KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Deividas Stonys

VIZUALIOS ROBOTŲ PROGRAMAVIMO KALBOS KŪRIMAS IR TYRIMAS

Baigiamasis magistro projektas

Vadovas
lekt. dr. Mikas Binkis

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

VIZUALIOS ROBOTŲ PROGRAMAVIMO KALBOS KŪRIMAS IR TYRIMAS

Baigiamasis magistro projektas
Programų sistemų inžinerija (621E16001)

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

Recenzentas
(parašas) doc. dr. Lina Čeponienė
(data)

Projektą atliko
(parašas) Deividas Stonys
(data)

KAUNAS, 2017

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos

(Fakultetas)

Deividas Stonys

(Studento vardas, pavardė)

Programų sistemų inžinerija, 621E16001

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Vizualios robotų programavimo kalbos kūrimas ir tyrimas“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 17 m. gegužės d.

Kaunas

Patvirtinu, kad mano, **Deivido Stonio**, baigiamasis projektas tema „Vizualios robotų programavimo kalbos kūrimas ir 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)*

Stonys, Deividas. Vizualios robotų programavimo kalbos kūrimas ir tyrimas. Magistro baigiamasis projektas / vadovas lekt. dr. Mikas Binkis; Kauno technologijos universitetas, Informatikos fakultetas.

Reikšminiai žodžiai: *vizuali programavimo kalba, robotų valdymas, programavimo mokymasis*.

Kaunas, 2017. 65 p.

SANTRAUKA

Pagrindinis šio darbo tikslas buvo sukurti vizualaus robotų programavimo ir robotų sąveikos modeliavimo aplinkos programą, kuri paskatintų kitaip pažvelgti į programavimo mokymąsi. Programavimo siūloma mokyti per programavimo mokymo programą su žaidybiniais elementais, kurioje naudojant vizualią robotų programavimo kalbą yra valdomas roboto judėjimas virtualioje programos aplinkoje. Norint, jog robotas atliktų skirtingas užduotis, jos aprašomos vizualia robotų programavimo kalba. Pasirinktas vizualios programavimo kalbos, roboto elgsenos algoritmų užrašymas, leidžia algoritmus užrašyti vartotojui, neturinčiam jokio arba turinčiam tik minimalų suvokimą apie programavimą.

Analizuojant panašių programų vizualias robotų programavimo kalbas pastebėta, jog šios kalbos nors ir kurtos patirties neturintiems vartotojams, tačiau laikui bėgant jų suprantamumas sumažėjo atsiradus papildomam funkcionalumui. Atsižvelgiant į jų struktūrą ir suteikiamą funkcionalumą, buvo sukurta vizuali robotų programavimo kalba pritaikyta vartotojams, turintiems mažą programavimo patirtį ir jo supratimą. Užrašytas vizualios programavimo kalbos kodas transformuojamas į C# programavimo kodą. Dėl šios priežasties, algoritmas, parašytas šia vizualia programavimo kalba, galės būti naudojamas ne vien šioje sistemoje, valdant virtualius robotus, bet ir šiek tiek pakoregavus C# programinį kodą, būtų lengvai pritaikomas tikrų robotų programavimui. Sistema realizuota grafinio 3D „Unity 5“ variklio aplinkoje ir C# programavimo kalba. Tikrinant ar vizualios programavimo kalbos sukurti mazgai tinkamai paverčiami į C# programinį kodą buvo sukurti automatizuoti testai.

Norint patvirtinti sukurtos vizualios programavimo kalbos privalumus buvo atlikti šios „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos parašytų algoritmų ir jų transformuoto C# programinio kodo metrikų eksperimentas. Šio eksperimento tikslas gautus rezultatus palyginti tarpusavyje ir rasti su tuo susijusias problemas.

Kito eksperimento metu buvo tikrinamas vartotojų naudojimasis „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualiomis programavimo kalbomis. Kiekvienas vartotojas turėjo su šiomis vizualiomis programavimo kalbomis pabandyti įgyvendinti tokį patį roboto elgesio algoritmą. Įgyvendinus algoritmą, skirtingomis vizualiomis programavimo kalbomis, vartotojai užpildė standartizuotą SUS (*angl. System Usability Scale*) klausimyną. Apibendrintus gautus rezultatus galima nusakyti vizualios programavimo kalbos SUS balą, kuris parodo jos naudojimosi lengvumą. Taip pat

šis eksperimentas nusako vartotojų nuomonę į tai, kaip galima pagerinti vizualią programavimo kalbą, jog kalba būtų patogesnė ir suprantamesnė vartotojui.

Stonys, Deividas. Research and Development of Visual Robot Programming Language. Master's final project / supervisor lect. ph. d. Mikas Binkis. The Faculty of Informatics, Kaunas University of Technology.

Key words: *visual programming language, robot control, learn programming*.

Kaunas, 2017. 65 p.

SUMMARY

The main goal of this project was to create visual robot programming and robotic interaction simulation environment program, which would encourage otherwise to look at programming learning. The programming offered to teach through programming tutorial with gamification in the visual robot programming language controlled robot movements in a virtual program environment. In order achieve that the robot would perform different task visual programming language was chosen. The selected visual programming language robot behaviour algorithms notation allows a user that does not have any or have only minimal understanding of programming to write algorithms.

The analysis of similar visual robot programming language programs observed that languages created by inexperienced users, but over time their intelligibility decrease for additional functionality. Visual robot programming language was developed depending on their structure and functionality. This language could be used by people with low programming experience and understanding. Written, visual programming language code is transformed into the C# programming code. For this reason, the algorithm written this visual programming language that can be used not only in system for managing virtual robots, but also adjusted C# source code can be easily adapted to real robots programming. The system was realized in 3D graphics "Unity 5" engine environment and C# programming language. Checking whether the visual programming language to create nodes correctly converted to C# programming code have been formed automatic test scripts.

To confirm the advantages of visual programming languages have been performed "RobotVL" and "Microsoft Robotics Developer Studio" visual programming languages written algorithms and their transformed to C# programming code metrics experiment. This experiment purpose to compare results with each other and find related problems.

Another experiment was tested how consumers use created "RobotVL" and "Microsoft Robotics Developer Studio" visual programming languages. Each user should try to implement the same algorithm for robot behaviour with these visual programming languages. Users filled a standardized questionnaire SUS (System Usability Scale), then implemented algorithm for different visual programming languages. Summarized results can be defined visual programming language SUS score, which indicates system usability. Also, this experiment describes users feedback, how to improve the visual programming language, that language would be more convenient and understandable for user.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

TERMINŲ IR SANTRUMPŲ ŽODYNAS

Terminas ar santrumpa	**Aprašymas**
.NET asembleris (*angl. assembly*)	.NET karkaso statybiniai blokai. Jie sudaro esminį vieneto darbą, versijų kontrolę, pakartotinio panaudojimo, apimties nustatymą ir saugumo leidimą. Gali būti įvairių rūšių ir išteklių, jog sukurtų kartu dirbančius ir derančius loginius funkcionalumo vienetus. .NET asembleris suteikia bendrą kalbos vykdymą su žinomo tipo realizuota informacija. Aplinkos vykdyme tipas neegzistuoja už .NET asemblerio konteksto.
Binarinis operatorius (*angl. binary operator*)	Skaičiavimų rinkinys, kuris jungia du elementus, jog gautų kitą elementą. Tai apima žinomus elementarius aritmetinius veiksmus: sudėtis, atimtis, daugyba ir dalyba.
Kompiliatorius (*angl. compiler*)	Kompiuterinė programa arba programų rinkinys, kuris parašytą programavimo kalbos kodą transformuoja į kitą programavimo kalbą, turinčią dvejetainę formą, jog būtų galima sukurti vykdomąją programą.
Liuksmetras	Prietaisas skirtas apšvietai matuoti. Apšvieta – šviesos srautas, tenkantis vienetiniam paviršiaus plotui.
Mazgas (*angl. node*)	Vizualios programavimo sąsajos elementas atitinkantis vienetinį programavimo kalbos funkcionalumą (ciklas, sąlyga, parametrai) ar komandą (pirmyn, sustoti, pasukti, groti melodiją).
PBD (*angl. Programming By Demonstration)*	Galutinio vartotojo vystymosi metodas, kai mokoma kompiuterį ar robotą naujo elgesio, perduodant užduotį tiesiogiai, o ne per programavimo komandas.
SUS (*angl. System Usability Scale*)	Standartinis 10 klausimų klausimynas programinės įrangos tinkamumui vertinti.
UML (*angl. Unified Modelling Language*)	Specifikacijos kūrimo kalba, skirta atvaizduoti objektiškai orientuotų programų dokumentus.
Vizuali programavimo kalba (*angl. VPL*)	Bet kokia programavimo kalba, kuri leidžia manipuliuoti jos mazgų elementais grafinėje programavimo aplinkoje, siekiant įgyvendinti įvairius algoritmus.

1. ĮVADAS

Šis magistro baigiamasis darbas atliktas studijuojant programų sistemų inžinerijos studijų programą. Jo metu sukurta vizualaus robotų programavimo ir robotų sąveikos modeliavimo aplinkos programa, norint paskatinti kitaip pažvelgti į programavimo mokymąsi. Šio mokslinio tyrimo tikslas būtų išanalizuoti vartotojų mokymosi ir prisitaikymo pasitenkinimą dėl vizualios programavimo kalbos naudojimo ir nors nedidele dalimi prisidėti prie jo pagerinimo.

1.1. Tyrimo sritis ir aktualumas

Šiuo metu informatikos specialistų paklausa tiek Lietuvoje, tiek pasaulyje yra viena iš didžiausių. Labiausiai trūksta programuotojų, todėl reikėtų paskatinti moksleivius domėtis šia sritimi. Viena iš tokių priemonių, galinčių supažindinti ir paskatinti domėtis programavimu, galėtų būti, jog mokyklose mokytojai supažindintų su programavimu per informatikos pamokas. Visgi moksleivius retai bandoma supažindinti informatika mokyklose, tik 2100 iš maždaug 42000 aukštųjų mokyklų JAV turi tinkamą informatikos kursą. Nuo 2005 metų tokių kursų mokymas sumažėjo 17 % [1].

Norint pagerinti šią situaciją reikalingos žaidybinės programos, kurios skatintų domėjimąsi ir mokytų programavimo logikos. Šiuo metu internete galima rasti įvairaus tipo žaidybiniu principu sukurtų programavimo mokymosi alternatyvų. Tokios programos informaciją pateikia iliustruotoje vaizdo medžiagoje, paveikslėliuose su paaiškinimais ar glaustuose programinio kodo pavyzdžiuose. Šie būdai palengvina svarbiausios informacijos įsisavinimą, o knygos lieka kaip papildomas informacijos šaltinis. Visgi šie būdai daugiau orientuojasi į žmones, kurie jau yra susipažinę su programavimo logika, tačiau nori pagilinti savo žinias.

Siūlomas sprendimas yra orientuotas į norinčius išmokti programavimo, nepriklausomai nuo amžiaus ar programavimo patirties, todėl bus mokoma programavimo per specialiai tam sukurtą vizualią programavimo kalbą, kuri yra supaprastinta tikros programavimo kalbos alternatyva. Taip moksleivis galėtų lengviau perprasti pačią programavimo logiką ir lengviau suprasti kitas programavimo kalbas. Visgi norint išlaikyti moksleivio susidomėjimą, reikia parodyti, ką galima padaryti su programavimu. Dėl šios priežasties sukurta vizuali programavimo kalba leis valdyti robotus virtualioje aplinkoje ir stebėti, kaip daromi vizualios programavimo kalbos pakeitimai veikia roboto elgseną. Robotų valdymas pasirinktas dėl vis didėjančio jo populiarumo, pritaikomumo ir galimybių, kurias galima su juo išbandyti. Vizualios programavimo kalbos naudojimas išsprendžia problemą, jog nepatyręs vartotojas, kuris nesupranta programavimo kalbų sintaksės, gali parašyti savo taisykles, pagal kurias judėtų robotas, o virtuali modeliavimo aplinka išsprendžia problemą dėl įvairių įrangos gedimų ir tinkamo veikimo (dalys dėvisi ir atsiranda netikslumų). Roboto valdymo su vizualia programavimo kalba sprendimai jau egzistuoja, tačiau dauguma jų pritaikyti patyrusiems programuotojams, todėl manoma, jog šis sprendimas turėtų pasiteisinti ir būti įdomus būdas išmokti programuoti norinčius vartotojus.

1.2. Darbo tikslas

Projekto tikslas yra sukurti sričiai orientuotą (*angl. domain specific*), riboto funkcionalumo vizualią programavimo kalbą, kuria būtų galima paskirti tam tikras užduotis robotui ir jį valdyti. Vizuali programavimo kalba turi būti paprastai ir greitai perprantama nepatyrusio programavimo vartotojo. Visgi norint ištirti algoritmų veikimą, sukurta vizuali programavimo kalba turi leisti atlikti pagrindinius programavimo veiksmus ir realizuoti esamus algoritmus. Visa padaryta programavimo logika pavaizduojama roboto judėjime virtualioje aplinkoje, kurioje bus galima rinktis roboto komponentus. Šie komponentai yra standartizuoti, todėl roboto funkcijos bus klasikinės, tokios kaip judėjimas, atstumo fiksavimas iki kliūties, spalvos atpažinimas, žodžio parodymas ekrane. Visgi funkcijos bus ribotos, nes tam tikri virtualūs jutikliai teiks tik jiems būdingą informaciją, kurią naudojant galima sukurti ir išbandyti įvairius algoritmus.

1.3. Tyrimo objektas

Tyrimo objektas – vizuali programavimo kalba. Vizuali programavimo kalba atitinka bet kokią programavimo kalbą, kuri turi grafinę programavimo aplinką, leidžiančią manipuliuoti skirtingais mazgų elementais, siekiant įgyvendinti įvairius algoritmus. Tokie elementai atitinka vienetinį programavimo kalbos funkcionalumą, tokį kaip ciklą, sąlygą ar parametrus. Visgi norint padidinti vartotojo vizualios programavimo kalbos skaitomumą, mazgų vaizdavimai dažniausiai skiriasi, priklausomai nuo jo tipo. Tokie mazgai skiriasi forma, spalva, įvedamų ar gaunamų parametrų kiekiu ir reikšmėmis. Taip pat skiriasi pačios vizualios programavimo kalbos aplinka, kurioje galima manipuliuoti esamais ir naujai sukurtais mazgais. Siekiant išsiaiškinti, kaip padaryti vizualią programavimo kalbą skaitomesnę ir paprastesnę, reikalingas tyrimas, kurio metu būtų analizuojami vizualių programavimo kalbų naudojimosi būdai.

1.4. Darbo uždaviniai

Šiame darbe buvo suformuluoti šie uždaviniai:

1. Išanalizuoti robotams valdyti skirtas vizualias programavimo kalbas, išsiaiškinti jų privalomus ir trūkumus.
2. Suprojektuoti ir sukurti vizualią programavimo kalbą, kuria būtų galima valdyti virtualioje aplinkoje roboto elgseną.
3. Pagal pasirinktus kriterijus palyginti sukurtą vizualią programavimo kalbą su kitomis.

2. VIZUALAUS ROBOTŲ PROGRAMAVIMO APLINKOS ANALIZĖ

2.1. Analizės tikslas

Analizės tikslas – atlikti panašaus funkcionalumo vizualių programavimų kalbų analizę, kuriomis būtų galima valdyti roboto elgseną, kad jis atliktų jam paskirtas užduotis. Analizės metu išanalizuojama, kokius privalomumus ir trūkumus gali turėti tokios vizualios programavimo kalbos, o jiems atsiradus - kaip su jais tvarkomasi. Analizės tikslui pasiekti bus atliktas vizualių programavimo kalbų tinkamumo, funkcionalumo ir naudojamumo įvertinimas. Be viso to bus siekiama pasinaudoti kitų mokslininkų atliktais vizualių programavimo kalbų tyrimais, pritaikant vizualią kalbą specifiniam panaudojamumui, tokiam kaip robotų sąveikos užtikrinimas.

Svarbiausias dėmesys tyrimo objekto analizės dalyje bus skiriamas užtikrinti vizualios programavimo kalbos paprastumui ir greitam išmokstamumui programuoti nepratusiam vartotojui. Visgi sukurta vizuali programavimo kalba turi leisti atlikti pagrindinius programavimo veiksmus ir suteikti galimybę realizuoti jau esamus algoritmus.

2.2. Egzistuojantys sprendimai

Rinkoje egzistuoja įvairių vizualių programavimo aplinkų, dauguma iš jų pritaikytos vartotojams, turėjusiems programavimo patirties anksčiau. Toliau bus nagrinėjamos labiausiai žinomos ir naudojamos vizualios programavimo aplinkos, kurios skirtos valdyti tiek realiems, tiek virtualioje aplinkoje esantiems robotams.

2.2.1. „Scratch“

„Scratch“ – programavimo aplinka, pritaikyta vaikams kurti interaktyvias istorijas ir paprastus vaizdo žaidimus. Sukūrus savo projektą, juo galima pasidalinti su kitais. Ši programa yra nemokama, o jos pritaikymo galimybės tikriems robotams leidžia įgyvendinti net sudėtingas elgesio valdymo užduotis [2]. Visgi pagal kūrėjus ši programavimo sąsaja yra supaprastina, jog ja galėtų kuo paprasčiau naudotis vartotojai, nepriklausomai ar jie turėjo anksčiau su programavimu susijusios patirties, ar ne. Dėl šios priežasties ši kalba naudojama „Jaunųjų kompiuterininkų mokyklos“ (JKM) programavimo mokyme, norint pagerinti mokymosi efektyvumą [3].

„Scratch“ kalbos sėkmingumas yra tai, jog ji susieta su bendruomene, kurioje žmonės padeda, bendrauja ir kritikuoja vienas kito padarytus darbus. Per 27 mėnesius nuo „Scratch“ paleidimo jų internetinėje svetainėje buvo paskelbta ir pasidalinta daugiau nei 500000 projektų, sukurtų naudojant šią vizualią kalbą. Pateikdami savo projektą, kūrėjai sulaukia įvairių patarimų ir pasiūlymų iš kitų kūrėjų, kurie padeda greičiau tobulėti [4]. Šiuo metu „Scratch“ vizualia kalba sukurta ir pasidalinta per 11590000 projektų, taip pat „Scratch“ kalba yra naudojama daugiau nei 150 valstybių ir jau pasiekiama daugiau nei 40 skirtingų kalbų.

„Scratch 1.0“ turėjo 92 komandų blokus, dar vadinamais mazgais. „Scratch“ vystantis buvo visada norima sumažinti komandų skaičių. Visgi kiekvienos naujos versijos išleidimas prideda naujų

galimybių, funkcijų ir komandų. Kartais mazgai pašalinami, kai jų suteikiamas funkcionalumas nebėra aktualus ar sutampa su kitų mazgų galimybėmis. Vis dėlto buvo pridėta daugiau mazgų, nei pašalinta. „Scratch 1.4“ turi 125 komandų blokus, nors kai kurie nėra rodomi, kol jų nereikia, bet naujam vartotojui būna gana sudėtinga pasirinkti tinkamą komandą iš sąrašo. „Scratch“ vizualios programavimo kalbos demonstravimas paleidžiant du veiksmus parodytas 2.1 paveikslėlyje [5].

2.1 pav. Transliavimo (*angl. broadcast*) komanda su dviem veiksmais

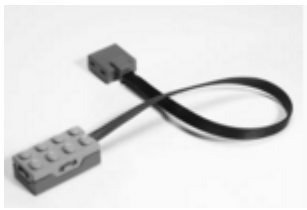
„Scratch“ vizualios programavimo kalbos pagrindas sudarytas iš keturių mazgų rūšių, kurie pavaizduoti 2.1 lentelėje [5]. Šios „Scratch“ pagrindinės mazgų rūšys užtikrina, jog būtų galima sukurti projektą pagal vartotojo pageidavimus. Tereikia jungiant šiuos mazgus pasirinkti atitinkamą funkcionalumą atliekančią komandą.

2.1 lentelė. „Scratch“ mazgų tipai

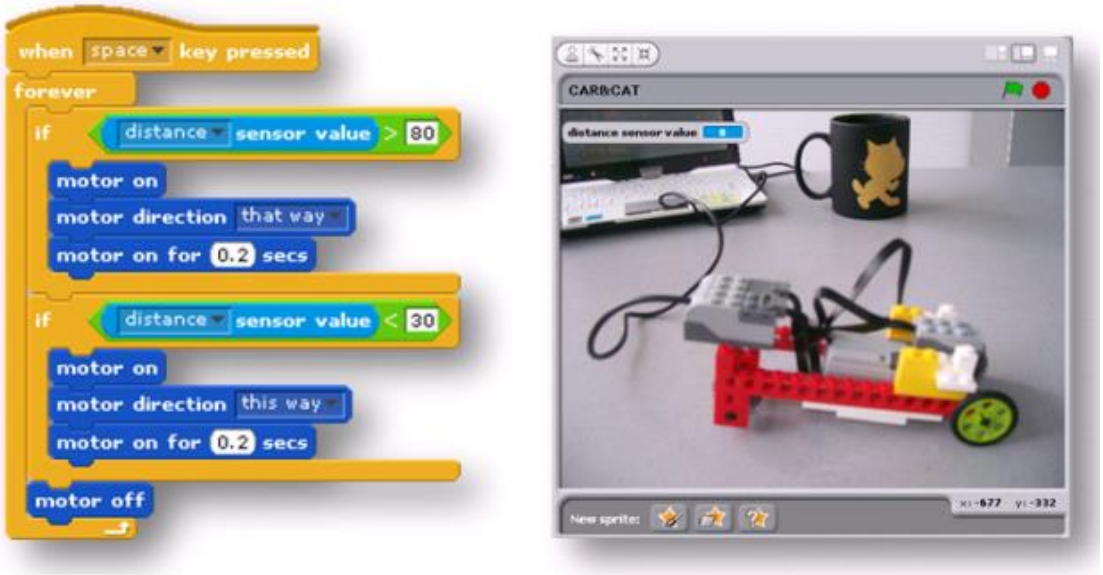
	*Komandos* blokas turi išpjovą viršuje ir atitinkamai griovelį apačioje. Komandos blokus jungiant galima sukurti komandų seką, vadinamą *krūvą*.
	*Funkcijos* blokas grąžina reikšmę. Funkcijos blokai neturi griovelių.
	*Trigerio* blokas turi apvalų viršų. Jis paleidžia žemiau esančią taktiką, kai įvyksta nustatytas įvykis.
	*Valdymo* struktūros komandos blokai turi angas, skirtas laikyti komandų sekas, kurios paleidžiamos pagal sąlygos rezultatą.

Ši vizuali programavimo kalba pritaikoma tikrų robotų valdymui. „Automobilis ir Katė“ (*angl. Car and Cat*) yra edukacinis projektas, kurio tikslas yra sukurti automobilį, kuris išlaikytų saugų atstumą tarp katės (puodelis judinamas rankomis). Automobilis konstruojamas iš „Lego Education WeDo tollkit“ (dalys pavaizduotos 2.2 lentelėje), o jo valdymui skirta programinė įranga kuriama „Scratch“ vizualios programavimo kalbos aplinkoje.

2.2 lentelė. „Lego Education WeDo tollkit“ dalys

WeDo dalis	Kaip atrodo	Kam naudojama
Variklis		Skirtas priversti judėti roboto dalims.
Šviesos		Jie nėra įtraukti į *WeDo* komplektą, tačiau vis dar galima juos prijungti ir valdyti per „Scratch“.
Atstumo jutikliai		Jutiklis nusako, kiek nuo jo nutolęs koks nors kitas objektas.
Pakrypimo jutiklis		Jutiklis nusako pakrypimą.
Jungtis		Sujungia *WeDo* dalis su jūsų kompiuteriu.

Projektuoti ir kurti automobilį galima kokį tik norima. Galimo automobilio išvaizda parodyta 2.2 paveikslėlyje. Automobilio priekyje yra du, o gale vienas ratas, kurie krumpliaračiu prijungti prie variklio. Raudonas automobilio rėmelis laiko variklį, ratus ir jutiklius, kurie perduoda informaciją į „Scratch“ programos aplinką. Sudedant iš „Scratch“ elementų programinį automobilio valdymo algoritmą, galima išmėginti jo elgseną, keisti kodą, atsižvelgiant į pastebėtas problemas [2].



2.2 pav. Robotas, kuriame įgyvendintas savarankiškas judėjimas

2.2.2. NXT-G

NXT-G – vizuali programavimo kalba sukurta „Lego“ bendradarbiaujant su „LabVIEW“ dėl „Lego Mindstorms NXT“ robotų. Ši sukurta vizuali kalba tikslinga tiek vaikams, tiek suaugusiems, kurie neturi programavimo patirties. Vizuali programavimo kalba sudaryta blokelių pagrindu: kiekvienam fiziniam „Lego“ blokelyje yra paslėptas skaitmeninis įgyvendinimas, kurie kartu leidžia įgyvendinti numatytą elgsenos seką. Jutiklių blokeliai gauna įvairių duomenų iš išorinio pasaulio, kuriuos apdorojama ir persiunčiama į kitus blokelius.

Išsamus roboto elgesys pagal atitinkamai parašytą NXT-G programos kodą parodytas 2.3 paveikslėlyje [6]. Visgi nors ir vizualaus programavimo aplinka orientuota į roboto programavimą, ji aprašoma gana sudėtingai, norint pritaikyti sudėtingesnius algoritmus.

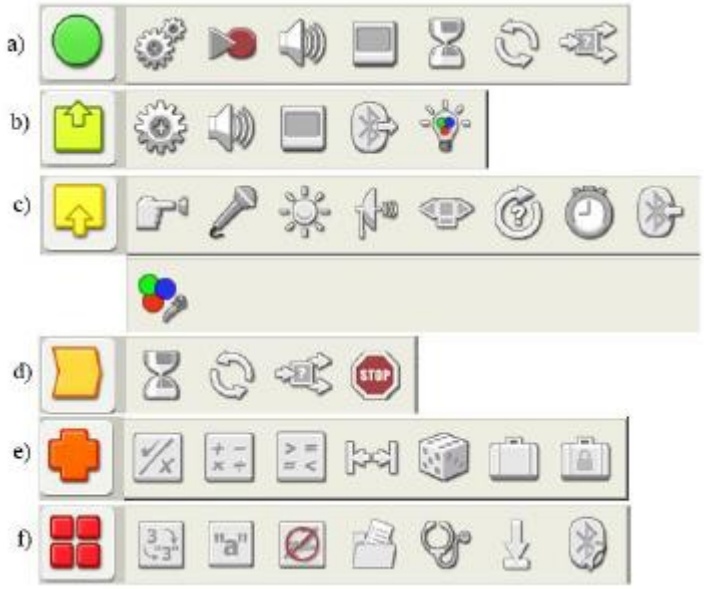
2.3 pav. NXT-G vizualaus programos kodo demonstravimas

Pavaizduoto 2.3 paveikslėlio programos kodo veikimas:

- Robotas pasako „Labas“;
- 2 sekundes ekrane parodomas šypsenėlės veidas;
- Abiejuose pusėse įjungiamos kairės ir dešinės šviesos;
- Važiuoja 2 sekundes pirmyn;
- Sustoja.

NXT-G blokai, pavaizduoti 2.4 paveikslėlyje, suskirstyti į 6 kategorijas [7]:

a) *Bendri* – judėjimo ir kiti dažniausiai naudojami blokų veiksmai;
b) *Veiksmai* – bendri roboto kontrolės veiksmai, tokie kaip variklio, garso ar ekrano valdymas;
c) *Jutikliai* – blokai kurie leidžia gauti įvairių duomenų iš išorės;
d) *Srautas* – srauto kontroliavimo blokai, tokie kaip „Laukti“ (*angl. Wait*) ar „Ciklas“ (*angl. Loop*);
e) *Duomenys* – blokai, leidžiantys apibrėžti kintamuosius ir atlikti matematinius arba loginius veiksmai su jais.
f) *Išplėstiniai* – papildomų veiksmų blokai, tokie kaip failo prieigos ir panašūs.



2.4 pav. „Lego“ blokai ir jutiklių blokai

Kiekvienas blokas gali būti papildomai keičiamas naudojant vizualų nustatymų langą. Toks judėjimo blokelio nustatymo langas pavaizduotas 2.5 paveikslėlyje. Iš šio paveikslėlio matoma, jog vizualiai galima redaguoti kryptį, trukmę, judėjimo stiprumą, nustatyti, kurioms prijungtoms išvestims takyti šias taisykles (A, B, C) ar ką daryti užbaigus šį veiksmą.

2.5 pav. Judėjimo blokelio nustatymų langas

2.2.3. „Microsoft Robotics Developer Studio“

Sudėtinga programinė įranga skirta pažengusiems vartotojams, kuri naudojama programuoti roboto elgsenai. Visgi trūkstant galimybių arba pasitikint savimi galima persijungti į rėžimą pažengusiems (*angl. advanced*). Šiame režime vizualios kalbos kodo rašymas pakeičiamas į

programinio kodo rašymą. Toks pasirinktas rašymas suteikia didesnes galimybes programuoti roboto elgseną, tačiau net ir pažengusiems vartotojams gali būti sudėtinga. Ši programinė įranga gali valdyti tiek realų fizinį robotą, tiek simuliuoti virtualią aplinką ir įkelta robotą, kuris galės atlikti jam numatytus veiksmus. Tai naudinga norint išbandyti robotą tam tikroje aplinkoje, o nesėkmės atveju jo nesugadinti [8].

„Microsoft Robotics Developer Studio“ vizualaus programavimo kalba (*angl.VPL*) yra grafinė programavimo aplinka, skirta įgyvendinti įvairius robotų elgsenos algoritmus. Visiems šiems komponentams „Microsoft Robotics Developer Studio“ komanda pateikė pabaigtas programas su įgyvendintais panaudojimo pavyzdžiais [9]. Pagrindiniai šios vizualios programinės kalbos elementai pateikti 2.3 lentelėje [10].

2.3 lentelė. „Microsoft Robotics Developer Studio“ mazgų tipai

	*Duomenys* - duomenų komponentai naudojami nustatyti pradinės kintamojo arba įvesties reikšmei, jog galėtų apskaičiuoti komponentą. Tipas gali būti bet koks kurį palaiko .NET programavimo kalba.
	*Sąrašas* - sąrašų komponentai naudojami siekiant sukurti tam tikrą konkretų duomenų tipą, kuriuo naudosis kitų veiklų komponentai.
	*Sąrašas funkcijų* – sąrašo funkcijų komponentai naudojami atlikti operacijas su sąrašų komponentuose esančiais elementais. Šias operacijas apima *Append, Concatenate, Reverse, RemoveItem, InsertItem* ir *GetIndex*.
	*Sujungti* – sujungimo komponentai naudojami sujungti kelias įvesties reikšmes ir gauti vieną išeities reikšmę. Šis komponentas turės tik išėjimo pranešimą, kai dvi įvesties reikšmės bus pateiktos į komponentą.
	*Sulieti* - suliejimo komponento tikslas yra sulieti kelis šaltinius į vieną. Šį komponentą pradedantieji vartotojai dažnai painioja su sujungimo komponentu. Nors suliejimo komponentas palaiko keletą įvesties reikšmių ir pateikia tik vieną išėjimo reikšmę, taip pat gali veikti net jei buvo įvesta viena reikšmė. Tai yra pagrindinis skirtumas tarp suliejimo ir sujungimo komponento.

	*Jeigu* – sprendimo komponentas turi tą pačią reikšmę kaip tradicinės programos konstruktorius „Jeigu... tada... priešingu atveju...“ (*angl. „If... then... else...“*). Jo sintaksė apima „lygu“ (atitinka du lygybės ženklus, ==), „mažiau nei“ (atitinka <), „daugiau nei“ (atitinka >) ir „nelygu“ (atitinka !=).
	*Perjungti* - perjungimo komponente pasirinkimas nustatomas pagal įvesties reikšmes ir sprendimų modulį. Jų funkcija yra panaši į perjungimo (*angl. switch*) sintaksę C++ programavimo kalboje.
	*Komentaras* - komentavimo komponentai naudojami pridėti komentarus prie veiklos komponentų, jog būtų galima paaiškinti jų veikimą ar pateikti kitą susijusią informaciją.
	*Pasirinktinė veikla* – pasirinktos veiklos komponentas naudojamas, jog vartotojas galėtų sukurti savo veiklą. Šis komponentas leidžia vartotojui nustatyti įvairius veiklos pasirinkimus. Kiekviena sukurta veikla gali gauti įvesties ir gražinti išvesties reikšmę, taip pat iškviesti pranešimų atvejį.

2.3. Programų sistemų savybių kiekybinis ir/ arba kokybinis palyginimas

Išanalizavus naudojamas vizualias programavimo kalbas ir jų aplinkas, kurios skirtos valdyti robotų elgsenai. Gauti apibendrinti analizės rezultatai pateikti 2.4 lentelėje.

Analizei naudojami šie kriterijai:

- Pritaikyta tik robotų valdymui – vizualaus programavimo aplinka skirta ne vien tik robotų elgsenos valdymui.
- Režimas pažengusiems (*angl. advanced*) – galimybė perjungti vizualią programavimo aplinką į kompiliuojamą programinės kalbos kodo tekstinį redaktorių.
- Programinio kodo generavimas – ar vizuali programavimo aplinka sugeba iš sudėtų elementų paversti į kokią nors kompiliuojamą programinės kalbos kodą.
- Vartotojo sąsaja turi lietuvių kalbą – ar galima vizualioje programavimo kalbos aplinkoje pakeisti vartotojo sąsajos kalbą į lietuvių.
- Pritaikyta „LEGO MINDSTORMS“ robotams – ar galima vizualią programavimo kalbą pritaikyti „LEGO MINDSTORMS“ robotų valdymui.
- Blokelių dėjimų / skaitomumo tvarka – kokia tvarka reikia sudėti ir skaityti vizualios programavimo kalbos elementus, jog nurodytų vykdymo eiliškumą.

- Modeliavimo aplinka – ar sistema turi modeliavimo aplinką, kurioje būtų galima išbandyti parašyto kodo veikimą.
- Kaina – kokia vizualios programinės sistemos kaina.

2.4 lentelė. Vizualių programinių sistemų palyginimas

Lyginimo kriterijai	**„Scratch“**	**NXT-G**	**„Microsoft Robotics Developer Studio“**
Pritaikyta tik robotų valdymui	-	+	+
Režimas pažengusiems	-	-	+
Programinio kodo generavimas	-	-	+
Vartotojo sąsaja turi lietuvių kalbą	+	-	-
Pritaikyta „LEGO MINDSTORMS“ robotams	+	+	+
Blokelių dėjimų / skaitomumo tvarka	Iš viršaus į apačią	Gali būti bet kokia (priklauso nuo nubrėžtos sijos)	Gali būti bet kokia (priklauso nuo jungiančių linijų)
Modeliavimo aplinka	+	-	+
Kaina	Nemokamas	Nemokamas	Nemokamas

2.4. Įgyvendinimo problemos

Kuriant naują vizualią programavimo kalbą, ji turi atitikti programavimo logiką, jog būtų galima išmokti pačio programavimo principo. Iš viso to gauname pagrindines įgyvendinimo problemas kurias turime spręsti kuriant vizualią programavimo kalbą. Pagrindinė problema vizualios programavimo kalbos mazgų žymėjimo dizainas, kuris turi nusakyti aiškią paskirtį, būti lengvai suprantamas bei prisitaikyti prie daromų pakeitimų.

Vizualus programavimo kalbos mazgų žymėjimai turi būti kuo efektyviau optimizuoti pagal žmogaus psichologiją. Vizualūs žymėjimai yra unikalūs į žmogų orientuoti pateikimai, kurių vienintelis tikslas yra palengvinti žmogaus supratimą ir išspręsti problemą [11]. Šie žymėjimai turi ilgą istoriją, prasidedančią nuo programų schemų iki šių dienų UML notacijos. Tačiau net po tiek daug laiko projektavimo žymėjimams trūksta mokslinio pagrindo. Šiuo metu vertinant, lyginant ir kuriant vizualinius žymėjimus pagal D. Moody tenka pasikliauti intuicija ir nykščio taisykle: „Mes neturime nei teorijos, nei sistemingos empirinės dalies įrodymų, jog galėtume jais vadovautis“ [12]. Todėl sunku nusakyti koks turi būti žymėjimas, jog vartotojai lengvai jį suprastų.

Vizuali programavimo kalba turi išlaikyti pačio vizualaus programinio kodo skaitomumą atsiradus daugiau skirtingo funkcionalumo mazgų. Šiuo metu su tokia problema susiduria visos anksčiau minėtos vizualios programavimo kalbos, todėl reikia iš anksto į tai atsižvelgti prieš kuriant naują vizualią programavimo kalbą. Dėl šios priežasties reikia numatyti kokius veiksmus galės atlikti

robotas be standartinio judėjimo ir atstumo nustatymo iki kliūties. Galbūt dalis roboto veiksmų yra išties unikalūs ir tai turėtų atsispindėti vizualioje programavimo kalboje. Užtikrinti tokiam funkcionalumui reikėtų įgyvendinti nestandartinį unikalų sprendimą, kurį reikėtų numatyti prieš kuriant visą vizualią programavimo kalbą. Kitu atveju rizikuojama, jog vizualios programavimo kalbos skaitomumas taps mažas. Tai susiję ir su tuo, jog vizuali programavimo kalba neturi nukentėti, jeigu ateityje bus norima pridėti naujų roboto valdymo elementų.

2.5. Sprendimo būdai

Vizualios programavimo kalbos elementai gali būti pateikiami įvairiai ir skirtingai, todėl naujai kuriama vizuali programavimo kalba, skirta robotų valdymui, turi stengtis atitikti šiuos rekomenduotino pobūdžio reikalavimus, kurie suformuoti remiantis mokslininkų tyrimais ir padarytomis išvadomis:

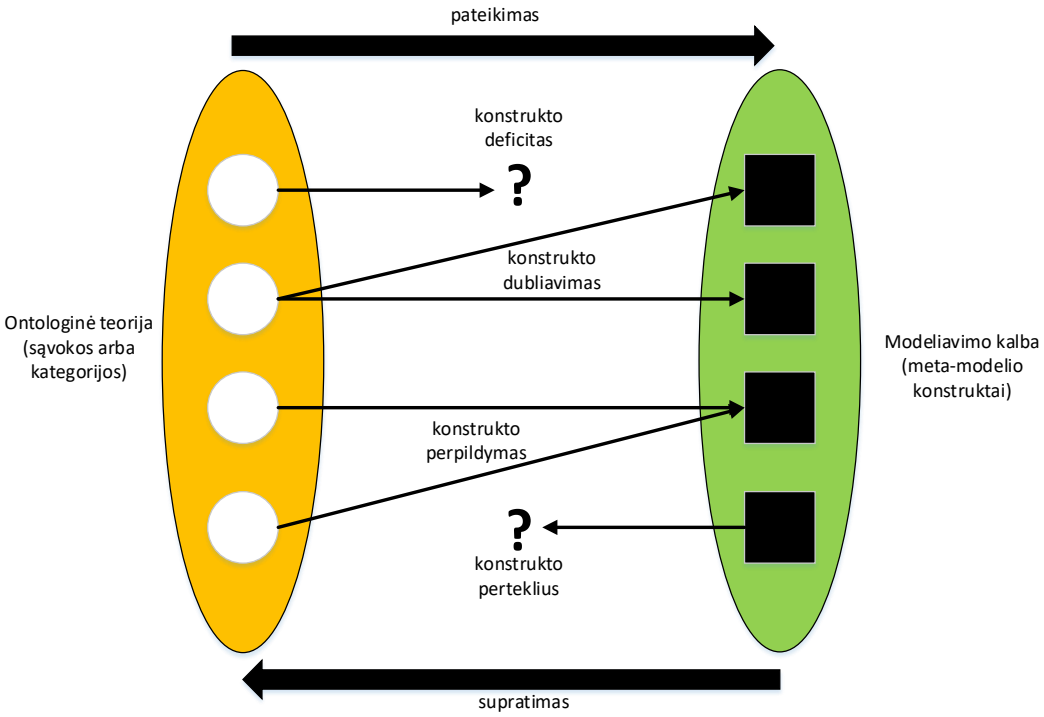
- Visi vizualios programavimo kalbos ženklų žymėjimai turi būti sugrupuoti pagal panaudojimą. Tokiu būdu išvengiama ilgo tam tikro elemento paieškai skirto laiko. Visgi tokie žymėjimai turi nusakyti kur kas daugiau nei paprastai nusakoma keliais žodžiais. Tokiu būdu elementai galės būti gana universalūs [13].
- Vizuali programavimo kalba turi palaikyti pagrindinius struktūros elementus: apdorojimą (*angl. process*), sprendimą (*angl. decision*), duomenų įvedimą ir išvedimą, bei galimybe juos sujungti. Taip pat reikėtų palaikyti kintamųjų ir ciklo galimybę [14].
- Vizuali programavimo kalba turi turėti klasikinį roboto elementų įgyvendinimą, tokį kaip judėjimą, teksto rašymą, atstumą iki kliūties nurodantį jutiklį. Visos analizuotos vizualios programavimo kalbos, kurios valdo robotus, turi specializuotus elementus. Šie elementai atsakingi už tam tikrą robotų elgesį ar grąžinimą gautų duomenų iš roboto aplinkos. Visgi reikia atskirti specializuotus roboto valdymo elementus nuo pagrindinių struktūros elementų. Taip ateityje bus užtikrinta lengvesnis naujų robotui skirtų specializuotų elementų pridėjimas.
- Vizualios programavimo kalbos elementai turi būti lengvai perkeliami iš vienos vietos į kitą, ištrinami, sukuriami nauji ar redaguojamos elementų reikšmės. Tokį sprendimą „Stagecast“ kūrėjai pristatė pirmame komerciniame produkte 1999 metais paremta PBD (*angl. Programming By Demonstration)* technologija, kuri leido vaikams kurti savo interaktyvias istorijas, žaidimus ir modeliavimus. Tokių elementų vykdomos operacijos pavaizduotos ir aprašytos 2.5 lentelėje [15].

2.5 lentelė. Galimų elemento operacijų pademonstravimas

Operacija	**Ką daro vartotojas**	**Ką užfiksuoja kompiuteris**
Perkelti	Vilkti objektą su pelyte	Perkelti <objektą> į <vietą>
Kurti	Vilkti objektą su pelyte	Kurti <objektą> į <vietą>
Ištrinti	Pasirinkti objektą jį paspaudžiant ir paspausti ištrynimo klavišą	Ištrinti <objektą>
Nustatyki reikšmę	Dukart paspausti ant objekto rodomos reikšmės ir nustatyti naują reikšmę	Nustatyti <reikšmę> į <objektą> <reikšmė>

2.5.1. Remiantis ontologine analize

Ontologinė analizė tapo plačiai naudojama vertinant programinės įrangos žymėjimus inžinerijos būdu [16]. Turėtų būti 1:1 sąvokų atitikimas tarp ontologijos teorijos ir kalbos konstruktų. Priešingu atveju egzistuos viena ar daugiau 2.6 paveikslėlyje parodytos problemos.



2.6 pav. Ontologinė analizė

- Konstrukto deficitas egzistuoja kai kalbos konstruktą (žymėjimą) neatitinka tam tikros ontologinės sąvokos.
- Konstrukto perpildymas egzistuoja kai vienas kalbos konstruktas (žymėjimas) gali reikšti kelias ontologines sąvokas.
- Konstrukto perpildymas egzistuoja kai kelių kalbų konstruktai (žymėjimai) naudojami pateikti vieną ontologinę sąvoką.
- Konstrukto perteklius egzistuoja kai kalbos konstruktas (žymėjimas) neatitinka nei vienos ontologinės sąvokos.

Jei egzistuoja konstrukto deficitas, tai žymėjimai ontologiškai neišsamūs. Jei nors vienas iš kitų trijų anomalijų egzistuoja, tai žymėjimai ontologiškai neaiškūs. Tokia ontologinė analizė anksčiau buvo taikomą įvertinti UML semantiką. Tačiau ši analizė ignoruoja vaizdinius aspektus [17].

2.6. Analizės išvados

Analizės metu gautos tokios išvados:

1. Apžvelgus vizualių programavimo kalbų pritaikymo galimybes nustatyta, jog yra kalbų, kurios skirtos edukaciniams tikslams mokyti vaikus. Šios kalbos būna labiau supaprastintos, todėl praranda dalį savo galimo funkcionalumo kaip programinio kodo generavimas. Taip pažengę programuoti vartotojai praranda galimybę, kurią galėtų naudoti, norint greičiau prisitaikyti ir išmokti skaityti, bei rašyti programinį kodą.
2. Dauguma apžvelgtų vizualaus programavimo aplinkų pritaikytos tik tam tikroms specifinėms užduotims atlikti. Dėl šios priežasties nuspręsta kurti specialiai pritaikytą vizualią programavimo kalbą, kuri modeliavimo aplinkoje galėtų užtikrinti robotą su įvairiu funkcionalumu.
3. Vizualios programavimo kalbos pagrindinė problema – kaip vartotojui pavaizduoti elementus, jog būtų aiškūs ir suprantami. Kiekvienas elementas turi padėti vartotojui suprasti ir išspręsti problemą. Visgi, jeigu kalba sudaryta iš daug elementų, tai apsunkina tinkamo elemento pasirinkimą. Tokių elementų turinčių panašius veiksmus reikėtų apjungti tarpusavyje, jog vartotojas negalėtų pasirinkti netinkamo elemento.
4. Analizuojant vizualias programavimo kalbas pastebėta, jog atsiranda problema realizuojant didesnius algoritmus. Tampa sunkiai skaitoma ir suprantama visa tokia atvaizduojama vizualios programavimo kalbos struktūra. Dėl šios priežasties reikia numatyti galimybę kaip būtų galima apibendrinti vizualios programavimo kalbos mazgus, jog būtų sumažintas mazgų kiekis sistemos pateiktame ekrane.
5. Pateikti sprendimo pasiūlymai naujai vizualiai programavimo kalbai leidžia numatyti pačius svarbiausius programavimui skirtus elementus (apdorojimas, sprendimas, duomenų įvedimas ir išvedimas, bei galimybe juos sujungti). Naujų specifinių elementų įgyvendinimas yra tolimesnių tyrimų tikslas.

3. REIKALAVIMŲ SPECIFIKACIJA IR ANALIZĖ

3.1. Vartotojų analizė

3.1.1. Vartotojų charakteristikos

Kuriamai sistemai ir jos posistemėms nustatytos pagrindinės vartotojo charakteristikos:

1. Orientuojamasi į mokyklinio amžiaus vartotojo grupę (5-12 kl.);
2. Neturintis jokių arba turintis tik minimalų suvokimą apie programavimą;
3. Neturintis jokių arba turintis tik minimalias algoritmų sudarymo žinias;
4. Žemo lygio kompiuterinio raštingumo žinios.

3.1.2. Vartotojo problemos

Vartotojams, neturintiems patirties programavimo srityje, įprastinė tekstinė programavimo sintaksė ir įprastiniai programavimo įrankiai nėra tokie intuityvūs kaip grafiniai. Egzistuoja labai daug ir įvairios literatūros programavimo tematika, tačiau šiuolaikinis vartotojas pirmiausia nori išbandyti ir pamatyti ką gali padaryti su programavimu, nei skaityti teorinius pagrindus iš knygų. Dėl šios priežasties gali būti sunku sudominti vartotojus šia mokslo sritimi.

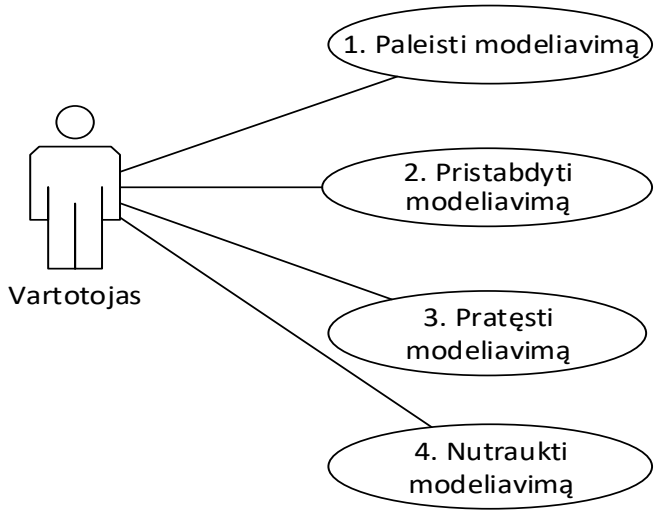
Kuriamas produktas sieks išspręsti šias problemas tokiais būdais:

1. Pateikiant paprastą, aiškią ir lengvai suprantamą vartotojo sąsają;
2. Paslepiant programinio kodo rašymą po vizualios programavimo kalbos mazgais;
3. Pateikiant standartinius robotų valdymo algoritmų užrašymus;
4. Pateikiant programavimą kaip žaidimą.

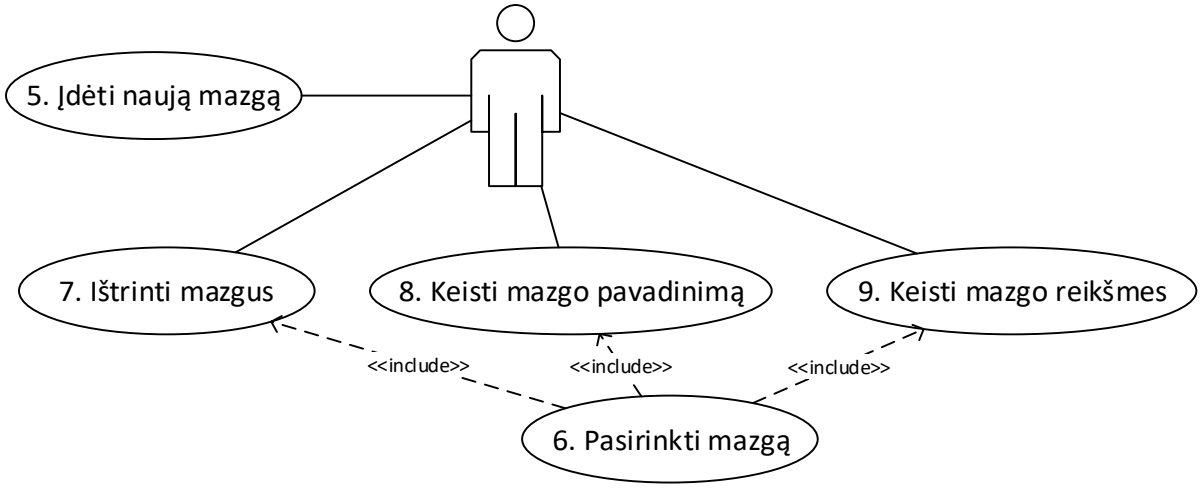
Vizualaus programavimo posistemė sprendžia problemą kaip pateikti paprastą, aiškią ir lengvai suprantamą vartotojo sąsają, jog paslepiant programinio kodo rašymą po vizualios programavimo kalbos mazgais būtų galima įgyvendinti įvairius standartinius ir ne tik roboto valdymo algoritmus.

3.2. Sistemos panaudos atvejų diagrama

3.1 ir 3.2 paveikslėliuose pateiktos sistemos panaudos atvejų diagramos, kurios nusako pagrindines sistemos funkcijas. Išsamiau panaudos atvejus aprašo pateikta 3.1 panaudos atvejų lentelė. Grafinės programavimo kalbos sąsajos posistemės išsamesnis aprašymas pateikiamas 3.2 lentelėje *Volere* šablonais.



3.1 pav. Robotų modeliavimo aplinkos posistemės panaudos atvejų diagrama



3.2 pav. Grafinės programavimo kalbos vartotojo sąsajos 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 vizualios programavimo kalbos kodą.
2	Pristabdyti modeliavimą	Vartotojas	Laikinai pristabdomas roboto modeliavimas.
3	Pratęsti modeliavimą	Vartotojas	Pratęsiamas laikinai sustabdytas modeliavimas su vizualios programavimo kalbos kodo pakeitimais, jeigu tokie buvo atlikti.
4	Nutraukti modeliavimą	Vartotojas	Nutraukiamas roboto modeliavimas.
5	Įdėti naują mazgą	Vartotojas	Į vizualią programavimo kalbos sąsają įdedamas elementas, atitinkantis vienetinį programavimo kalbos funkcionalumą.
6	Pasirinkti mazgą	Vartotojas	Vizualioje programavimo kalbos sąsajoje pasirenkamas norimas jos mazgas.
7	Ištrinti mazgus	Vartotojas	Ištrinami iš vizualios programavimo kalbos sąsajos pasirinkti mazgai.
8	Keisti mazgo pavadinimą	Vartotojas	Pakeičiamas pažymėto mazgo pavadinimas.
9	Keisti mazgo reikšmes	Vartotojas	Pakeičiamos pažymėto mazgo reikšmės.

3.2 lentelė. Sistemos panaudos atvejai, aprašyti *Volere* šablonais

PA „5 Įdėti naują mazgą“	
Aktorius	Vartotojas
Aprašas	Įdeda naują mazgą į vizualios programavimo kalbos sąsają.
Prieš sąlyga	Vartotojas paspaudė ant pasirinktos rūšies mazgo.
Sužadinimo sąlyga	Vartotojas nori įdėti naują mazgą.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia ant pasirinktos rūšies mazgo.	1.1. Sistema sukuria pasirinktos rūšies mazgą vizualioje programavimo kalbos sąsajoje.
2. Baigiamas PA.	
Po sąlyga	Sukuriamas pasirinktos rūšies mazgas vizualioje programavimo kalbos sąsajoje.

PA „6 Pasirinkti mazgą“	
Aktorius	Vartotojas
Aprašas	Pažymimas tik tas mazgas ant kurio paspaudžiama.
Prieš sąlyga	Vartotojas paspaudė ant vizualios programavimo kalbos mazgo.
Sužadinimo sąlyga	Vartotojas nori pažymėti mazgą.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia ant norimo mazgo.	1.1. Sistema pažymi vartotojo pasirinktą mazgą.
2. Baigiamas PA.	
Po sąlyga	Pažymimas vartotojo pasirinktas mazgas.

PA „7 Ištrinti mazgus“	
Aktorius	Vartotojas
Aprašas	Ištrina pasirinktus mazgus iš vizualios programavimo kalbos sąsajos.
Prieš sąlyga	Vartotojas turi būti pasirinkęs norimus mazgus.
Sužadinimo sąlyga	Vartotojas nori ištrinti mazgus.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia ant norimo ištrinti mazgo.	1.1. Sistema pažymi vartotojo pasirinktą mazgą ir po jo einančius mazgus vizualioje programavimo kalbos sąsajoje, jeigu tokių yra.
2. Vartotojas patvirtina mazgų ištrynimą.	2.1. Sistema ištrina pažymėtus mazgus iš vizualios programavimo kalbos sąsajos.
3. Baigiamas PA.	
Po sąlyga	Ištrinami vartotojo pasirinkti mazgai iš vizualios programavimo kalbos sąsajos.

PA „8 Keisti mazgo pavadinimą“	
Aktorius	Vartotojas
Aprašas	Pakeičiamas mazgo pavadinimas.
Prieš sąlyga	Vartotojas paspaudė ant mazgo pavadinimo.
Sužadinimo sąlyga	Vartotojas nori duoti mazgui kitokį pavadinimą.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia ant norimo mazgo pavadinimo.	1.1. Sistema įjungia redagavimo režimą mazgo pavadinimui.
2. Vartotojas įveda mazgo naują pavadinimą.	2.1. Sistema išsaugo mazgo naują pavadinimą.
3. Baigiamas PA.	
Po sąlyga	Pakeičiamas mazgo pavadinimas.
Klaidingas veikimas	Pavadinimas nepakinta, jei mazgas tokiu pavadinimu jau egzistuoja.

PA „9 Keisti mazgo reikšmes“	
Aktorius	Vartotojas
Aprašas	Leidžia pakeisti mazgo kintamąsias reikšmes.
Prieš sąlyga	Vartotojas paspaudė ant norimo mazgo keitimo reikšmės.
Sužadinimo sąlyga	Vartotojas nori pakeisti mazgo reikšmę ar reikšmes.
Pagrindinis įvykių srautas	**Sistemos reakcija ir sprendimai**
1. Vartotojas paspaudžia ant norimo mazgo reikšmės.	1.1. Sistema įjungia mazgo reikšmės redagavimo režimą.
2. Vartotojas pakeičia į norimą reikšmę.	2.1. Sistema išsaugo naują reikšmę.
3. Baigiamas PA.	
Po sąlyga	Pakeičiamos mazgo vidinės mazgų reikšmės.

3.2.1. Funkciniai reikalavimai

Reikalavimo Nr.	1
Aprašymas	
Įgyvendinti standartinius robotų algoritmus vizualioje programavimo aplinkoje kaip loginius programavimo mazgus.	
Atitikimo kriterijus	
Vizualios programavimo kalbos loginiais mazgais įgyvendintas bent vienas kliūčių išvengimo, kelio paieškos algoritmas, kurį galima įsidėti į savo robotą.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	5

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

Reikalavimo Nr.	3
Aprašymas	
Vizualios programavimo kalbos transformacija į C# programavimo kalbos kodą.	
Atitikimo kriterijus	
Vizualia programavimo kalba sukurti programų kodų algoritmai transformuojami į C# programavimo kalbos kodą.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	5

Reikalavimo Nr.	4
Aprašymas	
Galimybė peržiūrėti ir redaguoti C# programavimo kalbos kodą, gautą transformavus iš vizualios programavimo kalbos.	
Atitikimo kriterijus	
Vartotojo sukurtas vizualus programinis kodas transformuotas į C# programavimo kalbos kodą, kurį vartotojas gali peržiūrėti ir pakeisti pagal savo poreikius.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	4

Reikalavimo Nr.	5
Aprašymas	
Vizualios programavimo kalbos parašyto algoritmo išsaugojimas ir įkėlimas.	
Atitikimo kriterijus	
Vartotojo sukurtas vizualus programinis kodas išsaugomas su norimu pavadinimu ir gali būti vėl įkeltas į sistemą pakartotiniam panaudojimui. Išsaugotas vizualus programinis kodas neturi skirtis nuo vėl į sistemą įkelto vizualaus programinio kodo.	
Vartotojo patenkinimas	5
Vartotojo nepatenkinimas	4

3.3. Nefunkciniai reikalavimai

3.3.1. Reikalavimai sistemos išvaizdai

Grafinė vartotojo sąsaja turi būti patraukli vaikams. Šiam reikalavimui įgyvendinti turi būti įgyvendinti šie kriterijai: aplinka turi būti spalvota, naudoti įprastus šriftus. Šis kriterijus yra įgyvendintas, jeigu apklausus 50 vaikų iki 16 metų tiek spalvos, tiek šrifto pasirinkimą 90% vaikų įvertina teigiamai.

3.3.2. Reikalavimai panaudojamumui

3.3.2.1. Naudojimosi paprastumas

Vartotojas neturėdamas jokių specialių žinių turi sugebėti naudotis sistema. Sistemos sąsaja turi būti lengvai suprantama ir valdoma tiek liečiamo ekrano, tiek kompiuterinės pelės. Šis kriterijus laikomas įgyvendintu, jeigu 47 iš 50 vartotojų, mokančių naudotis sistema su kompiuterine pele, gali be jokios dokumentacijos naudotis sistema su liečiamu ekranu.

3.3.2.2. Mokymosi reikalavimai

Vartotojai, neturėdami jokių specialių programavimo žinių, turėtų sugebėti lengvai naudotis sistema. Vartotojui, nesuprantant kokios nors vizualios programavimo kalbos funkcijos, turi būti leista peržiūrėti to elemento išsami informacija su panaudojimo pavyzdžiais. Šis kriterijus laikomas įgyvendintu, jeigu iš 50 bandančių vartotojų bent 44 vartotojai sugeba atlikti pagrindinius sistemos veiksmus (įdėti naują mazgą, pervadinti mazgą, pakeisti jos reikšmę) ir suprasti ką reiškia kiekvienas toks mazgas. Vartotojai, nesusipažinę su pateiktu to elemento panaudojimo pavyzdžiu ir aprašymu, nėra įtraukiami į skaičiavimą.

3.3.2.3. Suprantamumas ir mandagumas

Vizualios programavimo kalbos suprantamumas ir mandagumas siekiamas tokiais būdais:

- Vartotojas paspaudęs ant vizualios programavimo kalbos mazgo redaguojamos vietos, kaip pavadinimas ar mazgo reikšmės, gali jį redaguoti;
- Vartotojui, keičiant tam tikrą mazgo reikšmę, neturi būti suteikta teisė įvesti to kas pakenktų sistemai. Vartotojui neleidžiama įvesti raidžių skaitinės reikšmės reikalaujančiame laukelyje;
- Vartotojo reikšmių įvedimo palengvinimui yra apribotas jų pasirinkimas. Toks apribotas pasirinkimas yra spalvos pavadinimo, aritmetinio ar palyginimo ženklo, tam tikro tipo keitimas iš duoto sąrašo. Vartotojui reikia pasirinkti norimą reikšmę iš duotų, negalvojant kokia reikšmė galima ar sistema ją tinkamai supras.

3.3.3. Reikalavimai vykdymo charakteristikoms

Grafinio programavimo posistemė turi palaikyti bent 5000 vizualios programavimo kalbos elementų. Šis reikalavimas bus įgyvendintas, jeigu sistema turėdama iki 5000 sujungtų mazgų vienoje programoje nesulėtės ar neužstrigs.

3.3.4. Reikalavimai veikimo sąlygoms

3.3.4.1. Numatoma fizinė aplinka

Programinė įranga bus naudojama vidutinio triukšmo aplinkoje, kurioje galimai daug vartotoją galinčių išblaškyti objektų.

3.3.4.2. Reikalavimai darbui su gretimomis sistemomis

Sistema dirbs kartu su „Visual Studio“ programos siūlomu .NET kompiliatoriumi. Šiuo kompiliatoriumi sistema naudosis norint sukompiliuoti C# programinį kodą, kurį sistema gauna transformavus vizualią programavimo kalbą. Programinis projekto kodas kompiliatoriui perduodamas per komandinę eilutę.

3.3.4.3. Reikalavimai sistemos platinimo/gamybos formatui

Produktas bus platinamas dviem būdais:

- Atviro kodo programa, kurią vartotojas gali pats sukompiliuoti ir naudotis.
- Vykdomasis diegimo failas.

Abiem atvejais produktas bus platinamas internete be jokio mokesčio.

3.3.4.4. Reikalavimai sistemos priežiūrai

Planuojamas produkto plėtojimas kartu su bendruomenės pagalba. Populiariausi pakeitimai ar klaidų pataisymai atlikti bendruomenės integruojami į pagrindinę programos versiją.

3.3.5. Reikalavimai sistemos priežiūrai

3.3.5.1. Sistemos aptarnavimas

Sistemos aptarnavimui nėra keliami konkretūs reikalavimai. Kadangi tai atviro kodo programinė įranga, tai modifikuoti gali bet kuris vartotojas. Papildomas sistemos aptarnavimas vykdomas tik pagal specializuotas sutartis.

3.3.5.2. Sistemos palaikymas

Sistemos palaikymui bus sukurtas bendruomenės internetinis puslapis. Jame vartotojai galės vieni kitiems suteikti reikalingą pagalbą ir gauti atsakymus į iškilusius klausimus.

3.3.5.3. Perkėlimo į kitas platformas reikalavimai

Pirmą produkto versiją planuojama paleisti tik „Windows“ operacinėse sistemose. Visgi ateityje gali atsirasti poreikis sistemą perkelti į kitas operacines sistemas, kurias palaiko „Unity“ žaidimų variklis.

3.3.6. Įstatyminiai reikalavimai

Sistema turi laikytis Lietuvos Respublikos įstatymų reikalavimams. Visas teises į sistemą įgauna jos užsakovas, kai sistemos kūrimas užbaigiamas. Programinė įranga kuriama atviro kodo ir apsaugota GNU GPL V3 standartu.

3.4. Perspektyviniai reikalavimai

Ateityje gali prireikti:

- Sukurti naujus mazgus su nauju funkcionalumu ir įtraukti į vizualią programavimo kalbą;
- Vizualioje programavimo kalboje sukurtą programinį kodą perkelti į tikrus robotus;
- Išversti sistemą į kitas kalbas.

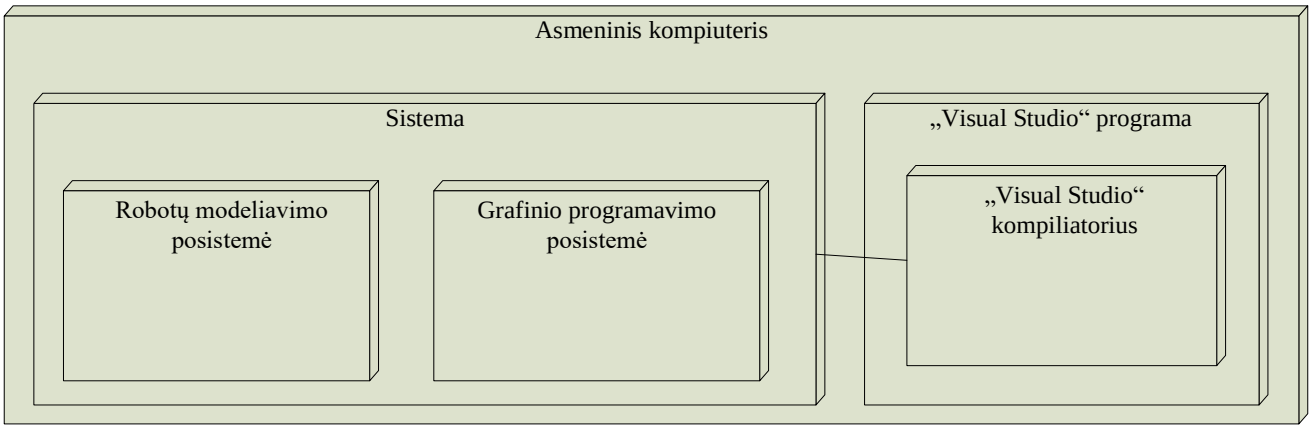
3.5. Apribojimai reikalavimams

3.5.1. 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 namuose ir mokymo įstaigose yra „Windows“.
Atlikimo kriterijus:	Sistema veikia „Windows 7“, „Windows 8“, „Windows 8.1“ ir „Windows 10“ operacinėse aplinkose.
Reikalavimas:	Sistemai reikalingas sukurtas 3D variklis.
Aprašymas:	3D variklio kūrimas reikalauja daug laiko ir pastangų. Siekiant sutaupyti lėšas ir sumažinti laiko sąnaudas reikalingas sukurtas 3D variklis, veikiantis „Windows“ aplinkoje.
Atlikimo kriterijus:	Pasirinktas „Unity 5“ žaidimų variklis, atitinkantis specifikacijai keliamus reikalavimus.
Reikalavimas:	Vizualios programavimo kalbos universalumas.
Aprašymas:	Vizuali programavimo kalba turi būti universali. Šiam tikslui pasiekti turi būti transformuojama į imperatyvios programavimo kalbos kodą.
Atlikimo kriterijus:	Vizuali programavimo kalba transformuojama į C# programavimo kalbos kodą.

3.5.2. Diegimo aplinka

Išdėstymo (*angl. deployment*) diagrama pateikta 3.3 paveikslėlyje.



3.3 pav. Išdėstymo (*angl. deployment*) diagrama

„Visual Studio“ kompiliatorius – kliento kompiuteryje turi būti įdiegta „Visual Studio 2010“ (arba naujesnė) programa su kartu pridedamu C# programavimo kalbos palaikomu kompiliatoriumi.

Sukurta sistema susideda iš dviejų pagrindinių dalių:

- Robotų modeliavimo posistemė – tai virtuali aplinka, kurioje atvaizduojamas robotų elgesys pagal užrašytą algoritmą.
- Grafinio programavimo posistemė – tai vizualios programavimo kalbos posistemė, kurioje vartotojas naudojant vizualios programavimo kalbos mazgais gali užrašyti roboto elgsenos algoritmą.

3.5.3. Komunikuojančios sistemos

Kuriamai sistemai reikalinga transformacija iš sukurtos vizualios programavimo kalbos kodo į vykdomąjį kodą. Kadangi sistema transformuos kodą į C# programavimo kalbos kodą reikalingas ir atitinkamas kompiliatorius. Pasirinktas naudoti 3D „Unity“ žaidimų variklis turi savo .NET analogą vadinama „Mono“. Jis palaiko .NET asemblerį (*angl. assembly*) iki 2.0 versijos. Dėl šios priežasties bus naudojamas į „Visual Studio“ programą įeinantis .NET kompiliatorius.

3.5.4. Prieinama specializuota programinė įranga

Sistemai sukurti reikalingas grafinis 3D variklis. Egzistuoja įvairių siūlomų sprendimų šiam keliamam reikalavimui išspręsti. Šiam reikalavimui išpildyti pasirinktas „Unity 5“ žaidimų variklis. Pasirinkta specializuota grafinio 3D „Unity 5“ variklio nemokama programinės įrangos versija, leidžiama naudoti komerciniams, asmeniniams ir moksliniams tikslams.

3.5.5. Numatoma darbo vietos aplinka

Sistema kuriama asmeniniams kompiuteriams ir orientuojamasi į dvi tikslines vartotojų auditorijas:

- Mokiniai – šių vartotojų darbo vieta yra kompiuterių klasės, kuriose vienu kompiuteriu naudojasi vienas ar du mokiniai.
- Namų vartotojai – tai tokie vartotojai, kurie naudojasi programos versija skirta namams. Tokių vartotojų darbo aplinka bus jų namai su asmeniniais kompiuteriais.

4. SISTEMOS PROJEKTAS

4.1. Sistemos pagrindimas ir esmės išdėstymas

Siekiama, jog „Vizualios robotų programavimo ir robotų sąveikos modeliavimo aplinkos“ sistemos architektūra būtų lengvai išplečiama. Dėl šios priežasties architektūra sudaryta iš komponentų. Sistemos komponentai nėra tarpusavyje stipriai sujungti, juos galima lengvai pakeisti. Sistema suskirstyta į tris pagrindines posistemes:

1. Vizualios programavimo kalbos;
2. Vizualaus kodo transformavimo;
3. Robotų modeliavimo.

Šie komponentai veikia kaip atskiros paprogramės. Juos lengva pakeisti kitais, nerizikuojant pažeisti kitų posistemių integralumo. Tokį architektūrinį pasirinkimą įtakojo „Visual Studio“ programos kartu su C# kompiliatoriumi licencijavimas. Dėl šios priežasties negalima jo įdėti į kuriamą sistemą. Ši problema išsprendžiama kompiliavimo posistemėje, tai izoliuota paprogramė, kuri naudojama kviečiant kompiliavimo komandą.

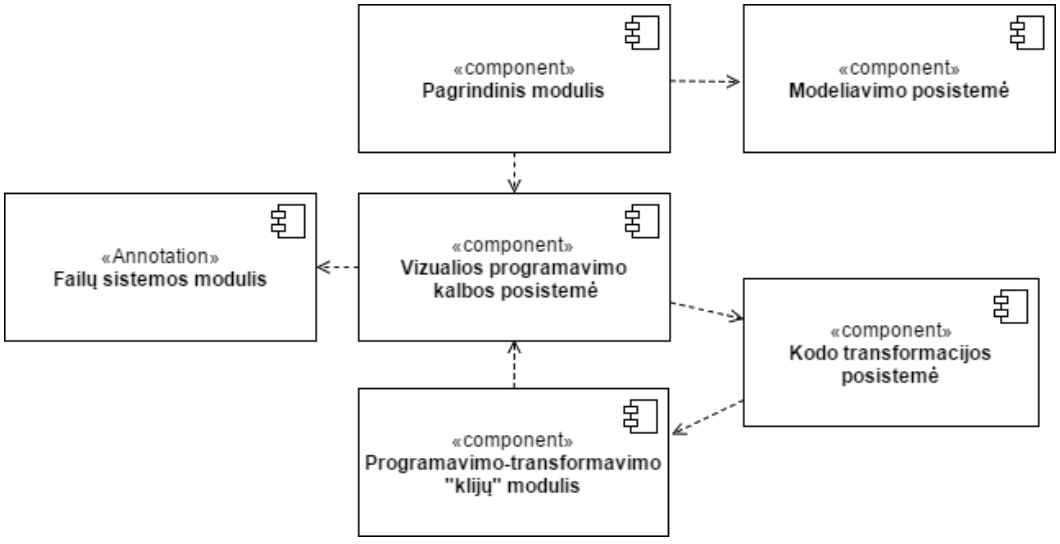
Transformavimo posistemei parinkta abstraktaus sintaksės medžio struktūra siekiant padidinti išplečiamumo galimybes ateityje, kadangi toks architektūrinis sprendimas leidžia vykdyti atvirkštinę kodo transformaciją ar transformaciją į kitas programines kalbas.

4.2. Sistemos architektūra

Kuriama sistema turi tris tarpusavyje sąveikaujančias posistemes:

- Vizualios robotų programavimo kalbos posistemė;
- Robotų modeliavimo posistemė;
- Mazgų transformavimo į C# programinį kodą posistemė.

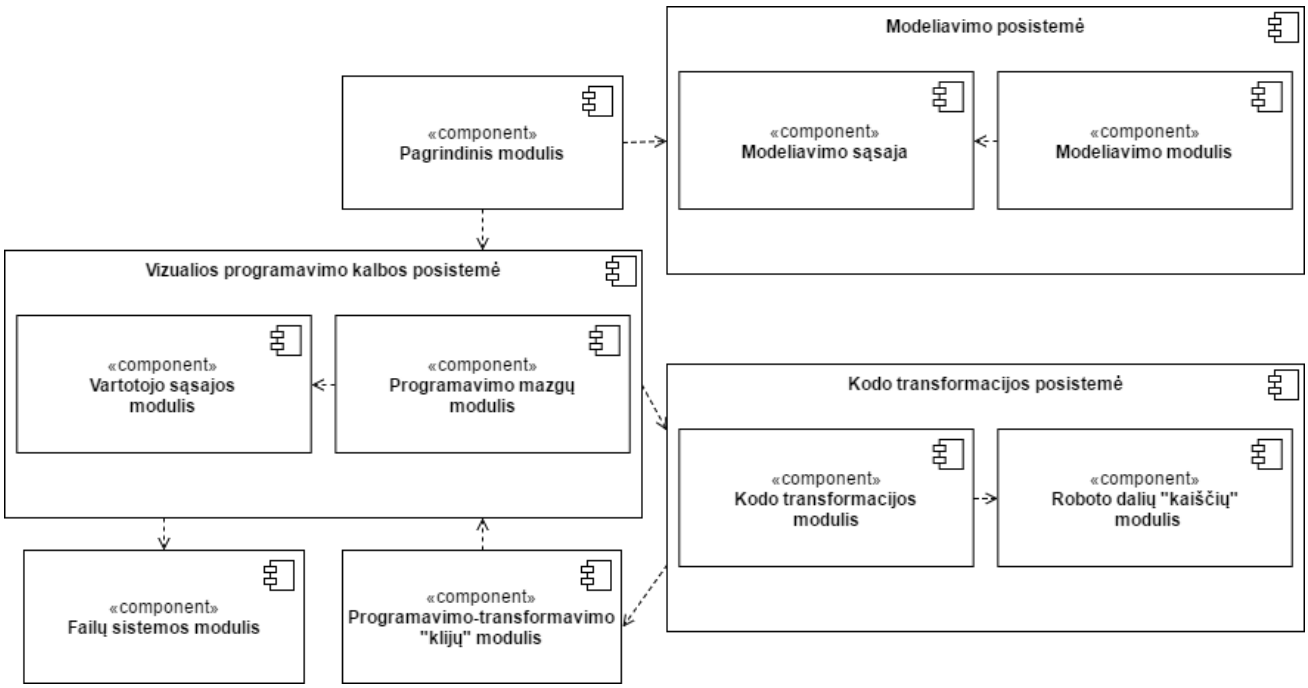
Komponentų sąveikos diagrama pateikiama 4.1 diagramoje.



4.1 pav. Komponentų sąveikos diagrama

4.3. Detalus projektas

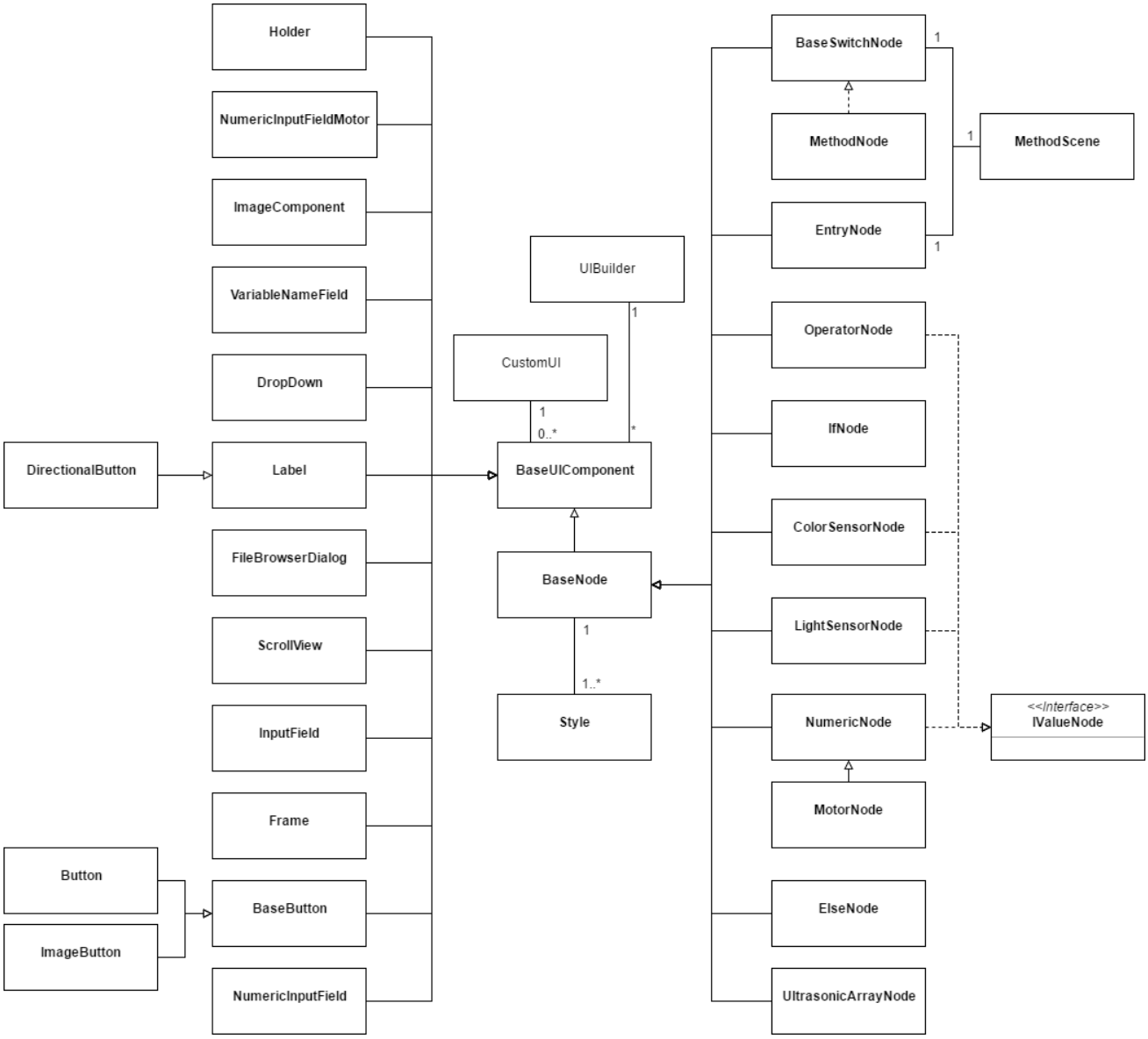
Detalizuota visos sistemos komponentų sąveikos diagrama pateikiama 4.2 paveikslėlyje.



4.2 pav. Detalizuoti visos sistemos komponentų sąveikos diagrama

Komponentas „Robotų modeliavimo posistemė“ ir jo sąsaja atsakinga už robotų elgsenos, aprašytos vizualia robotų programavimo kalba, modeliavimą virtualioje erdvėje. Šiai posistemei priklauso bendrinė sąsaja, kuri sistemoje dinamiškai paleidžia vartotojo sukurtus algoritmus vizualia programavimo kalba. Norint paleisti šiuos algoritmus, jie turi būti transformuoti į vykdomus robotų failus.

Komponentas „Vizualios programavimo kalbos posistemė“ atsakingas už vizualios robotų programavimo kalbos sintaksės vykdymą ir vartotojo sąsajos atvaizdavimą ekrane. Šios posistemės klasių diagrama pateikiama 4.3 paveikslėlyje, o jos duomenų žodynas pateikiamas 4.1 lentelėje.



4.3 pav. Vizualios programavimo kalbos posistemės klasių diagrama

4.1 lentelė. Vizualios programavimo kalbos posistemės duomenų žodynas

Pavadinimas	Vardų erdvė	Aprašymas	Tipas
MethodScene	RobotDesigner	Metodo sceną aprašanti klasė.	Klasė
BaseNode	RobotDesigner.Nodes	Mazgo elementą aprašanti bazinė klasė.	Abstrakti klasė
BaseSwitchNode	RobotDesigner.Nodes	Elementų, turinčių vidinę sceną, bazinė klasė. Pavyzdžiui: metodo mazgas, bet kuri sąlygos sakinio šaka.	Abstrakti klasė
ColorSensorNode	RobotDesigner.Nodes	Spalvų jutiklio mazgą aprašanti klasė.	Klasė
ElseNode	RobotDesigner.Nodes	Sąlygos sakinio šakos mazgą, kai sąlyga nėra tenkinama, aprašanti klasė.	Klasė
EntryNode	RobotDesigner.Nodes	Pirmąjį scenos elementą aprašanti klasė. Tai įėjimo į sceną mazgas.	Klasė
IfNode	RobotDesigner.Nodes	Sąlygos sakinio mazgą, kai sąlyga yra tenkinama, aprašanti klasė.	Klasė
LightSensorNode	RobotDesigner.Nodes	Liuksmetro mazgą aprašanti klasė.	Klasė
MethodNode	RobotDesigner.Nodes	Metodo mazgą aprašanti klasė.	Klasė
MotorNode	RobotDesigner.Nodes	Variklio mazgą aprašanti klasė.	Klasė
NumericNode	RobotDesigner.Nodes	Klasė aprašanti kintamojo, įgaunančio skaitinę reikšmę, mazgą.	Klasė

Pavadinimas	Vardų erdvė	Aprašymas	Tipas
OperatorNode	RobotDesigner.Nodes	Operatoriaus mazgą aprašanti klasė.	Klasė
IValueNode	RobotDesigner.Nodes	Mazgą, įgyjantį reikšmę, aprašanti sąsaja.	Sąsaja
UltrasonicArrayNode	RobotDesigner.Nodes.Sensors.Arrays	Ultragarso jutiklių masyvo mazgą aprašanti klasė.	Klasė
CustomUI	RobotDesigner.Views	Vartotojo sąsajos vaizdo bazinė klasė.	Abstrakti klasė
Style	Designer	Mazgų išvaizdos bazinė klasė.	Abstrakti klasė
BaseUIComponent	UI	Vartotojo sąsajos elemento bazinė klasė.	Abstrakti klasė
UIBuilder	UI	Vartotojo sąsajos iš failo pertvarkanti klasė.	Klasė
DropDown	UI.Components	Vartotojo sąsajos elementą su nusileidžiančiu meniu aprašanti klasė.	Klasė
FileBrowserDialog	UI.Components	Vartotojo sąsajos elementą, skirtą failų sistemos valdymui, aprašanti klasė.	Klasė
Frame	UI.Components	Vartotojo sąsajos rėmelio elementą aprašanti klasė.	Klasė
Holder	UI.Components	Vartotojo sąsajos elementų grupę aprašanti klasė.	Klasė
ImageComponent	UI.Components	Paveikslo komponento elementą aprašanti klasė.	Klasė
Label	UI.Components	Neredaguojamo vartotojo sąsajos teksto elementą aprašanti klasė.	Klasė
ScrollView	UI.Components	Slenkančio vartotojo sąsajos vaizdo elementą aprašanti klasė.	Klasė
BaseButton	UI.Components.Input	Vartotojo sąsajos mygtuko elementą aprašanti bazinė klasė.	Abstrakti Klasė
Button	UI.Components.Input	Vartotojo sąsajos mygtuko elementą aprašanti klasė.	Klasė
DirectionalButton	UI.Components.Input	Vartotojo sąsajos mygtuko elementą, turintį kryptį, aprašanti bazinė klasė. Elementas naudojamas mazgų grandinės kūrimui.	Klasė
ImageButton	UI.Components.Input	Vartotojo sąsajos mygtuko elementą, turintį tekstūrą, aprašanti klasė.	Klasė
InputField	UI.Components.Input	Vartotojo sąsajos elementą su redaguojamu lauku aprašanti klasė.	Klasė
NumericInputField	UI.Components.Input	Vartotojo sąsajos elementą su redaguojamu lauku, leidžiantį įvesti tik skaitinę reikšmę, aprašanti klasė.	Klasė
NumericInputFieldMotor	UI.Components.Input	Vartotojo sąsajos elementą, leidžiantį įvesti tik taisyklingą variklio stiprumo reikšmę, aprašanti klasė.	Klasė
VariableNameField	UI.Components.Input	Vartotojo sąsajos elementą, leidžiantį įvesti tik galimą kintamojo pavadinimą, aprašanti klasė.	Klasė

Programavimo-transformavimo „Klijų“ modulio, atsakingo už duomenų pateikimą transformavimo posistemei, duomenų žodynas pateiktas 4.2 lentelėje.

4.2 lentelė. Programavimo-transformavimo „Klijų“ modulio duomenų žodynas

Pavadinimas	Vardų erdvė	Aprašymas	Tipas
RobotIO	DesignerGlue	Kodo transformavimo posistemei perduodama roboto komponentų informacija.	Klasė
SyntaxTree	DesignerGlue	Sintaksės medžio generavimui skirta klasė.	Klasė
BaseIfElseLink	DesginerGlue.Chain	Sąlygos sakinio mazgų naudojama bazinė klasė.	Abstrakti klasė
SyntaxChain	DesignerGlue.Chain	Kiekvieną grandinės elementą aprašanti klasė.	Klasė
SyntaxDoubleLink	DesignerGlue.Chain	Grandinės elementą, turintį dvi jungtis, aprašanti klasė.	Klasė
SyntaxLink	DesignerGlue.Chain	Grandinės elemento jungtį aprašanti klasė.	Klasė
ColorSensorLink	DesignerGlue.Chain.Types	Spalvų jutiklio grandinės jungtį aprašanti klasė.	Klasė
IfOtherwiseLink	DesignerGlue.Chain.Types	Grandinės jungtį sąlygos sakiniui, kai sąlyga nėra tenkinama, aprašanti klasė.	Klasė
IfSuccessLink	DesignerGlue.Chain.Types	Grandinės jungtį sąlygos sakiniui, kai sąlyga tenkinama, aprašanti klasė.	Klasė
LightSensorLink	DesignerGlue.Chain.Types	Liuksmetro grandinės jungtį aprašanti klasė.	Klasė
MethodLink	DesignerGlue.Chain.Types	Grandinės jungtį, skirtą apibūdinti metodą, aprašanti klasė.	Klasė
MotorLink	DesignerGlue.Chain.Types	Variklio grandinės jungtį aprašanti klasė.	Klasė
NumericLink	DesignerGlue.Chain.Types	Skaitinio kintamojo grandinės jungtį aprašanti klasė.	Klasė
OperatorLink	DesignerGlue.Chain.Types	Operatoriaus grandinės jungtį aprašanti klasė.	Klasė
UltrasonicArrayLink	DesignerGlue.Chain.Types	Ultragarso jutiklių masyvo grandinės jungtį aprašanti klasė.	Klasė
IVariableLink	DesignerGlue.Chain.Types	Sąsaja nurodanti, kad grandinės jungtis grąžina kintamojo reikšmę.	Sąsaja

Komponentas „Kodo transformavimo posistemė“ atsakingas už vizualios robotų programavimo kalbos sintaksės transformavimą į abstraktų sintaksės medį ir šio medžio transformavimą į C# programavimo kalbos kodą. Šios posistemės duomenų žodynas pateikiamas 4.3 lentelėje.

4.3 lentelė. Kodo transformavimo posistemės duomenų žodynas

Pavadinimas	Vardų erdvė	Aprašymas	Tipas
ISyntaxNode	-	Bazinė sintaksės vieneto sąsaja, kurią paveldi visi sintaksės vienetai.	Sąsaja
BaseMethod	-	Metodus generuojančius sintaksės vienetus apibūdinanti abstrakti klasė.	Abstrakti klasė
ComparatorNode	Comparators	Palyginimo sintaksinius vienetus aprašanti abstrakti klasė.	Abstrakti klasė
EQComparatorNode	Comparators	Sintaksės vienetas palyginantis ar du elementai lygūs.	Klasė
GTComparatorNode	Comparators	Sintaksės vienetas palyginantis ar vienas elementas yra didesnis už kitą.	Klasė
GTEComparatorNode	Comparators	Sintaksės vienetas palyginantis ar vienas elementas didesnis arba lygus už kitą.	Klasė

Pavadinimas	Vardų erdvė	Aprašymas	Tipas
LTComparatorNode	Comparators	Sintaksės vienetas palyginantis ar vienas elementas mažesnis už kitą.	Klasė
LTEComparatorNode	Comparators	Sintaksės vienetas palyginantis ar vienas elementas mažesnis arba lygus už kitą.	Klasė
NEQComparatorNode	Comparators	Sintaksės vienetas palyginantis ar vienas elementas nelygus kitam.	Klasė
InvokeMethod	Invocation	Sintaksės vienetų iškvietimo metodus aprašanti abstrakti klasė.	Abstrakti klasė
InvokeRegularMethod	Invocation	Sintaksės elementas iškviečiantis standartinį objekto metodą.	Klasė
InvoekGetterMethod	Invocation	Sintaksės vienetas iškviečiantis *getter* tipo objekto metodą.	Klasė
InvokeSetterMethod	Invocation	Sintaksės vienetas iškviečiantis *setter* tipo objekto metodą.	Klasė
InvokeStaticRegularMethod	Invocation	Sintaksės vienetas iškviečiantis standartinį statinį metodą.	Klasė
InvokeStaticGetterMethod	Invocation	Sintaksės vienetas iškviečiantis *getter* tipo statinį metodą.	Klasė
IfNode	Logical	*If-else* sąlygos sakinį aprašantis sintaksės vienetas.	Klasė
ForeachNode	Loops	*Foreach* ciklą aprašantis sintaksės vienetas.	Klasė
ClassNode	Main	Klasę aprašantis sintaksės vienetas.	Klasė
RobotNode	Main	Klasę su roboto specifiniais kintamaisiais ir metodais aprašantis sintaksės vienetas.	Klasė
EvalNode	Members	Sintaksės vienetas generuojantis nurodytą kodo fragmentą.	Klasė
MethodNode	Members	Metodą aprašantis sintaksės vienetas.	Klasė
ObjectNode	Members.Types	Objektą aprašantis sintaksės vienetas. Ši klasė taip pat bazė visiems kintamųjų tipams.	Klasė
IMotorNode	Members.Types	*IMotor* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
IPartNode	Members.Types	*IPart* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
IRobotNode	Members.Types	*IRobot* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
ISensorNode	Members.Types	*ISensor* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
ISensorArrayNode	Members.Types	*ISensorArray* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
ListNode	Members.Types	Sąrašo kintamojo tipą aprašantis sintaksės vienetas.	Klasė
LiteralNode	Members.Types	Tipą, kurio reikšmei netaikoma tipo konversija, aprašantis sintaksės vienetas.	Klasė
NumericNode	Members.Types	*Double* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
PartTypeNode	Members.Types	*PartType* kintamojo tipą aprašantis sintaksės vienetas.	Klasė
TextNode	Members.Types	S*tring* kintamojo tipą aprašantis sintaksės vienetas.	Klasė

Pavadinimas	Vardų erdvė	Aprašymas	Tipas
OperatorNode	Operators	Operatoriaus tipo sintaksės vienetus aprašanti bazinė klasė.	Abstrakti klasė
BinaryOperatorNode	Operators.Binary	Binarinių operatorių sintaksės vienetus aprašanti bazinė klasė.	Abstrakti klasė
AdditionNode	Operators.Binary	Binarinę sudėtį aprašantis sintaksės vienetas.	Klasė
AssignmentNode	Operators.Binary	Priskyrimo operatorių aprašantis sintaksės vienetas.	Klasė
DivisionNode	Operators.Binary	Binarinį dalybos operatorių aprašantis sintaksės vienetas.	Klasė
MultiplicationNode	Operators.Binary	Binarinį daugybos operatorių aprašantis sintaksės vienetas.	Klasė
SubtractionNode	Operators.Binary	Binarinį atimties operatorių aprašantis sintaksės vienetas.	Klasė
CodeGeneration	Generation	Klasė sugeneruojanti programinį kodą iš nurodyto sintaksės medžio.	Klasė
ProjectGeneration	Generation	Klasė sugeneruojanti papildomus projekto failus, kurie reikalingi kompiliavimo metu.	Klasė
ProjectInfo	Generation.DataStructures	Projekto informaciją aprašanti klasė.	Klasė
AssemblyInfo	Generations.DataStructures	Sukompiliuoto .NET asemblerio (*angl. assembly*) informaciją aprašanti klasė.	Klasė
Extensions	-	Klasių plėtinių rinkinys.	Klasė
Utilities	-	Pagalbinių metodų rinkinys.	Klasė
StringWriterWithEncoding	IO	Praplečiamas *StringWriter* klasės funkcionalumas, kad būtų palaikomas skirtingas teksto kodavimas.	Klasė
PlugNode	RobotVL.Plugs	Kaištį aprašanti abstrakti klasė. Kaiščiai naudojami kaip laikini elementai kodo generavime, kurie atitinka roboto dalių pakaitalą.	Abstrakti klasė
MotorPlugNode	RobotVL.Plugs.Parts	Kaiščio elementas varikliui.	Klasė
ColorSensorPlugNode	RobotVL.Plugs.Parts.Sensors	Kaiščio elementas spalvų jutikliui.	Klasė
LightSensorPlugNode	RobotVL.Plugs.Parts.Sensors	Kaiščio elementas liuksmetrui.	Klasė
UltrasonicSensorArrayPlugNode	RobotVL.Plugs.Parts.Sensors.Arrays	Kaiščio elementas ultragarso jutiklių masyvui.	Klasė

5. SISTEMOS REALIZACIJA

5.1. Realizacijos ir veikimo aprašymas

5.1.1. Apie sistemą

Vizualaus robotų programavimo ir robotų sąveikos modeliavimo aplinkos sistema suteikia galimybę vartotojui sukurti algoritmą riboto funkcionalumo vizualia programavimo kalba, kuria būtų galima valdyti robotą, jog atliktų jam paskirtas užduotis. Pateikiama visa bendra sistema, kuri leidžia specialiai šiai modeliavimo aplinkai aprašyti roboto elgseną, naudojantis supaprastinta roboto valdymui pritaikyta vizualia programavimo kalba.

Sukurta vizuali programavimo kalba leidžia atlikti pagrindinius programavimo veiksmus ir realizuoti esamus algoritmus, jog būtų galima ištirti jų veikimą. Visa padaryta programavimo logika atvaizduojama roboto judėjime virtualioje aplinkoje, kurioje galima rinktis roboto komponentus. Šie komponentai yra standartizuoti, todėl roboto funkcijos bus klasikinės kaip judėjimas, atstumo fiksavimas iki kliūties, spalvos atpažinimas, žodžio išvedimas ekrane. Visgi funkcijos bus ribotos, nes tam tikri virtualūs jutikliai teiks tik jiems būdingą informaciją. Remiantis gautais jutiklių duomenims galima įgyvendinti įvairius roboto elgsenos algoritmus.

5.1.2. Pagrindinės funkcijos

Vizualios programavimo sąsajos pagrindinės funkcijos:

- Sukurti pasirinktą mazgo tipą:
  - Variklis;
  - Ultragarso jutiklių masyvas;
  - Liuksmetro jutiklio duomenys;
  - Spalvos jutiklio duomenys;
  - Metodas;
  - Kintamasis;
  - Sąlygos sakinys;
  - Aritmetinis veiksmas.
- Keisti mazgų reikšmes;
- Ištrinti mazgus;
- Peržiūrėti mazgo panaudojimo pavyzdį.

Robotų modeliavimo posistemės pagrindinės funkcijos:

- Pradėti modeliavimą;
- Pristabdyti modeliavimą;
- Tęsti modeliavimą;
- Nutraukti modeliavimą;

- Dinaminis kodo įkrovimas bei atnaujinimas.

5.2. Vizualios programavimo sąsajos realizacija

Vizualios programavimo kalbos posistemė susideda iš dalių:

1. Programavimo mazgų modulis.
2. Vartotojo sąsajos modulis.

Programavimo mazgų modulis aprašo mazgų išvaizdą, jų atributus, įvedimo laukelius su tikrinimu ir nustato mazgų jungimosi tarpusavyje taisykles. Padedant vartotojo sąsajos moduliui vizualios programavimo kalbos mazgai atvaizduojami vartotojo sąsajoje. Taip pat tokius vizualios programavimo kalbos mazgus galima ne vien naujai sukurti, bet galima redaguoti jau sukurtų mazgų atributus ar iš vis ištrinti tokius mazgus.

Vizualios programavimo kalbos posistemė, užtikrinant tinkamą jos veikimą, tiesiogiai bendrauja su kitais moduliais:

1. Pagrindinis modulis.
2. Failų sistemos modulis.
3. Programavimo-transformavimo „klijų“ modulis.
4. Kodo transformacijos posistemė.

Pagrindinis modulis yra programos pradžioje paleidžiamas modulis. Šis modulis užkrauna vizualios programavimo kalbos posistemės vartotojo sąsajos elementus, todėl vartotojas gali matyti mazgus ir juos redaguoti sistemos vartotojo sąsajoje.

Failų sistemos modulis gali išsaugoti ir vėliau įkelti anksčiau išsaugotą vizualios programavimo kalbos parašytą algoritmą. Šis modulis vizualios programavimo kalbos struktūrą transformuoja į XML failą, kuriame išsaugomi mazgų pavadinimai, jų atributų reikšmės bei informacija kaip mazgai jungiasi vienas su kitu. Žinant tokią, vizualios programavimo kalbos aprašyto algoritmo, informaciją lengva pakartotinai įkelti ir panaudoti.

Programavimo-transformavimo „klijų“ modulis konvertuoja vizualios programavimo kalbos sukurtą kodą į struktūrą, kurią suprastų kodo transformavimo posistemė. Pirmiausia sugeneruojama mazgų grandinė iš vizualios programavimo kalbos pateiktų mazgų. Vėliau ši mazgų grandinė transformuojama į sintaksinį medį. Iš tokio sintaksinio medžio kodo transformavimo posistemė generuoja C# programavimo kalbos kodą.

Kodo transformavimo posistemė skirta C# programavimo kalbos programinio kodo generavimui. Šis modulis sugeneruoja C# programavimo kalbos kodą iš nurodytų duomenų ir jį sukompiliuoja. Kompiliavimui naudojamas „Visual Studio“ programos kartu pridedamą šios kalbos kompiliatorių.

5.3. Testavimo modelis

5.3.1. Testavimo tikslai ir objektai

Pagrindinis testavimo tikslas yra pateikti įtikinamus įrodomus, kad sukurta vizuali robotų programavimo ir robotų sąveikos modeliavimo programa veikia tinkamai ir be klaidų. Visgi tai padaryti praktiškai neįmanoma, bet galima pateikti programinę įrangą su kuo mažesniu galimu defektų kiekiu.

Šiam tikslui įgyvendinti pagal testavimo planą išskiriami pagrindiniai testavimo objektai yra vartotojo sąsaja, atskirti programinės įrangos moduliai ir šių skirtingų sistemos komponentų bendravimas tarpusavyje. Taip pat šiuo testavimu siekiama patikrinti ir užtikrinti, jog sukurta programinė įranga atitinka sistemos reikalavimų specifikaciją.

5.3.2. Pagrindiniai apribojimai

Šiame skyrelyje aprašoma organizacinius, techninius ir programinius testavimo apribojimus.

1. Kūrėjai neturi resursų atlikti programos veikimo testavimo įvairiuose techniniuose įrenginiuose, todėl bus daroma prielaidą, jog veikiant mažiau resursų turinčiuose įrenginiuose veiks ir daugiau resursų turinčiuose.
2. Vizualioje robotų programavimo sistemoje leidžiama sukurti iki 5000 skirtingų mazgų.

5.3.3. Testavimo procedūra

5.3.3.1. Vienetų testavimas

Vienetų testavimo metu testuojamas kiekvienas galimas vizualios programavimo kalbos elementas. Išbandoma ar mazgai sukuria tinkamus programinius medžius. Patikrinamas individulių elementų veikimas. Testuojama ar kuriant abstraktų sintaksės medį tinkamai perduodamos vartotojo nustatytos mazgų reikšmės. Vizualios programavimo testavimas tuo ir baigiamas, kadangi tolimesni testavimai yra integracinio ir priėmimo tipo.

5.3.3.2. Integravimo testavimas

Integravimo testavimas atliekamas tarp skirtingų sistemos komponentų siekiant užtikrinti teisingą tarpusavio sąveiką.

Testavimo metu parengiami testai apjungiantys sistemą nuo mažiausių pavienių komponentų vis prijungiant naujus. Integraciniam testavimui sukuriami fiktyvūs objektai galintys imituoti dar neprijungtų komponentų veiklą. Esant sėkmingam testui prijungiamas naujas komponentas. Tęsiama kol visi sistemos komponentai integruojami į sistemą ir atitinka testų scenarijų reikalavimus. Integraciniu testavimu siekiama rasti komponentų tarpusavio sąveikos defektus.

5.3.3.3. Priėmimo testavimas

Priėmimo testavimas atliekamas juodos dėžės principu. Šio testavimo esmė patikrinti vizualaus robotų programavimo sąsajos ir robotų sąveikos modeliavimo aplinkos sistemos atitikimą specifikacijoje keliamiems reikalavimams.

Testavimo metu tikrinama ar vizualios robotų programavimo kalbos elementai taisyklingai paverčiami į programinį kodą, kurį taisyklingai galėtų suprasi robotų modeliavimo aplinka. Siekiama užtikrinti, jog iš modeliavimo aplinkos gauti jutiklių duomenys būtų korektiški, o vizualios programavimo kalbos elementai atvaizduojami taisyklingai. Patikrinama sujungtos vizualios robotų programavimo kalbos mazgų grandinės taisyklingumas paverčiant į robotų simuliacijai suprantamą programinį kodą.

5.3.3.4. Aukšto lygio testavimas

Vizuali robotų programavimo sąsaja bus tikrinama kaip vartotojas greitai prie jos prisitaiko ir kaip lengvai ją perpranta. Atsižvelgiant į jų pageidavimus daromi neesminiai pakeitimai. Testavimas vykdomas dalies vartotojų nesupažindinus su aplinka, siekiant išgauti jų reakciją. Nustatoma kaip lengvai be jokio gido žmogus supranta vartotojo sąsają. Kita vartotojų grupė apmokoma elgtis su programine įranga, siekiant įvertinti kaip lengvai vartotojai išmoksta naudotis sistema.

5.3.4. Testavimo resursų paskirstymas

5.3.4.1. Resursai

Pagrindinis testavimo resursas – vizuali robotų programavimo ir robotų sąveikos modeliavimo aplinkos sistema. Visa ši sistema susideda iš 3 skirtingų didelių sistemos komponentų, kurie testuojami atskirai ir juos apjungiant. Šie komponentai: vizuali robotų programavimo posistemė, robotų sąveikos modeliavimo aplinkos posistemė ir šias dvi sistemas apjungiantis komponentas.

5.3.4.2. Personalas

Programuotojai ir testuotojai testuoja sukurtos sistemos:

- Visus atskirus modulius;
- Sąveika tarp atskirų modulių;
- Visą sistemos funkcionalumą.

5.3.5. Testavimo rezultatų kaupimas

Testų aprašymai ir rezultatai saugomi „Microsoft Excel“ failo sukurtoje lentelėje. Pavaizduota tokią 5.1 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
...	...	...	...	...

5.4. Testavimo rezultatai ir išvados

Naudojant *test driven development* sumažintas kodo klaidų skaičius dar jų rašymo metu. Taip pat turint vienetų testus ir padarius pakeitimus aptinkamos pakeitimų sukeltos klaidos. Tai leido sumažinti kodo defektų skaičių ir padidinti programos kokybę. Sistemai atlikti pakeitimai priimami tik tada, kai visi vienetų testai būna sėkmingi.

6. EKSPERIMENTINIS ALGORITMŲ METRIKŲ TYRIMAS

6.1. Eksperimento planas

Vizualią programavimo kalbą transformavus į programavimo kalbos kodą galima nustatyti įvairias metrikas, kurios padėtų įvertinti sugeneruoto kodo sudėtingumą. Kadangi sukurta vizuali robotų programavimo kalba transformuojama į C# programinį kodą palyginimui buvo pasirinkta „Microsoft Robotics Developer Studio“ vizualaus programavimo kalba, kuri pritaikyta robotų elgsenai valdyti bei gali būti transformuojama į C# programavimo kalbos kodą.

Pagrindinis tikslas sužinoti vizualių programavimo kalbų transformavimo galimybes. Išsiaiškinti, kaip galima supaprastinti transformuojama programinį kodą, jog iš vizualios programavimo kalbos gautą kodą būtų galima lengvai suprasti, redaguoti ir pritaikyti tolimesniam naudojamui.

Tyrimui naudojamos „SourceMonitor“ programos matuojamos metrikos [18]:

- **Eilutės** (*angl. lines*) – visų failų eilučių skaičius į kurį įskaičiuojamas programinio kodo, tuščių ir komentarų eilutės.
- **Sakiniai** (*angl. statements*) – programinio kodo eilučių skaičius, išskyrus tuščias ir komentarų eilutes. C# programavimo kalboje skaičiuojami sakiniai yra nutraukiami kabliataškio simboliu. Metodai, nors nutraukiami ne kabliataškiais, skaičiuojami kaip sakiniai. Visi atributai skaičiuojami kaip sakiniai, nors iškvietimai į atributus yra ignoruojami. Šakos kaip **if**, **for** ir **while**, bei išimties kontrolių sakiniai **try, catch** ir **finally** yra skaičiuojami kaip sakiniai. Tokios direktyvos kaip **#define** ir **#undef** skaičiuojami kaip sakiniai, tačiau visos kitos direktyvos ignoruojamos. Visgi visi sakiniai tarp kiekvieno **#else** ar **#elif** sakinio ir uždarančio **#endif** sakinio ignoruojami. Goto etiketės (*angl. labels*) skaičiuojamos kaip sakiniai.
- **Komentarai** (*angl. comments*) – eilutės, kuriuose pateikti komentarai. Gali būti pateikti C tipu (/*...*/) arba C++ tipu (//...). Skaičiuojamos visos komentarų eilutės ir palyginamos su bendru eilučių skaičiumi faile.
- **Dokumentacija** (*angl. documentation*) – eilutės, kurios dokumentuoja programinį kodą. Tai eilučių skaičius, kurios yra tarp „///“ simbolių turinčias eilutes. Skaičiuojamos visos dokumentacijos eilutės ir palyginamos su bendru eilučių skaičiumi faile.
- **Klasės** (angl. classes) – klasių (*angl. classes*), sąsajų (*angl. interfaces*) ir struktūrų (*angl. structs*) bendras skaičius.
- **Metodai klasėje** (*angl. methods/class*) – skaičiuojamos metodų realizacijos klasėse (*angl. class*), struktūrose (*angl. struct*) ir šablonuose (*angl. template*).
- **Sakinių metoduose vidurkis** (*angl. avg stmts/method*) – vidutinis sakinių skaičius rastas metodų viduje.

- **Kitų metodų iškvietimai metoduose** (*angl. calls/method*) – visų kitų faile rastų metodų iškvietimų skaičius per metodą ir visų metodų faile skaičiaus santykis.
- **Maksimalus sudėtingumas** (*angl. max complexity*) – didžiausia sudėtingumo reikšmė iš sudėtingiausio metodo ar funkcijos failo. Sudėtingumo skaičiavimo metrikas apibrėžė Steve McConnell savo knygoje „Code Complete“, Microsoft Press, 1993m. Sudėtingumo metrika nusako vykdymo kelių skaičių funkcijoje ar metode. Kiekvienos funkcijos ar metodo sudėtingumas didėja vienu dėl kiekvienos šakos sakinio kaip **if**, **else**, **for**, **foreach** ar **while**. Perjungimo (*angl. switch*) sakiniai padidina sudėtingumo skaičių kiekvieno išėjimo atveju (dėl **break, goto, return, throw, continue, default** ar panašių sakinių), taip kaip ir kiekvienas **catch** ar **except** sakinys bandymo bloke (bet ne **try** ar **finally** sakiniai).
- **Maksimalus gylis** (*angl. max depth*) – surastas didžiausias bloko gylio lygis. Pradžioje kiekvieno failo bloko lygis yra nulinis.
- **Vidutinis gylis** (*angl. avg depth*) – vidutinis bloko gylio koeficientas gautas apskaičiavus visus failo sakinius. Blokai dažniausiai visada pateikiami su vykdymo kontrolės sakiniais, kaip **if**, **case** ir **while**, todėl didėjant gyliui sunkėja kodo skaitymo suprantamumas, nes su kiekvienu nauju gylio lygiu atsiranda daugiau sąlygų, todėl sunkiau įvertinti jeigu norime sužinoti kada kodas yra iškviečiamas.
- **Vidutinis sudėtingumas** (*angl. avg complexity*) – skaičiuojama paprastu aritmetiniu vidurkiu iš visų sudėtingumo reikšmių, kurios gautos apskaičiuojant nurodytų failų metodus.

6.2. Eksperimento rezultatai

6.1. lentelėje pateikti rezultatai buvo gauti analizuojant „Microsoft Robotics Developer Studio“ ir sukurtos „RobotVL“ vizualiomis programavimo kalbomis parašyti algoritmai ir jų transformacijos į C# programavimo kalbos kodą.

Šioje 6.1 lentelėje pateikti tą patį roboto veiksmą turintys atlikti algoritmai. Kiekvienas algoritmas turi važiuoti tiesiai, kol priartės prie kliūties. Priartėjus prie kliūties robotas turi suktis į dešinę pusę, kol nebematys jos. Nematant robotui kliūtis, jis vėl turi važiuoti tiesiai.

Šį paprastą algoritmą su vizualiomis programavimo kalbomis galima parašyti įvairiai, todėl prioritetas parašyti panaudojus kuo mažiau vizualios programavimo kalbos mazgų. Visgi net šiuo atveju viskas priklauso nuo programuotojo, todėl tikslinga išbandyti kelis panašius variantus ir išmatuoti transformuoto C# programinio kodo pusę, norint ją tinkamai įvertinti.

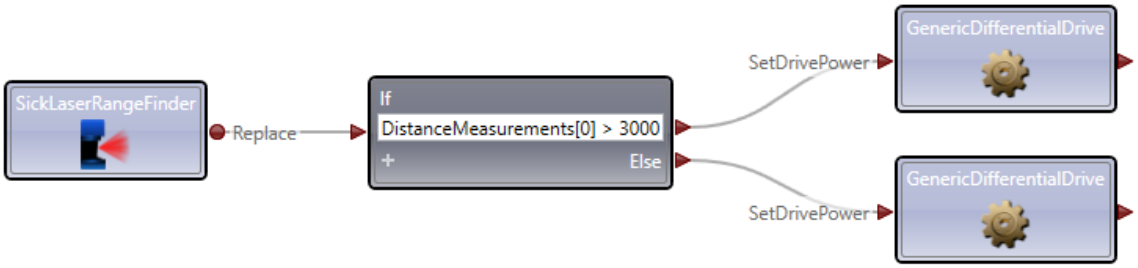
6.1. lentelėje pateikti VPL-1 ir VPL-2 (pavaizduoti 6.1 ir 6.2 paveikslėliuose) algoritmai parašyti „Microsoft Robotics Developer Studio“ vizualia programavimo kalba. VPL-1 algoritme nenaudojami vizualios programavimo kalbos kintamųjų mazgai, visos reikiamos reikšmės aprašomos mazgus

jungiančiuose linijose. Priešingai nuo VPL-2 algoritmo, kuriame naudojami vizualios programavimo kalbos kintamųjų mazgai, todėl šis algoritmas turi daugiau vizualios programavimo kalbos mazgų.

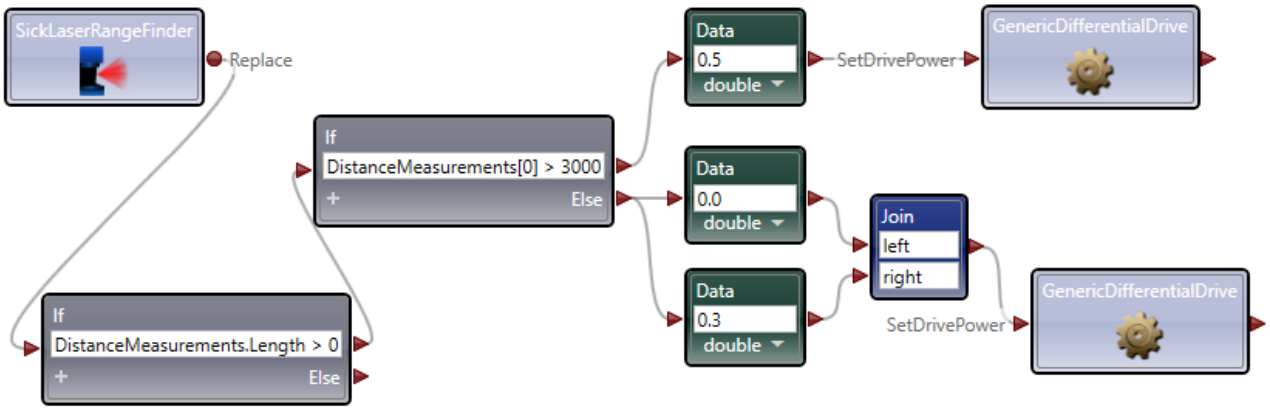
6.1. lentelėje pateikti RobotVL-1 ir RobotVL-2 (pavaizduoti 6.3 ir 6.4 paveikslėliuose) algoritmai parašyti kurtos „RobotVL“ vizualia programavimo kalba. RobotVL-1 algoritmas priešingai nuo RobotVL-2 yra pilnai aprašytas. Visgi pilnam suformuotam užduoties įgyvendinimui nereikia pilno kodo užrašymui, todėl sutrumpintas RobotVL-1 alternatyva yra RobotVL-2 parašytas kodas, kuris turi 3 mazgais mažiau nei RobotVL-1 parašytas algoritmas.

6.1 lentelė. VPL-1, VPL-2, RobotVL-1, RobotVL-2 algoritmų programinio kodo matavimų rezultatai

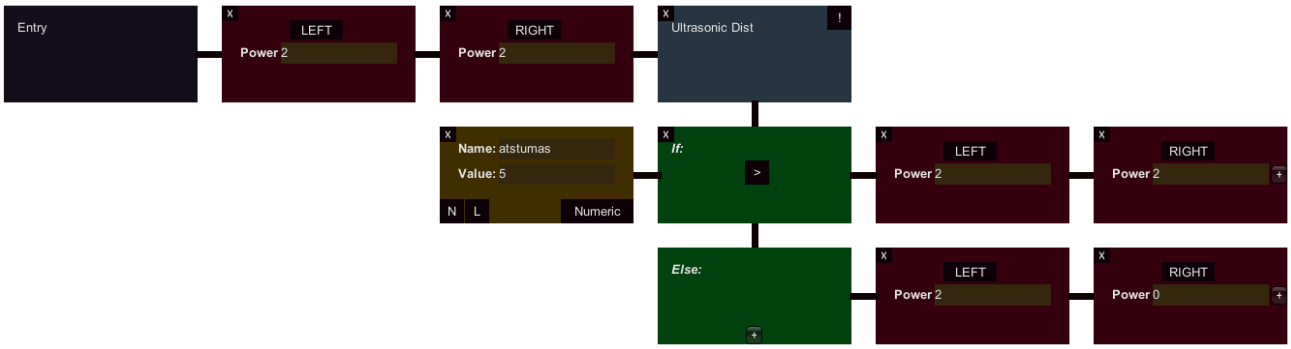
	Mazgų skaičius	Failų	Eilutės	Sakiniai	Komentarai, %	Klasės	Metodai klasėje	Kitų metodų iškvietimai metoduose	Sakinių metoduose vidurkis	Maksimalus sudėtingumas	Maksimalus gylis	Vidutinis gylis	Vidutinis sudėtingumas
VPL-1	4	2	864	354	2,4	14	4,93	1,09	2,83	7	6	2,31	1,84
VPL-2	9	2	872	359	2,4	14	4,93	1,09	2,87	7	6	2,32	1,84
RobotVL-1	11	1	101	53	0	2	3,5	1,57	5,43	9	8	3,45	3,14
RobotVL-2	8	1	95	49	0	2	3,5	1,14	4,86	9	8	3,43	3



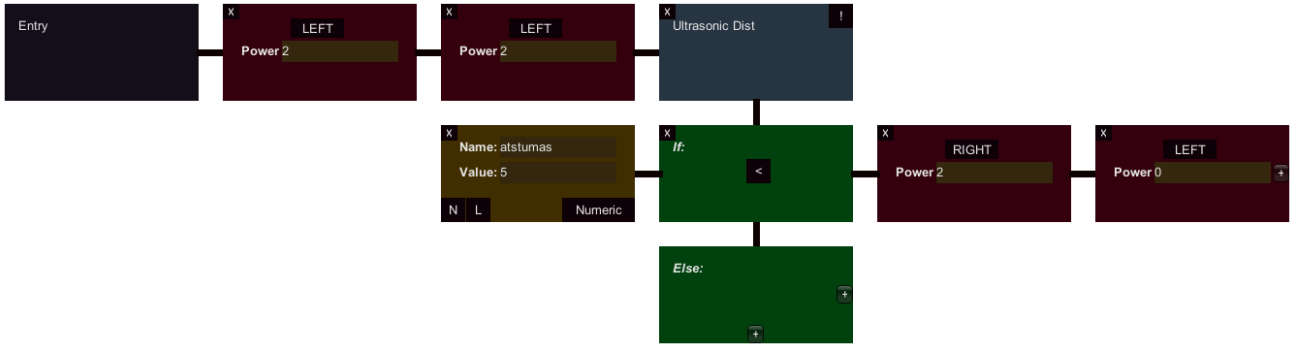
6.1 pav. VPL-1 algoritmo programinio kodo vaizdas



6.2 pav. VPL-2 algoritmo programinio kodo vaizdas

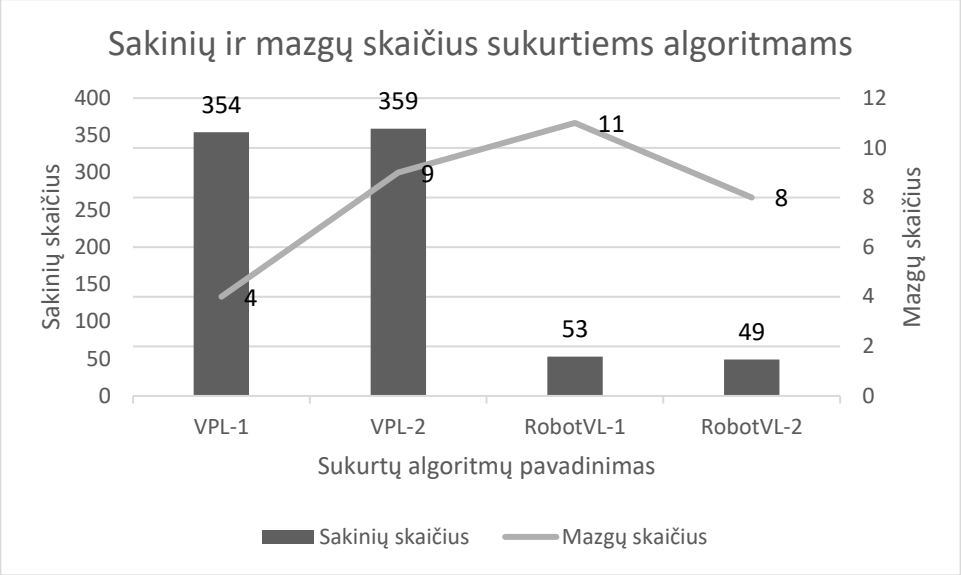


6.3 pav. RobotVL-1 algoritmo programinio kodo vaizdas



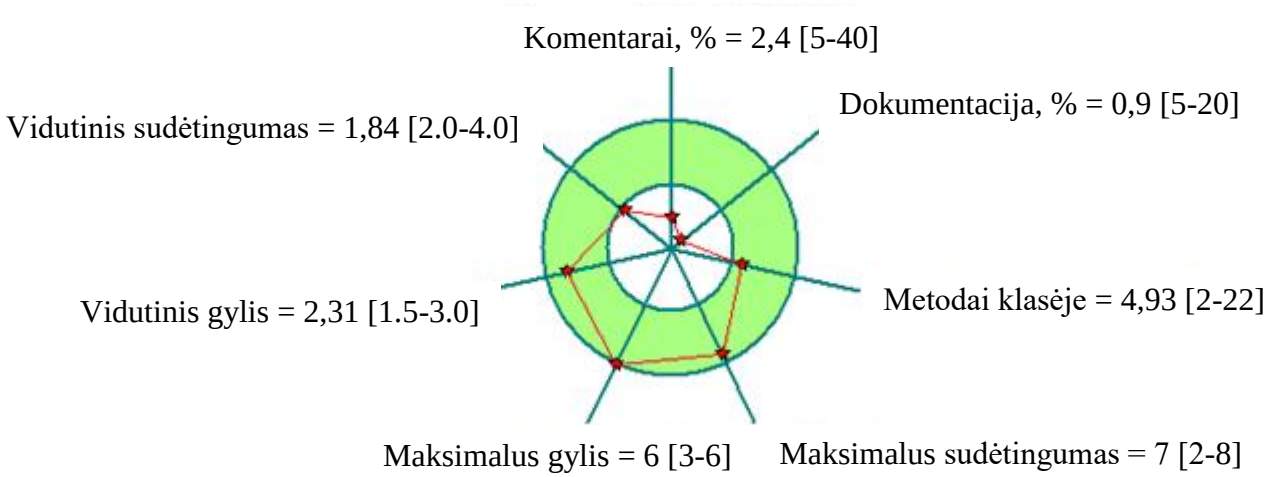
6.4 pav. RobotVL-2 algoritmo programinio kodo vaizdas

Pateiktame 6.5 paveikslėlyje matoma sukurtų algoritmų vizualios programavimo kalbos mazgų skaičių ir juos atitinkančius transformuoto C# programinio kodo sakinių skaičių. Ta pačia vizualia programavimo kalba tiek VPL-1 ir VPL-2, tiek RobotVL-1 ir RobotVL-2 parašytas algoritmo įgyvendinimas skiriasi mazgų skaičiumi, tačiau dėl šios priežasties ne daug tik padidėja C# programinio kodo sakinių skaičius. Visgi matoma, jog „Microsoft Robotics Developer Studio“ vizualaus programavimo kalba (*angl.VPL*) parašytas algoritmas transformavus į C# programavimo kalbos kodą sakinių skaičius yra kur kas didesnis, nei sukurtos „RobotVL“ vizualios programavimo kalbos, nors naudojamas vizualios programavimo kalbos mazgų skaičius apyligis. Mažiau sakinių turintis kodas lengviau suprantamas.

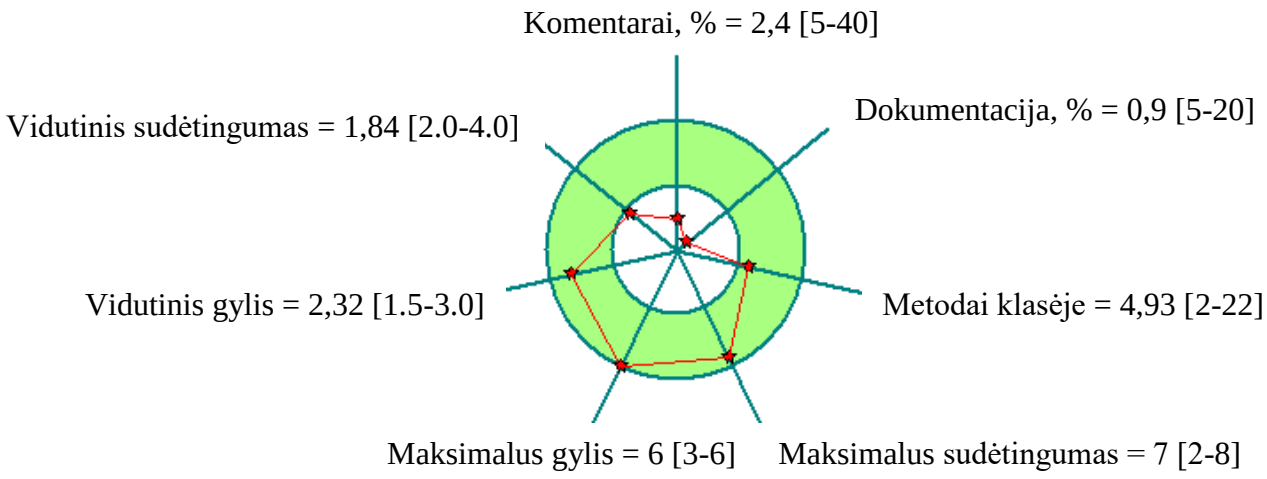


6.5 pav. VPL-1, VPL-2, RobotVL-1, RobotVL-2 algoritmų sakinių ir mazgų skaičiaus pasiskirstymas pagal sukurtus algoritmus diagrama

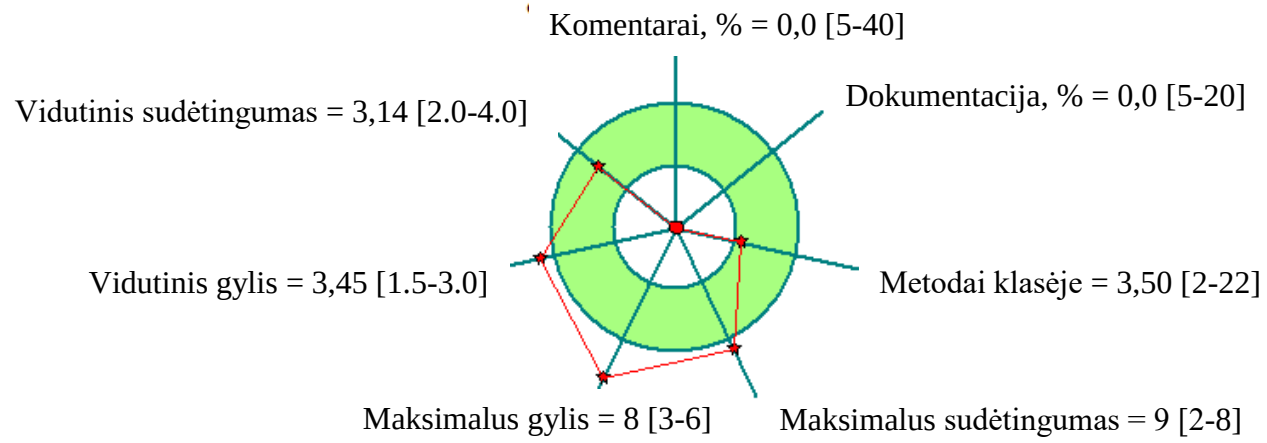
Pateikti 6.6, 6.7, 6.8, 6.9 paveikslėliai parodo sukurtų algoritmų *Kiviat* metrikų diagramas. Iš šių diagramų matoma, kaip metrikos atitinka rekomenduojama jų dydį. Algoritmo programinis kodas transformuotas iš „RobotVL“ vizualios programavimo kalbos neturi komentarų ir dokumentacijos, taip pat viršija numatytas rekomenduotinas vidutinio gylio, maksimalaus gylio ir maksimalaus sudėtingumo reikšmes. Priešingu atveju algoritmo programinis kodas transformuotas iš „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos atitinka rekomenduojamas metrikų reikšmes, o komentarų ir dokumentacijos dalis yra mažesnė nei rekomenduojama.



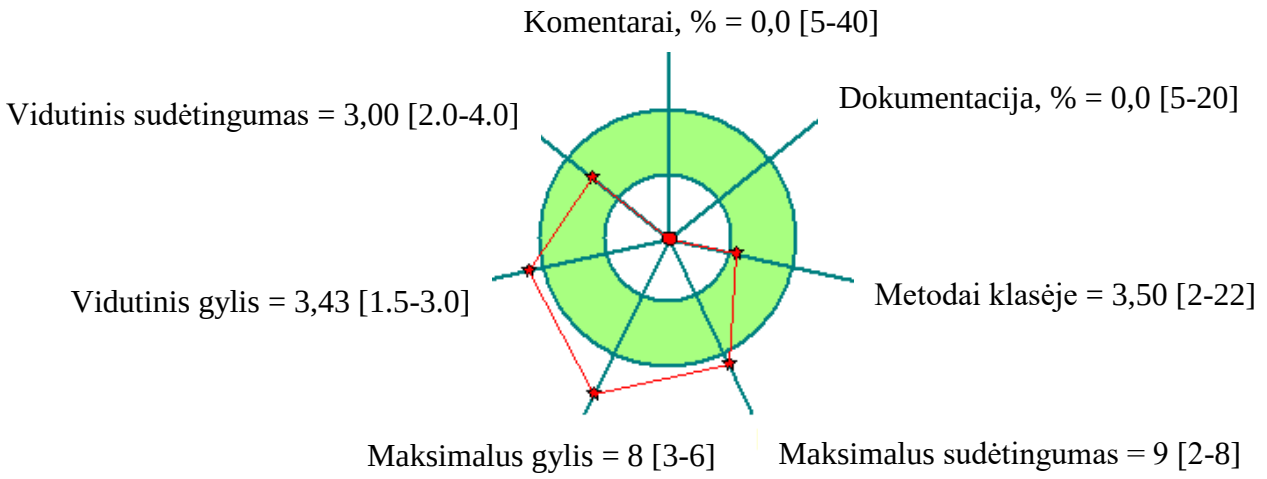
6.6 pav. VPL-1 *Kiviat* metrikų diagrama



6.7 pav. VPL-2 *Kiviat* metrikų diagrama



6.8 pav. RobotVL-1 *Kiviat* metrikų diagrama



6.9 pav. RobotVL-2 *Kiviat* metrikų diagrama

Kitu atveju buvo tiriamas sunkesnės užduoties įgyvendinimas pasitelkus „Microsoft Robotics Developer Studio“ ir sukurtos „RobotVL“ vizualiomis programavimo kalbomis. Šiuo atveju užduotis buvo įgyvendinti roboto algoritmą, kuris galėtų išvengti roboto kelyje pasitaikančias kliūtis, kai robotas turi skirtinguose pusėse ultragarsinius jutiklius, išdėstytus jo centre, kairėje ir dešinėje pusėje. Robotui, gaunant iš skirtingų pusių ultragarso jutiklio duomenims, reikia nuspręsti į kurią pusę geriausia judėti. Algoritmo supaprastinimui robotas juda ta kryptimi, kurioje nustatytas didžiausias atstumas iki kliūties.

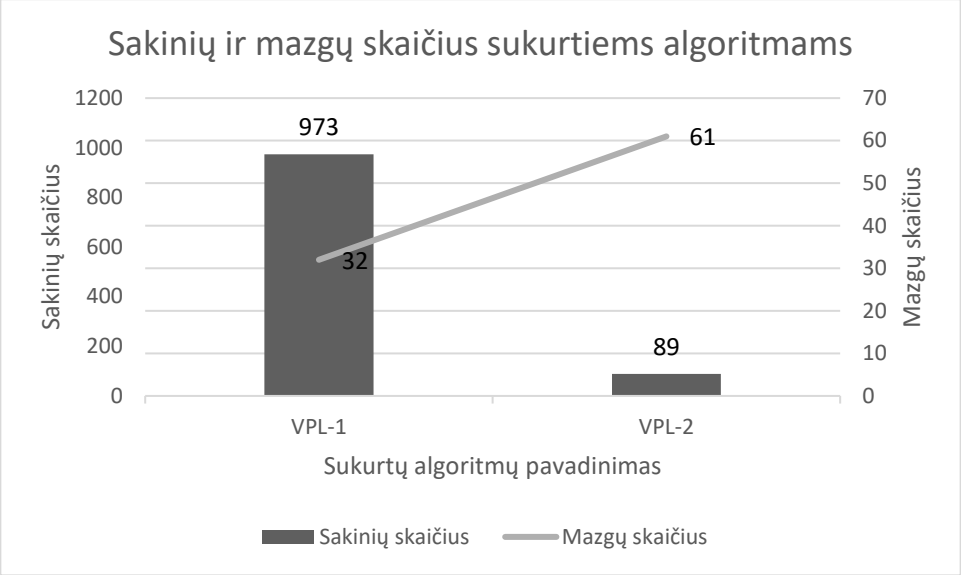
Šio algoritmo matavimų rezultatai pateikti 6.2. lentelėje buvo gauti išmatavus „Microsoft Robotics Developer Studio“ ir „RobotVL“ vizualių programavimo kalbų transformacijos į C# programavimo kalbos kodą.

6.2. lentelėje pateiktas VPL-3 algoritmas parašytas „Microsoft Robotics Developer Studio“ vizualia programavimo kalba, o RobotVL-3 algoritmas parašytas atitinkamai „RobotVL“ vizualia programavimo kalba. Buvo bandoma įgyvendinti kuo panašesnį algoritmo veikimą ir palyginti su anksčiau aprašyto roboto judėjimo algoritmo gautais matavimų rezultatais, jog galėtume įvertinti kaip algoritmo pasunkinimas daro įtaką vizualiai programavimo kalbai ir jos transformacijai į C# programinį kodą matavimams.

6.2 lentelė. VPL-3 ir RobotVL-3 algoritmų programinio kodo matavimų rezultatai

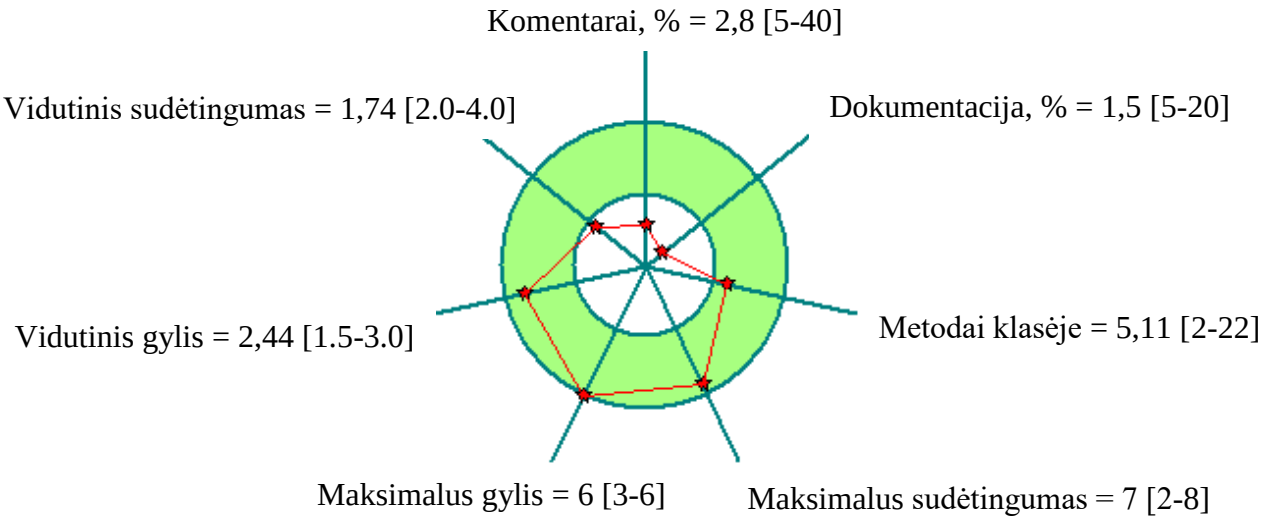
	Mazgų skaičius	**Failų**	**Eilutės**	**Sakiniai**	**Komentarai, %**	**Klasės**	**Metodai klasėje**	**Kitų metodų iškvietimai metoduose**	**Sakinių metoduose vidurkis**	**Maksimalus sudėtingumas**	**Maksimalus gylis**	**Vidutinis gylis**	**Vidutinis sudėtingumas**
VPL-3	32	4	2280	973	2,8	35	5,11	1,21	3,06	7	6	2,44	1,74
RobotVL-3	61	1	163	89	0	2	5,5	1,55	5,73	9	8	3,27	3,18

Pateiktame 6.10 paveikslėlyje matoma sukurtų algoritmų vizualios programavimo kalbos mazgų skaičių ir juos atitinkančius transformuoto C# programinio kodo sakinių skaičių. Dvejomis skirtingomis vizualiomis programavimo kalbomis parašyti algoritmai VPL-3 ir RobotVL-3 įgyvendinimui prireikė parašyti skirtingą mazgų skaičių. Pastebima, jog stipriai išaugo mazgų skaičius RobotVL-3 algoritme, tačiau sakinių skaičius C# programiniame kode išliko daug mažesnis. Tai galima daryti išvada, jog sudėtingėjant įgyvendinamai užduočiai parašyta „RobotVL“ vizualia programavimo kalba prireikia daugiau mazgų skaičiaus, nei tokią pačią užduotį įgyvendinančiai „Microsoft Robotics Developer Studio“ vizualiai programavimo kalbai. Visgi transformavus šį kodą į C# programinį kodą „RobotVL“ užrašomas kur kas mažiau sakinių turinčiu C# programavimo kalbos kodu, nei „Microsoft Robotics Developer Studio“ vizualia programavimo kalba sukurtas kodas.

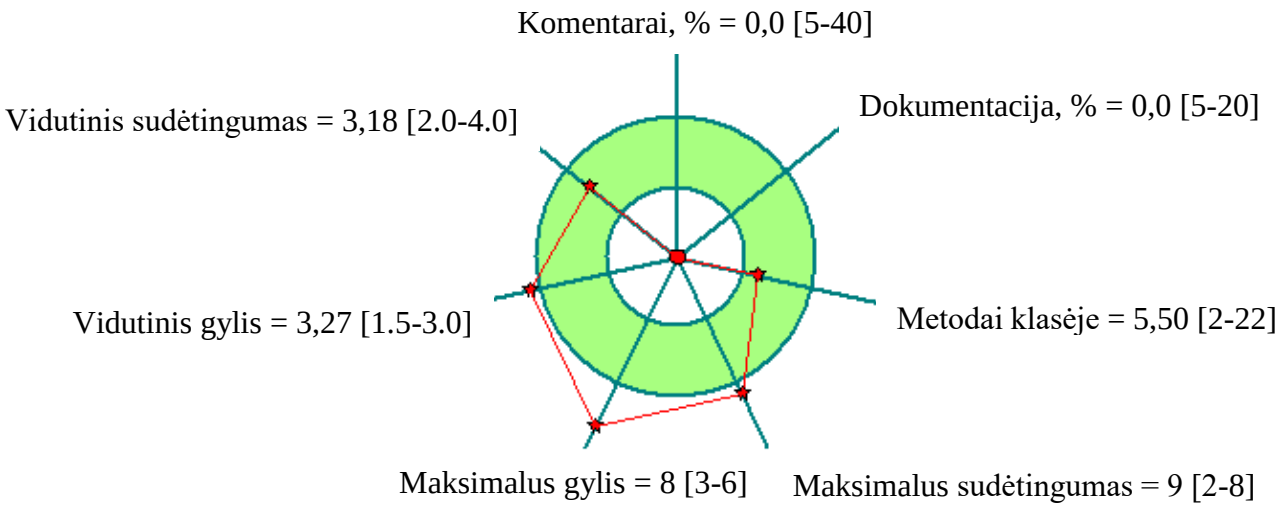


6.10 pav. VPL-3 ir RobotVL-3 algoritmų sakinių ir mazgų skaičiaus pasiskirstymas pagal sukurtus algoritmus diagrama

Pateikti 6.11 ir 6.12 paveikslėliai parodo sukurtų VPL-3 ir RobotVL-3 algoritmų *Kiviat* metrikų diagramas. Iš šių diagramų matoma kaip metrikos atitinka rekomenduojama jos dydį. Algoritmo C# programinis kodas transformuotas iš „RobotVL“ vizualios programavimo kalbos neturi komentarų ir dokumentacijos, kaip ir įgyvendinus paprastesnį algoritmą, šiame viršija numatytas rekomenduotinas vidutinio gylio, maksimalaus gylio ir maksimalaus sudėtingumo reikšmes. Visgi reikšmės išlikusios tokios pačios ar net pagerėjusios, reiškia transformuojamas C# programinis kodas nesudėtingėja. Algoritmo C# programinis kodas transformuotas iš „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos atitinka rekomenduojamas metrikų reikšmes. Šio algoritmo komentarų ir dokumentacijų dalis mažesnė nei rekomenduojama. Visgi sudėtingėjant įgyvendinamai užduočiai transformuota vizuali programavimo kalba į C# programinį kodą išlaiko rekomenduotinas kodo metrikų reikšmes.



6.11 pav. VPL-3 *Kiviat* metrikų diagrama



6.12 pav. RobotVL-3 *Kiviat* metrikų diagrama

6.3. Eksperimento išvados

Eksperimento metu pasiekti šie rezultatai ir nustatytos tokios išvados:

1. Eksperimento metu tirti algoritmų užrašymai „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualiomis programavimo kalbomis ir gauti šių algoritmų metrikų rezultatai parodė kaip skirtingų vizualių programavimo kalbos mazgų skaičius daro įtaka transformuoto C# programinio kodo sakinių skaičiui.
2. Eksperimento metu gauti rezultatai parodė, jog „RobotVL“ vizuali programavimo kalba reikalauja didesnio mazgų užrašymo skaičiaus tokiam pačiam algoritmui, nei „Microsoft Robotics Developer Studio“ vizuali programavimo kalba. Pastebėta, jog toks rezultatas gaunamas dėl galimybės mazgo reikšmes įvesti tik jo viduje, o ne kaip „Microsoft Robotics Developer Studio“ vizualioje programavimo kalboje mazgo reikšmes galima įvesti ir mazgus sujungiančioje linijoje.
3. Eksperimento metu iš gautų metrikų rezultatų nustatyta, jog „RobotVL“ vizualios programavimo kalbos transformacija į C# programavimo kalbos kodą sukuriama apie 8 kartus mažiau reikalaujančio kodo sakinių, nei „Microsoft Robotics Developer Studio“ vizuali programavimo kalba.
4. Eksperimento metu iš gautų metrikų rezultatų nustatyta, jog „RobotVL“ vizualios programavimo kalbos sugeneruotas C# programinis kodas neatitinka rekomenduotinų metrikų dydžių. Sugeneruotame programos kode iš vis nebuvo dokumentavimo, o vidutinio gylio, maksimalaus gylio ir maksimalaus sudėtingumo metrikų reikšmės per didelės. Dėl šios priežasties reikalingas transformuojamo kodo tvarkymas, kuris leistų pagerinti suformuoto kodo metrikų reikšmes.
5. Didesnis sakinių skaičius apsunkina programinio kodo skaitomumą ir suprantamumą, tačiau dėl didelio funkcionalumo atskyrimo į metodus šis sugeneruotas C# programinis

kodas atitinka beveik visus rekomenduotinus metrikų dydžius. Metrikų rezultatas parodė, jog sugeneruoto kodo dokumentacijos dalis yra per maža.

6. Eksperimento metu, įgyvendinant skirtingus algoritmus „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualiomis programavimo kalbomis, pastebėta, jog transformuoto į C# programinį kodą metrikos išlieka panašaus dydžio, skiriasi tik sakinių skaičius.
7. Tiriant sudėtingesnio algoritmo užrašymą „Microsoft Robotics Developer Studio“ vizualia programavimo kalba gautas sakinių skaičius dar labiau išauga lyginant su „RobotVL“ vizualia programavimo kalba. Visgi kitos metrikos išlieka beveik nepakitusios.
8. Eksperimento metu skirtingo sunkumo algoritmų įgyvendinimo gauti rezultatai parodė, jog „RobotVL“ vizualia programavimo kalba rašant sudėtingesnius algoritmus greičiau didėja reikalingų mazgų skaičius įgyvendinat tokį patį algoritmą, nei „Microsoft Robotics Developer Studio“ vizualia programavimo kalba. Dėl šios priežasties, galima daryti išvadą, jog „RobotVL“ sudėtingesnių algoritmų užrašymui prireiks didesnio mazgų skaičius. Norint tai pagerinti reikėtų „RobotVL“ vizualios programavimo kalbos metodo mazgo tipui suteikti didesnį funkcionalumą, jog metodą su tuo pačiu atliekamu funkcionalumu būtų galima perpanaudoti, rašant algoritmą vizualia programavimo kalba.

7. EKSPERIMENTINIS SISTEMOS APKLAUSOS TYRIMAS

7.1. Eksperimento planas

Vartotojui leisti išmėginti parašyti roboto elgesio algoritmą su „Microsoft Robotics Developer Studio“ ir sukurta „RobotVL“ vizualiomis programavimo kalbomis. „Microsoft Robotics Developer Studio“ vizuali programavimo kalba palyginimui pasirinkta dėl savo modeliavimo aplinkos turėjimo, kuri atitinka sukurtai „RobotVL“ sistemai. Visgi „Microsoft Robotics Developer Studio“ yra komercinė sistema turinti didelį robotų funkcijų pasirinkimą, todėl vartotojas turėtų abiejomis vizualiomis programavimo kalbomis parašyti tą patį roboto elgesį atitinkantį algoritmą.

Norint vartotojui leisti išmėginti kuo įvairesnį vizualių programavimo kalbų funkcionalumą, tačiau neužmirštant, jog bandantys vartotojai gali pirma kartą susidurti su vizualiomis programavimo kalbomis ar iš vis programavimu. Taigi roboto elgesio algoritmas neturėtų būti sudėtingas, o tik leisiantis išbandyti sistemų ir vizualių programavimo kalbų naudojamumą. Kiekvienas vartotojas, bandantis vizualias programavimo kalbas, turės parašyti roboto elgesio algoritmą, kuris leistų robotui judėti į priekį kol nėra jokių kliūčių. Atsiradus kliūčiai robotas turi sukti į dešinę, kol nebematys kliūties ir tęsti roboto judėjimą.

Parašius algoritmą vizualia programavimo kalba vartotojams, norint sužinoti jų vertinimą, duodamas SUS (*angl. System Usability Scale*) klausimynas, kurį sudaro 10 klausimų. Kiekvienas klausimas gali būti vertinamas nuo 1 iki 5 (nuo stipriai nesutikti iki stipriai sutikti). Pateikiami 10 SUS klausimų [19]:

1. Manau, jog norėčiau naudotis šia sistema reguliariai.
2. Manau, jog sistema be reikalo tokia sudėtinga.
3. Manau, jog sistema yra lengva naudotis.
4. Manau, jog man reikėtų techninio personalo pagalbos, kad galėčiau naudotis šia sistema.
5. Manau, jog įvairus sistemos funkcionalumas yra gerai integruotas.
6. Manau, jog sistema per daug nenuosekli ir nesuderinta.
7. Įsivaizduoju, jog dauguma žmonių išmoktų greitai naudotis šia sistema.
8. Man sudėtinga naudotis šia sistema.
9. Naudojantis sistema aš jaučiausi patikimai.
10. Man reikėjo išmokti daug dalykų, jog galėčiau normaliai naudotis šia sistema.

Užpildytas visas vartotojų anketas reikia įvertinti. Anketų atsakymai (nuo 1 iki 5 balų) perskaičiuojami pagal šias taisykles:

- Iš nelyginio klausimo balo atimant 1.
- Kiekvienas lygino klausimo balas atimamas iš 5.

Tokiu būdu pertvarkomi balai svyruoja nuo 0 iki 4, o visus balus sudėjus ir padauginus iš 2.5 gaunamas įvertinimas nuo 0 iki 100. Remiantis tyrimais SUS įvertinimas virš 68 būtų laikomas didesnis už vidutinį, o priešingu atveju žemesnis, nei 68 būtų laikomas mažesniu už vidutinį [20]. Toks vertinimas padeda įvertinti vartotojų sąsajų ar produktų naudojamumo kokybę.

7.2. Eksperimento rezultatai

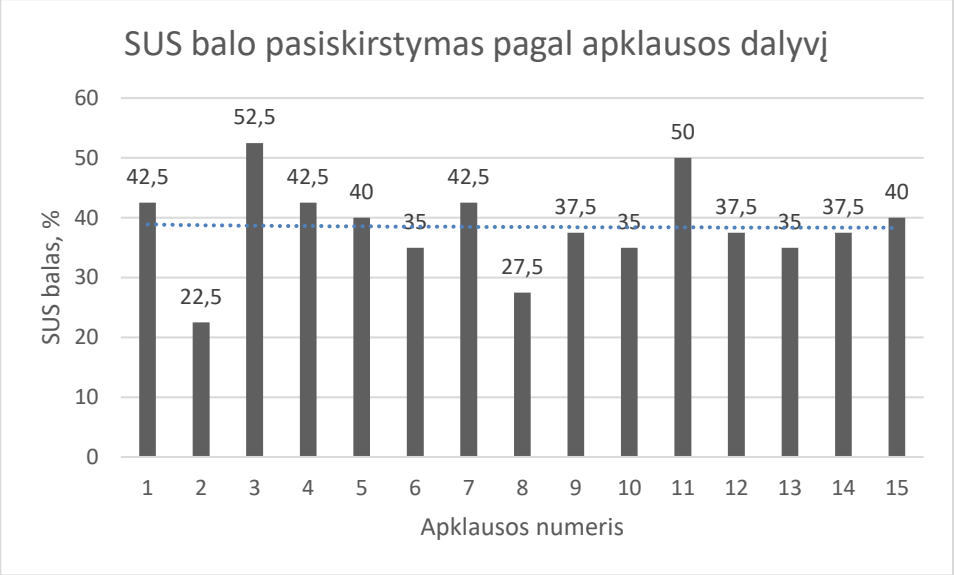
15 vartotojų išbandė „Microsoft Robotics Developer Studio“ ir „RobotVL“ vizualias programavimo kalbas. Kiekvienas vartotojas atsakė į pateiktus 10 SUS klausimus, iš kurių buvo apskaičiuotas SUS balas.

„Microsoft Robotics Developer Studio“ vizualios programavimo kalbos (*angl. VPL*) apklausos rezultatai pateikti 7.1 lentelėje. Iš lentelės matome, jog blogiausias SUS balas – 22,5%, geriausias SUS balas – 52,5%. Visų apklausų SUS balo rezultatai nusako „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos vidutinį 38,5% SUS balą.

7.1 lentelė. „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos apklausos rezultatai

VPL	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Vid.**
1	3	2	3	3	3	3	3	2	3	3	3	3	3	3	3	2,87
2	4	5	4	4	4	4	4	5	4	4	4	4	4	4	4	4,13
3	2	1	3	2	2	3	3	2	2	2	3	3	2	3	3	2,40
4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5,00
5	5	3	2	5	4	3	2	3	4	4	3	2	4	4	2	3,33
6	1	3	1	1	1	3	1	2	1	3	1	2	2	3	2	1,80
7	3	2	4	3	3	2	3	2	3	2	4	3	3	2	4	2,87
8	5	5	3	5	5	4	4	5	5	4	3	4	5	4	4	4,33
9	4	4	4	4	4	4	4	4	3	4	4	4	3	4	4	3,87
10	5	5	2	5	5	5	4	5	5	5	4	5	5	5	5	4,67
Balų suma	17	9	21	17	16	14	17	11	15	14	20	15	14	15	16	15,4
SUS balas	42,5	22,5	52,5	42,5	40	35	42,5	27,5	37,5	35	50	37,5	35	37,5	40	38,5

Apibendrinta 7.1 lentelės visų apklausų duomenų diagrama pateikta 7.1 paveikslėlyje. Iš paveikslėlio matoma, jog yra nemažas SUS balo pasiskirstymas. Kadangi ši sistema komercinė ir išleista senai, galima daryti prielaidą, jog šią sistemą geriau įvertinto vartotojai, kurie naudojosi šia sistema ir anksčiau. Visgi vidutinis ar net aukščiausiu SUS balu įvertinta vizualios programavimo kalbos tinkamumas (*angl. usability*) nesiekia numatyto vidutinio 68% ribos.



7.1 pav. „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos apklausa pagal SUS balą diagrama

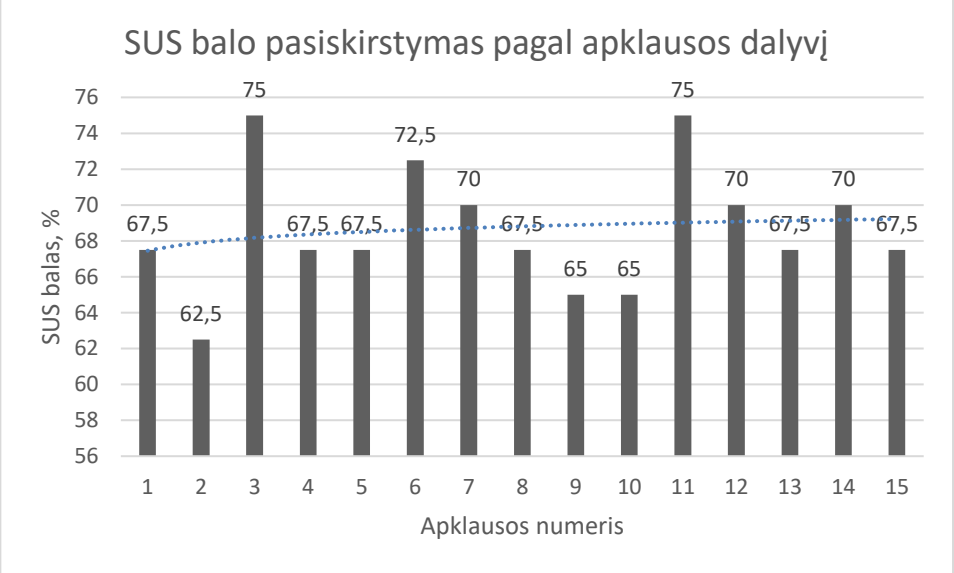
Sukurtos „RobotVL“ vizualios programavimo kalbos apklausos rezultatai pateikti 7.2 lentelėje. Iš lentelės matome, jog blogiausias SUS balas – 62,5%, geriausias SUS balas – 75,5%. Visų apklausų SUS balo rezultatai nusako „RobotVL“ vizualios programavimo kalbos vidutinį 68,67% SUS balą.

7.2 lentelė. „RobotVL“ vizualios programavimo kalbos apklausos rezultatai

RobotVL	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Vid.**
1	3	4	4	3	4	3	4	4	3	3	4	4	4	4	3	3,60
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2,00
3	4	3	5	4	4	5	4	4	4	4	5	5	5	5	5	4,40
4	4	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4,07
5	5	3	4	4	4	4	3	3	3	4	4	3	3	4	3	3,60
6	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	1,07
7	4	5	5	4	4	5	5	5	4	4	5	4	4	4	4	4,40
8	4	3	3	3	4	3	3	3	3	3	3	3	3	3	3	3,13
9	4	4	4	4	4	4	4	4	4	3	4	4	4	3	4	3,87
10	2	3	2	2	2	2	2	3	2	2	2	2	2	2	2	2,13
Balų suma	27	25	30	27	27	29	28	27	26	26	30	28	27	28	27	27,47
SUS balas	67,5	62,5	75	67,5	67,5	72,5	70	67,5	65	65	75	70	67,5	70	67,5	68,67

Sukurtos „RobotVL“ vizualios programavimo kalbos vartotojų apklausos tyrimas pateiktas apibendrintoje 7.2 lentelėje, o šių apklausų duomenų diagrama pateikta 7.2 paveikslėlyje. Iš paveikslėlio matoma, jog yra nemažas SUS balo pasiskirstymas, bet jis kur kas mažesnis nei 7.1 paveikslėlyje esantys duomenys. Kadangi ši vizuali robotų programavimo kalba niekur kitur nebuvo naudojama išskyrus sukurtoje sistemoje. Visi vartotojai pirmą kartą ja naudojosi, be jokios galimos ankstesnės patirties, todėl gautas apklausų duomenų pasiskirstymas kur kas mažesnis nei 7.1 paveikslėlyje esantys duomenys. Vidutinis SUS balas įvertina vizualios programavimo kalbos tinkamumą (*angl. usability*), kuris tik vos viršija 68% ribą. Aukščiausiu net 75% SUS balu įvertino 2

vartotojai, kurie bandė naudotis šia sistema. Visgi tai nauja vizuali robotų programavimo kalba, kuriai reikia atsižvelgti į vartotojų atsiliepimus ir tobulėti, norint sulaukti geresnio SUS balo įvertinimo.



7.2 pav. „RobotVL“ vizualios programavimo kalbos apklausa pagal SUS balą diagrama

7.3. Eksperimento išvados

Eksperimento metu pasiekti šie rezultatai ir nustatytos tokios išvados:

1. Eksperimento metu lygintos vartotojų naudojimosi lengvumas „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualių programavimo kalbų. Iš 15 vartotojų, išbandžiusių abi vizualias programavimo kalbas, gauti apklausų rezultatai parodė, jog vartotojams buvo lengviau naudotis „RobotVL“, nei „Microsoft Robotics Developer Studio“, vizualia programavimo kalba.
2. Eksperimento metu gauti apibendrinti rezultatai parodė, jog „RobotVL“ vizualios programavimo kalbos SUS balas 68,67%. Toks gautas SUS balas yra aukščiau nustatytos vidutinės 68% reikšmės.
3. Eksperimento metu gauti apibendrintų rezultatų palyginimas parodė, jog sukurtos „RobotVL“ vizualios programavimo kalbos SUS balas 68,67% yra didesnis nei „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos SUS balas 38,5%. Dėl šių gautų rezultatų galima teigti, jog sukurta vizualia programavimo kalba lengviau naudotis naujiems vartotojams dėl savo paprastesnės vartotojo sąsajos.
4. Eksperimento metu iš gautų rezultatų matoma, jog vartotojai į klausimus iš galimų 5 galimų balų dažnai įvertino mažiau. Dėl šios priežasties, daroma prielaida, jog vartotojų netenkina esamas vizualios programavimo kalbos sprendimas ir tikisi jo patobulinimo, atsižvelgiant į jų norimus pakeitimus.
5. Atsižvelgiant, jog šios vizualios programavimo kalbos gali būti naudojamos namų vartotojo, kuris neturės užtikrinto techninio žmogaus pagalbos, naudojantis vizualia

programavimo kalba. Į šį klausimą atsako SUS klausimyno 4 klausimas. Visų bandžiusių vartotojų nuomonė sutapo, jog naudojant „Microsoft Robotics Developer Studio“ vizualią programavimo kalbą reikia techninio žmogaus pagalbos, tačiau vartotojų nuomonė išsiskyrė dėl „RobotVL“ vizualios programavimo kalbos. Kai kurių vartotojų nuomone „RobotVL“ vizuali programavimo kalba reikalauja mažiau techninio žmogaus pagalbos, norint ja tinkamai naudotis.

8. IŠVADOS

Darbo metu pasiekti šie rezultatai ir nustatytos tokios išvados:

1. Analizuojant analogiškas vizualias programavimo kalbas nustatyta, jog kiekviena kalba turi savitą mazgų vaizdavimą, todėl nėra vieno aiškaus ir suprantamo būdo juos pavaizduoti. Dėl šios priežasties analizuojami būdai, kokiomis savybėmis turėtų pasižymėti naujai kuria vizuali programavimo kalba, jog būtų galima lengvai ją suprasti ir būtų aiški naujiems vartotojams. Pagrindinis dėmesys skiriamas vizualios programavimo kalbos mazgams, kaip apdorojami duomenys, atliekamas sprendimas, įvedami ar išvedami duomenys, bei kokia yra galimybė juos sujungti.
2. Atsižvelgiant į tai, jog plečiant vizualios programavimo kalbos „RobotVL“ funkcionalumą, ją būtų galima panaudoti ir kitų sistemų kūrimui, buvo nuspręsta kurti atskirą kodo transformavimo posistemę, gebančią transformuoti vizualų programavimo kodą į C# programavimo kalbos ir kitų objektinių programavimo kalbų programinį kodą.
3. Atlikus algoritmų užrašymą „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualiomis programavimo kalbomis eksperimentą nusatyta, jog „RobotVL“ vizualiai programavimo kalbai tam pačiam algoritmui užrašyti reikia daugiau mazgų lyginant su „Microsoft Robotics Developer Studio“ vizualia programavimo kalba. Visgi iš „RobotVL“ suformuotas C# programinis kodas yra mažesnės apimties, nei „Microsoft Robotics Developer Studio“, tačiau įvertinus kitas metrikas buvo aptikta, jog vidutinio gylio, maksimalaus gylio ir maksimalaus sudėtingumo metrikų reikšmės per didelės lyginant su rekomenduojamu. Tyrimas parodė, jog reikalingas transformuojamo kodo posistemės optimizavimas, jog galėtų atitikti rekomenduojamas programinio kodo metrikų reikšmes.
4. Atlikus vartotojų naudojimosi lengvumo „RobotVL“ ir „Microsoft Robotics Developer Studio“ vizualiomis programavimo kalbomis tyrimą nustatyta, jog „RobotVL“ vizuali programavimo kalba bandžiusiems vartotojams buvo patogesnė. Apibendrinus eksperimento rezultatus su standartizuotais SUS klausimais gautas 68,67% SUS balas, kuris yra daug aukštesnis nei lygintos „Microsoft Robotics Developer Studio“ vizualios programavimo kalbos 38,5% SUS balas. Tyrimas parodė, jog sukurta vizuali programavimo kalba „RobotVL“ yra patogesnė naudojimui. Visgi tai pirmasis vartotojų naudojimosi tyrimas, todėl gauta pastabų kaip vizualią programavimo kalbą padaryti dar patogesnę, jog ją bandantys vartotojai vertintų geriau. Atsižvelgus į šias pastabas „RobotVL“ vizuali programavimo kalba toliau bus tobulinama, nepamirštant daryti pakartotinius tokius vartotojų naudojimosi tyrimus.
5. Ateityje planuojama vizualią programavimo kalbą papildyti nauju funkcionalumu, kuris robotui modeliavimo aplinkoje suteiktų didesnes galimybes. Taip pat atsižvelgiant į darytus

eksperimentus tobulinti vizualią programavimo kalbą, jog pagerėtų tiek iš jos suformuoto C# programinio kodo metrikų reikšmės, tiek padidėtų vartotojų naudojimosi lengvumo SUS balas. Šie tyrimai vizualios programavimo kalbos tobulinimo etape numatomi daryti kas tam tikrą laiką, jog būtų galima stebėti kaip daromi pakeitimai vizualioje programavimo kalboje turi įtakos kitiems gaunamiems iš eksperimentų rezultatams.

9. LITERATŪRA

[1] Yasmin B. Kafai, Quinn Burke, Phi Delta Kappan, „Computer Programming Goes Back to School,“ *Education Week,* 05 09 2013.

[2] J.C., Olabe, M.A. Olabe, X. Basogain, C. Castario, „Programming and Robotics with Scratch in Primary Education,“ įtraukta *Education in a technological world: communicating current and emerging research and technological efforts*, Formatex, 2011, pp. 356-363.

[3] E. Asipauskas, „Virtualių 3D aplinkų tyrimas ir taikymas mokymesi,“ Kaunas, 2011.

[4] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, Yasmin Kafai, „Scratch: Programming for All,“ *Communications of the ACM,* t. 52, nr. 11, pp. 60-67, 2009.

[5] J. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond, „The scratch programming language and environment,“ *ACM Transactions on Computer Education,* t. 10(4), nr. 16, p. 1–15, 2010.

[6] K. A. Nguyen, „A case study on the usability of NXT-G programming language,“ įtraukta *Proc. of 23rd Conf. in Psychology of Programming*, 2011.

[7] I. Plauska, R. Lukas, R. Damasevicius, „Reflections on Using Robots and Visual Programming Environments for Project-Based Teaching,“ *Elektronika ir Elektrotechnika,* t. 20, nr. 1, pp. 71-74, 2014.

[8] J.S. Cepeda, L. Chaimowicz, R. Soto, „Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics,“ įtraukta *Latin American Robotics Symposium and Intelligent Robotics Meeting*, 2010.

[9] Kyle Johns, Trevor Taylor, Professional Microsoft® Robotics Developer Studio, Birmingham, UK: Wrox Press, 2008.

[10] Shih-Chung Kang, Wei-Tze Chang, Kai-Yuan Gu, Hung-Lin Chi, Robot Development Using Microsoft, Taylor & Francis, 2011.

[11] D. Harel, „On Visual Formalisms,“ *Communications of the ACM,* t. 31, nr. 5, pp. 514-530, 1988.

[12] D. L. Moody, „The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering,“ *IEEE Transactions on software engineering*, t. 35, nr. 5, pp. 756-779, 2009.

[13] Jill H. Larkin, Herbert A. Simon, „Why a Diagram is (Sometimes) Worth,“ *Cognitive Science,* t. 11, nr. 1, pp. 65-99, 1987.

[14] M. Binkis, „Objektinės vizualios scenarijų kalbos tyrimas ir kūrimas,“ *[Technologija],* p. 31, 2012.

[15] David Canfield Smith, Allen Cypher, Larry Tesler, „Novice Programming Comes of Age,“ *Communications of the ACM,* t. 43, nr. 3, pp. 75-81, 2000.

[16] G. Shanks, E. Tansley ir R. Weber, „Using Ontology to Validate Conceptual Models,“ *Communications of the ACM,* t. 46, nr. 10, pp. 85-89, 2003.

[17] D. Moody ir J. v. Hillegersberg, „Evaluating the Visual Syntax of UML: An Analysis of the Cognitive Effectiveness of the UML Family of Diagrams,“ įtraukta *Software Language Engineering*, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science (LNCS), 2008, pp. 16-34.

[18] J. Wanner, *SourceMonitor Help V3.5,* Campwood Software, 2014.

[19] J. Brooke, „SUS - A quick and dirty usability scale,“ UsabilityNet.org, [Tinkle]. Available: http://www.usabilitynet.org/trump/documents/Suschapt.doc. [Kreiptasi 05 04 2017].

[20] J. Sauro, „Measuring Usability with the System Usability Scale (SUS),“ 02 02 2011. [Tinkle]. Available: https://measuringu.com/sus/. [Kreiptasi 10 04 2017].

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

[22] 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.