaidų įsigyjant fizinius robotus.
2. Atlikta kelio planavimo bei aplinkos tyrimo algoritmų analizę. Analizės metu nustatyta jog nustatyta jog *D** algoritmas nėra tinkamas kuriamai sistemai. Tai yra dėl to, nes jis nesuteikia visiškos judėjimo laisvės robotams t.y. judėjimas apribojamas 45 arba 90 laipsnių inkrementais. Taip pat – šis algoritmas reikalauja jog kiekviena grafo viršūnė nuolatos žinotų apie pasikeitusią savo būseną, tam reikėtų kiekvienai viršūnei turėti sensorių – tai būtų per brangu perkeliant algoritmą į fizinę aplinką.
3. Eksperimento metu nustatėme *Dijkstros* algoritmas yra sparčiausias. Šis algoritmas turi mažiausią asimptotinį laiką kai nežinome kur yra mūsų tikslas. Tačiau dėl savo veikimo principo šis algoritmas netinka aplinkoms kurių nesam pilnai ištyrinėję t.y. dėl savo veikimo principo netinka begaliniams grafams.
4. Tyrimo metu buvo ištirti *Dijkstros*, *A** bei *Uniform-Cost Search* algoritmai pagal nustatytus algoritmo kokybės kriterijus. Tyrimo metu nustatyta, jog tinkamiausias pasirinkimas įgyvendinimui sistemoje yra A* bei *Uniform-Cost Search* kombinacija. Šis sprendimas leis turėti optimaliausią kelio planavimo sprendimą tiek laiko, tiek kelio optimalumo atžvilgiu.

8. LITERATŪRA

[1] M. Fattah, „A Low-Overhead, Fully-Distributed, Guaranteed-Delivery Routing Algorithm for Faulty Network-on-Chips," įtraukta *International Symposium on Networks-on-Chip,* 2015.

[2] D. Nieuwenhuisen, A. Kamphuis ir M. H. Overmars, „High quality navigation in computer games," *Science of Computer Programming,* t. 67, nr. 1, pp. 91-104, 2007.

[3] P. Xavier, „Shortest path planning for a tethered robot," *Robotics and Automation,* t. 2, pp. 1011-1017, 1999.

[4] E. W. Dijkstra, „A note on two problems in connexion with graphs," *Numerische Mathematik,* t. 1, nr. 1, pp. 269-271, 1959.

[5] P. Hart, N. Nilsson ir B. Raphael, „A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *Systems Science and Cybernetics,* t. 4, nr. 2, pp. 100 - 107, 1968.

[6] S. Koenig ir M. Likhachev, „Fast Replanning for Navigation in Unknown Terrain," *Robotics,* t. 21, nr. 3, pp. 354-363, 2005.

[7] M. Likhachev, G. Gordon ir S. Thrun, „ARA*: Anytime A* with Provable Bounds on Sub-Optimality," įtraukta *IN ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 16,* Whistler, 2004.

[8] K. Tanakitkorn, „Grid-based GA path planning with improved cost function for an over-actuated hover-capable AUV," įtraukta *AUV2014,* 2014.

[9] S. M. Persson ir I. Sharf, „Sampling-based A* algorithm for robot path-planning," *The International Journal of Robotics Research,* t. 33, nr. 13, pp. 1683-1708, 2014.

[10] X. Cui ir H. Shi, „A*-based Pathfinding in Modern Computer Games," *International Journal of Computer Science and Network Security,* t. 11, nr. 1, pp. 125-130, 2011.

[11] A. Stentz, „Optimal and Efficient Path Planning for Partially Known Environments," *Intelligent Unmanned Ground Vehicles,* pp. 203-220, 1997.

[12] S. Koenig ir M. Likhachev, „D* Lite," *Eighteenth national conference on Artificial intelligence,* pp. 476-483, 2002.

[13] D. Ferguson, N. Kalra ir A. Stentz, „Replanning with RRTs," *Robotics and Automation,* pp. 1243-1248, 2006.

[14] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli ir S. Teller, „Anytime motion planning using the RRT*," *Robotics and Automation,* pp. 1478-1483, 2011.

[15] „Valve Developer Community," [Tinkle]. Available: https://developer.valvesoftware.com/wiki/KeyValues_class. [Kreiptasi 12 Balandis 2017].