Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Programovanie robotov pomocou stavových automatov

Bakalárska práca

2024
Tomáš Vikiszály

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Programovanie robotov pomocou stavových automatov

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2024
Tomáš Vikiszály

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Tomáš Vikiszály
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Programovanie robotov pomocou stavových automatov
Programming Robots Using Finite State Machines

Anotácia: Robotické modely zo stavebnice Spike Prime sa programujú v troch rôznych jazykoch: IconBlocks – určený najmä pre prvý stupeň ZŠ, WordBlocks – určený najmä pre druhý stupneň ZŠ, Python – určený najmä pre pokročilých používateľov. Problém je, že ani jeden nie je ideálny na modelovanie správania robota: IconBlocks: neobsahuje ani podmienky, WordBlocks: obsahuje podmienky, cykly, a (bohužiaľ iba globálne) udalosti, Python: texový jazyk a preto neprehľadný a nevhodný na modelovanie správania. Najprirodzenejším modelom správania je stavový automat (state machine), pretože robot počas riešenia úlohy prechádza cez rozličné stavy/fázy – napr. stav1: je na čiare, stav2: hľadá pokračovanie čiary po jej prerušení, stav3: prechádza cez tunel, stav4: obchádza prekážku, ale aj jednotlivé stavy môžu o úroveň abstrakcie nižšie obsahovať stavové automaty, napr. kým je na čiare, tak sa mení stav medzi dvoma stavmi: hranavpravo: nachádza sa naľavo od hrany, hranavľavo: nachádza sa napravo od hrany. Čiže každý stav môže byť vnútri znova stavový automat a naopak, celkové správanie automatu možno považovať za jeden makrostav v stavovom automate na vyššej úrovni abstrakcie. Na každom stavovom prechode je určená udalosť, kedy k nemu dochádza, pričom virtuálne udalosti môže automat generovať aj sám. V jednotlivých stavoch a na stavových prechodoch možno štartovať aj tradične zapísaný procedurálny kód (task), pričom tasky môžu interagovať, bežať aj v pozadí stavového automatu alebo byť automaticky ukončené pri zmene stavu.

Cieľ: Úlohou študenta bude naprogramovať a na netriviálnych ukážkach otestovať nový grafický programovací jazyk pre Spike Prime založený na stavových automatoch.

Literatúra:
R. A. Brooks, "A robot that walks; emergent behaviors from a carefully evolved network," in Proceedings, 1989 International Conference on Robotics and Automation, pp. 692-4+2 vol.2, 1989.
R. Balogh, D. Obdržálek, “Using Finite State Machines in Introductory Robotics”, in: Robotics in Education. RiE 2018. Advances in Intelligent Systems and Computing, vol 829. Springer, 2019.
R. Ghzouli et al. "Behavior Trees and State Machines in Robotics Applications," in IEEE Transactions on Software Engineering 49 (9), Sept. 2023.

Kľúčové slová: konečný automat, programovanie robotov, robotika vo vzdelávaní

96368802

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 10.10.2023

Dátum schválenia: 16.10.2023

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

.....

študent

.....

vedúci práce

Čestné prehlásenie

Čestne vyhlasujem, že celú bakalársku prácu na tému „Programovanie robotov pomocou stavových automatov“, vrátane všetkých jej príloh a obrázkov, som vypracoval samostatne, a to s použitím literatúry uvedenej v priloženom zozname.

Poďakovanie: Cítim obrovskú vďačnosť voči môjmu školiteľovi, ktorým bol Mgr. Pavel Petrovič, PhD, za všetku pomoc, podporu a hodiny so mnou strávene v laboratóriu. Vďačný som aj svojej rodine za podporu a povzbudenia, ktoré som počas svojho celého štúdia od nich dostával.

Abstrakt

Táto bakalárska práca sa venuje návrhu a implementácii novej metódy programovania robotických stavebníc Spike Prime od spoločnosti LEGO Education. Novou metódou je používanie stavových automatov, ktoré jasne popisujú správanie robota v rôznych situáciách a jeho reakcie na udalosti závislé od kontextu. Ich grafická reprezentácia je prehľadná a samovysveľujúca, čo nie vždy platí o kóde zapísanom pomocou blokového jazyka WordBlocks, kde môžu byť zadefinované iba udalosti s globálnou platnosťou, alebo pomocou textového zdrojového kódu, kde správanie robota nie je vizualizované vôbec. Návrh a implementácia systému je rozdelená do niekoľkých častí. Prvou časťou je grafická aplikácia, naprogramovaná v C#, v ktorej môže používateľ vytvárať a meniť stavové automaty. Druhou je webové prostredie, ktoré slúži na komunikáciu medzi počítačom a robotom a vzniklo modifikáciou prostredia pre systém PyBricks. Používateľom vytvorené automaty sa do Prime Hub nahrávajú cez konzolu prostredia po ich preložení do vnútornej reprezentácie. Treťou časťou bol nový firmware pre riadiacu jednotku Prime Hub, ktorý vznikol úpravou firmware k systému PyBricks, jeho rozšírením o čítanie a vykonávanie stavových automatov. Túto funkcionalitu sme implementovali a podrobili testovaniu, aby sme sa uistili, že implementácia nemá chyby. Naprogramovali sme niekoľko stavových automatov ako ukážku, ako takéto programy vyzerajú.

Kľúčové slová: Konečný automat, programovanie robotov, robotika vo vzdelávaní

Abstract

This bachelor's thesis deals with the design and implementation of a new method of programming Spike Prime robotic kits from LEGO Education. A new method is the use of state machines that clearly describe the robot behavior in different situations and its context-dependent reactions to events. Their graphical representation is clear and self-explanatory, which is not always the case when source code is expressed using blocks, where only global-events can be specified, or the source code is a plain text program with no graphical visualisation at all. Design and implementation of the system consists of several parts. The first part is a graphical application written in C# in which the user can create and edit state machines. The second is the web environment, which is used for communication between the computer and the robot. It was developed by modifying the PyBricks web environment. State machines programmed by the user are uploaded to Prime Hub through its console window after they are compiled to an internal representation. The third part was the new firmware for Prime Hub control unit. It was developed by modifying the PyBricks firmware so that it can read and execute the state machines. We implemented and tested this functionality to ensure that the implementation is without bugs. We programmed several state machines as a demonstration of how such programs looks like.

Keywords: Finite state machines, robot programming, robotics in education

Obsah

Úvod **1**

1 Východiská **3**

1.1 Stavebnice LEGO Education . . . 3

1.1.1 Nultá generácia . . . 3

1.1.2 Prvá generácia . . . 5

1.1.3 Druhá generácia . . . 5

1.1.4 Tretia generácia . . . 7

1.1.5 Štvrtá generácia . . . 8

1.2 Jazyky na programovanie robotov . . . 8

1.2.1 Konečné stavové automaty . . . 10

1.3 LEGO Spike Prime . . . 12

1.3.1 Programátorské prostredia . . . 14

1.4 Programovanie grafickej aplikácie v C# . . . 16

1.4.1 Serializácia v aplikáciách C# . . . 17

2 Špecifikácia a ciele práce **19**

2.1 Softvér na tvorbu automatov . . . 19

2.2 Nový firmvér . . . 19

2.3 Úprava webovej aplikácie . . . 20

3 Návrh **21**

3.1 Návrh modelu stavových automatov . . . 21

3.2 Návrh modelu jazyka pre akcie v automatoch . . . 23

3.2.1 Vzorové programy . . . 23

3.3 Návrh aplikácie . . . 27

3.4 Návrh firmvéru . . . 28

3.5 Návrh úprav webového prostredia . . . 29

4 Implementácia **31**

4.1 Čítanie dát zo senzorov . . . 31

4.2 Ovládanie motorov 31
4.3 Bluetooth a komunikácia 32

5 Výsledky **33**
5.1 Ukážky hotových programov 33
5.1.1 Aplikácia na tvorbu automatov 33
5.1.2 Webová stránka 35
5.1.3 Firmvér 35
5.2 Testovanie 36

Záver **39**

Príloha A **43**

Zoznam obrázkov

1.1 Ukážka programu LEGO DACTA 4
1.2 Ukážka programu LEGO RoboLab 4
1.3 Ukážka dvoch programov v Robotics Invention System 5
1.4 Ukážka programu LEGO RoboLab 6
1.5 Ukážka robota LEGO NXT-G 6
1.6 Ukážka programu LEGO NXT-G 7
1.7 Ukážka programu LEGO EV3 8
1.8 Ukážka modelu zo stavebnice LEGO EV3 8
1.9 Ukážka programu v jazyku Scratch 9
1.10 Ukážka modelu zo stavebnice LEGO Spike Prime 9
1.11 Obrázok popisujúci *subsuption architecture* 12
1.12 Program pohybu robota, ktorý prenasleduje objekt pred sebou 13
1.13 Jednoduchý robot s ultrazvukovým senzorom 14
1.14 Stavový automat pre pohyb robota podľa vzdialenosti 14

3.1 Návrh grafickej reprezentácie stavového automatu 22
3.2 Návrh aplikácie pre tvorbu automatov 28

5.1 Ukážka automatu 34
5.2 Ukážka netriviálneho automatu 34
5.3 Webové prostredia počas behu automatu 35
5.4 Fotografia z testovania 36
5.5 Fotografia z programu z testovania 37
5.6 Fotografia z testovania programu 38

Zoznam kódov

1.1 Automatický generovaný kód po vytvorení novej aplikácie C# 17
1.2 Automatický generovaný kód pre grafické komponenty v C# 17
3.1 Ukážka modelu jazyka 23
3.2 Program pre sledovanie objektu 25
3.3 Program pre prácu s gyroskopom a LED svetelným displejom 26

Úvod

Spôsobov, ako programovať robotov je niekoľko. Pre robotické stavebnice Spike Prime do spoločnosti LEGO Education sa sem radí programovací jazyk Python, vhodný pre najskúsenejších používateľov, vizuálny programovací jazyk WordBlocks založený na princípoch populárneho jazyka Scratch a ikonografický jazyk IconBlocks určený pre najmladších používateľov. Hoci jazyky majú jednoduchú syntax a ľahko pochopiteľné funkcie, problémom je, že ak sa používajú na programovanie viac alebo aj menej komplexnejšieho správania robota, programová štruktúra nezodpovedá priamo správaniu robota a preto takýto zápis správania robota nie je dostatočne priehľadný a zrozumiteľný a preto sa v ňom ťažšie hľadajú chyby. Navyše, ak sa programátor vráti k tomuto programu po nejakom dlhšom čase, potrvá mu nejakú dobu, kým pochopí, čo program robí.

Riešením tohto problému by bolo vytvoriť nový spôsob, ako programovať roboty. Pri tom ale dodržať zásady, že tento spôsob bude jednoduchý a prehľadný. Jedným z dobrých spôsobov, ktorý by dodržal tieto zásady, by bolo použitie stavových automatov. Stavové automaty sú jednoduché na pochopenie, sú prehľadné a ich grafická vizualizácia je zároveň aj dokumentačným materiálom a predovšetkým, keďže sú vhodné na modelovanie správania robota, našou hypotézou je, že budú vhodné aj na jeho programovanie – a to by sme vďaka výsledkom tejto práce chceli umožniť otestovať.

Cieľom tejto bakalárskej práce je zanalyzovať možnosti, navrhnúť riešenie, implementovať ho a následne ho otestovať a takto ukázať, že stavové automaty sú vhodným spôsobom ako programovať správanie robotov. Práca sa člení na niekoľko častí a to na:

1. Zbieranie a zosumarizovanie všetkých potrebných informácií, ktoré budeme potrebovať pre návrh riešenia. Patrí sem špecifikácia stavebnice, popis aktuálneho firmvéru ako aj popis nástrojov, ktoré budeme pre implementáciu používať.

2. Návrh riešenia, ktorý vznikne na základe informácií, ktoré sme získali. Pri návrhu udržať riešenie jednoduché a prehľadné ale efektívne a robustné, aby ho mohol využívať ako žiak tak aj skúsený programátor.

3. Implementáciu návrhu vo všetkých častiach. Tvorbu aplikácie na kreslenie stavových automatov, úpravu webového prostredia pre správnu komunikáciu medzi

počítačom a robotom a samotný firmvér, ktorý bude naše stavové automaty vedieť čítať a vykonávať.

4. Otestovanie aplikácie, aby naše riešenie neobsahovalo chyby a neželané správanie.

Kapitola 1

Východiská

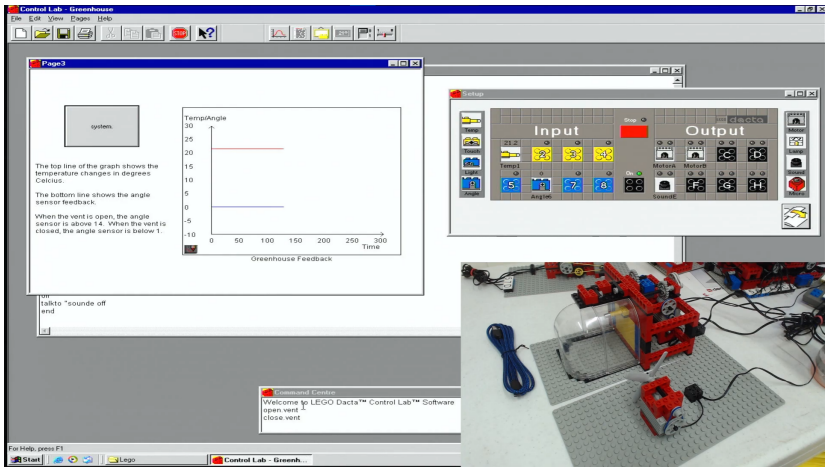
V tejto kapitole sa budeme venovať teoretickej časti tejto bakalárskej práce. Budeme v nej popisovať stavebnice LEGO Education, ich históriu a postupný vývoj, možnosti programovania týchto stavebníc, výhody a problémy takéhoto programovania a programátorské prostredia. Popíšeme ďalšie možnosti, ako programovať tieto stavebnice, kde spomenieme konečné stavové automaty, aktuálny firmvér a tvorbu grafickej aplikácie v jazyku C#.

1.1 Stavebnice LEGO Education

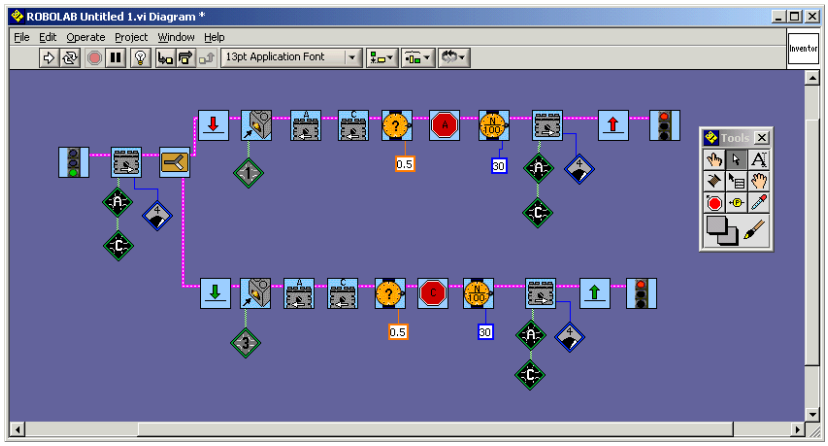
Prvé robotické stavebnice od spoločnosti LEGO Education vznikli v osemdesiatych rokoch minulého storočia a sú určené pre žiakov základných a stredných škôl. Hoci hračkárska divízia na trhu umiestňovala stavebnice s bojovými robotmi a základnými súčiastkami pre chlapcov, vzdelávacia divízia vždy vyrábala voči pohlaviu neutrálne stavebnice. Jednoduché motorčeky na jednosmerný prúd boli postupne vybavené zabudovanými otáčkovými senzormi, pribudli gyroskopy, ultrazvukové i farebné senzory, rôzne verzie displejov, komunikácia cez BlueTooth a ďalšie komponenty. Školy ich využívali na vzdelávanie informatiky, robotiky a programovania pomocou hry, kreativity a objavovania, k čomu LEGO Education pripravuje a neustále vylepšuje didaktické materiály.

1.1.1 Nultá generácia

Prvou skutočne robotickou stavebnicou bola stavebnica LEGO Dacta Control Lab. Táto generácia sa programovala na počítačoch v jazyku Logo pomocou softvéru Control Lab. V tomto softvéri sa vytvoril kompletný program, ktorý mohol vizualizovať informácie na displeji počítača a bol spúšťaný priamo v počítači. Stavebnica neobsahovala žiadne CPU, pamäť ani batériu, len hardvér pozostávajúci z jednosmerných motorčekov, žiaroviek a senzorov. Nevýhodou bolo to, že robot musel byť neustále pripojený



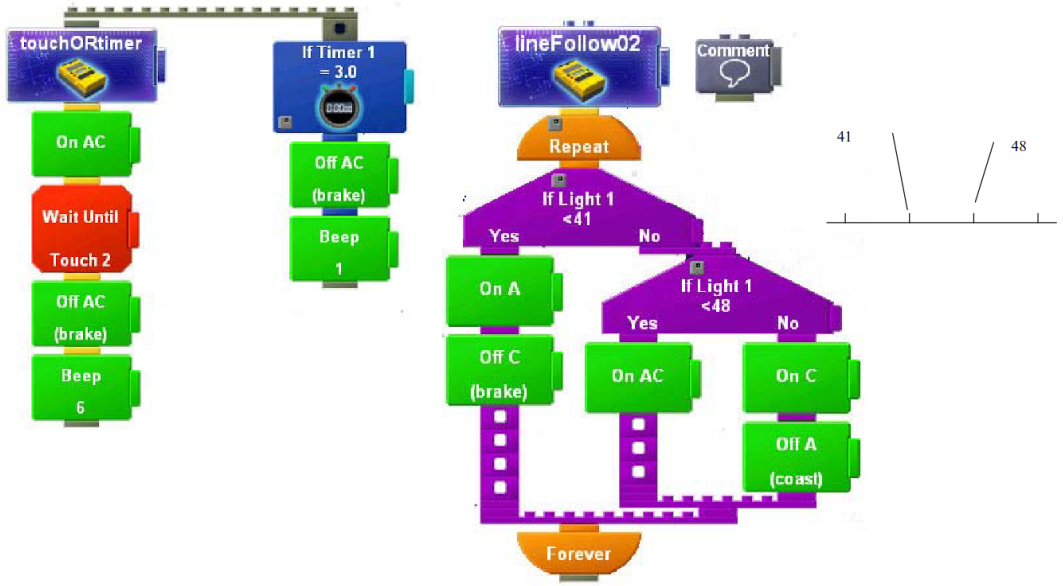
Obr. 1.1: Na obrázku je možné vidieť softvér Control Lab a stavebnicu LEGO Dacta Control Lab. Zdroj obrázka je snímok obrazovky z videa [2].



Obr. 1.2: Ilustračný obrázok programu v LEGO RoboLab. Prevzaté z [3].

k počítaču, čo obmedzovalo rozsah pohybov robota, čo ale zároveň bolo výhodou, že všetky dáta si mohol užívateľ nechať vykresľovať alebo vypisovať na obrazovku a robota riadiť interaktívne, spúšťať jednotlivé príkazy, jednoducho ladiť a testovať časti svojho programu. Ukážky môže čitateľ nájsť na tejto stránke [1].

Alternatívou k softvéru LEGO Dacta bol vytvorený softvér založený na jazyku LabView s názvom LEGO RoboLab. Na rozdiel od textového kódu v LEGO Dacta tento softvér bol grafický postavený na takzvanom toku dát. Môžeme si to predstaviť ako súbor nezávislých firiem, ktoré spracujú materiál okamžite po tom, čo k ním príde a hneď ho posielajú ďalej, do ďalšej firmy. Tým je zabezpečená rýchlosť behu programu a preto je tento jazyk veľmi populárny medzi fyzikmi programujúcimi senzory.



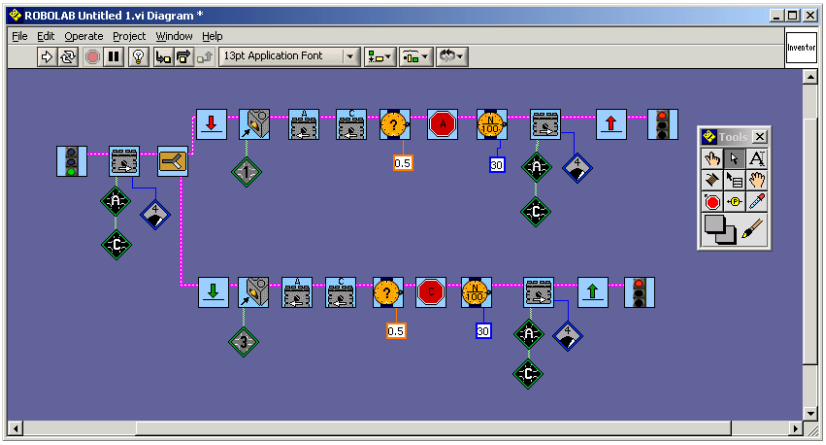
Obr. 1.3: Ukážka dvoch programov v Robotics Invention System. Globálne udalosti mohli reagovať na stavy senzorov alebo časovač. Vetvenie programu inšpirované nástrojmi CASE zo softvérového inžinierstva z minulého tisícročia - rovnaká myšlienka ako neskôr využil jazyk Scratch. Prevzaté z [16].

1.1.2 Prvá generácia

Podstatná zmena nastala v roku 1998, keď spoločnosť LEGO vydala stavebnicu LEGO MINDSTORMS Robotics Invention Kit so známou žltou kockou RCX, ktorá už bola plnohodnotným počítačom programovateľným cez proprietárny IR port a teda umožnila konštrukciu autonómnych mobilných robotov. Bolo k nej možné pripojiť tri výstupné a tri vstupné zariadenia. Programovala sa v jazyku, ktorý by sme mohli považovať za predchodcu jazyka Scratch, lebo jeho štruktúra bola veľmi podobná. Zaujímavou črtou boli globálne udalosti - podmienky, ktoré sa počas behu programu neustále kontrolovali a pri ich splnení sa odštartovala zodpovedajúca postupnosť kódu, príklad je zobrazený na obr. 1.3.

1.1.3 Druhá generácia

Keďže spoločnosť LEGO Education zaznamenala so stavebnicou úspech, tak pokračovali vo výskume a na trh priniesli novú generáciu stavebnice, čo popisuje aj jej názov, LEGO MINDSTORMS Next Gereration, skrátene NXT-G. Rozdielov bolo niekoľko. Niektoré boli malé a iné opäť veľké. Bezdrôtové pripojenie cez IR port nahradilo pripojenie cez BlueTooth ale zároveň možnosť pripojiť robota cez USB kábel. Bohužiaľ, rovnako ako s RCX, ani s NXT-G nemohli žiaci pracovať interaktívne zo svojho počítača - zobrazovať si pomocné textové, alebo grafické výstupy, umiestňovať na plochu



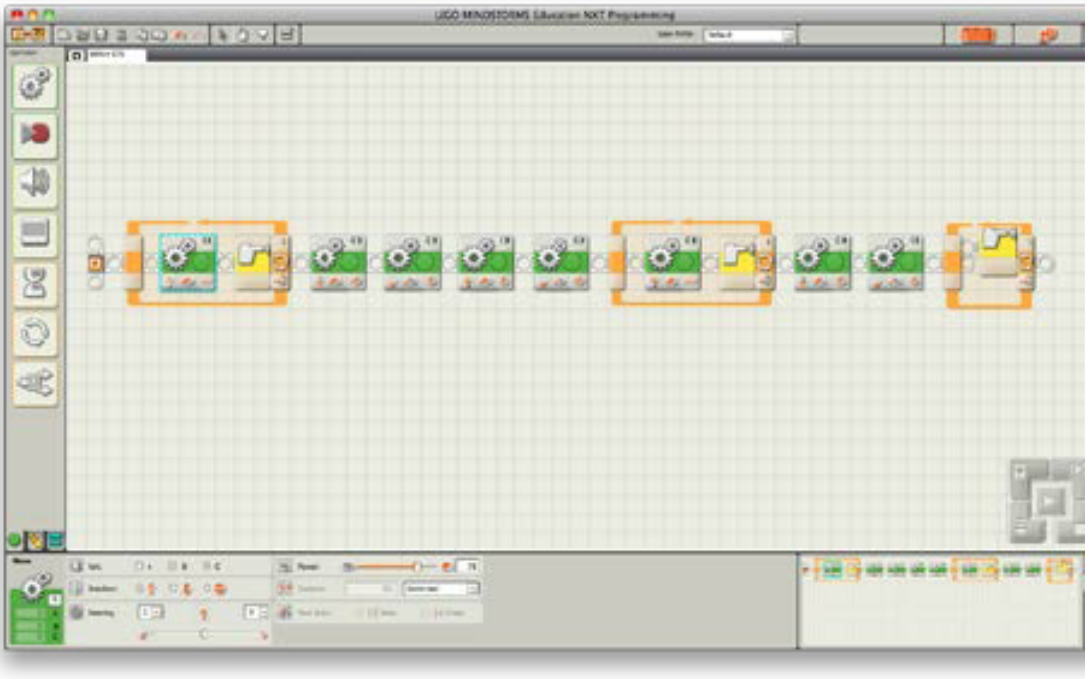
Obr. 1.4: Ilustračný obrázok programu v LEGO RoboLab. Prevzaté z [3].

Obr. 1.5: Robot LEGO MINDSTORMS NXT-G. Prevzaté z [5].

tlačidlá alebo iné prvky, ktorými sa model riadil ako to bolo v systéme Control Lab. Namiesto toho nastúpila paradigma Program - Download - Run, ktorej výsledkom je náročnejšie ladenie robotov.

Na programovanie slúžil nový grafický softvér, ktorého jadro bolo postavené na systéme LabView, alternatívou bolo programovanie v samotnom LabView pre pokročilých. Tento softvér už nebol textový jazyk (aj keď zaobalený do dielikov puzzle), ale plnohodnotne ikonografický jazyk. Program pozostával z prepojených ikon jednotlivých príkazov a po rozkliknutí nejakého bloku sa otvorila paleta, kde si žiak mohol navoliť parametre, s ktorými bude príkaz pracovať. Keď ale chcel používateľ zdieľať svoj program v grafickej podobe, tak to bolo veľmi nepraktické, lebo by musel odfotiť rozkliknutý každý blok samostatne, lebo bez parametrov by druhý užívateľ nemal žiadaný výsledok.

Počas produkcie tejto generácie sa aj na Slovensku začal usporadúvať program First



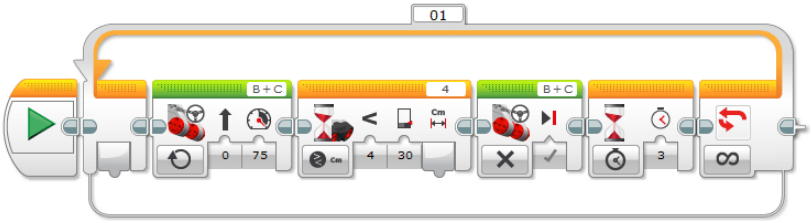
Obr. 1.6: Program pre LEGO MINDSTORMS NXT-G. Prevzaté z [3].

LEGO League, ktorý je pre deti vo veku od 4 až 16 rokov formou súťaže, ktorá má u účastníkov podporovať kritické a tvorivé myslenie pri riešení reálnych problémov z nášho sveta. FLL sa v USA začalo organizovať ešte v roku 1998, keď vyšla prvá verzia LEGO MINDSTORMS RCX a zrejme je stále najväčšou robotickou súťažou na svete.

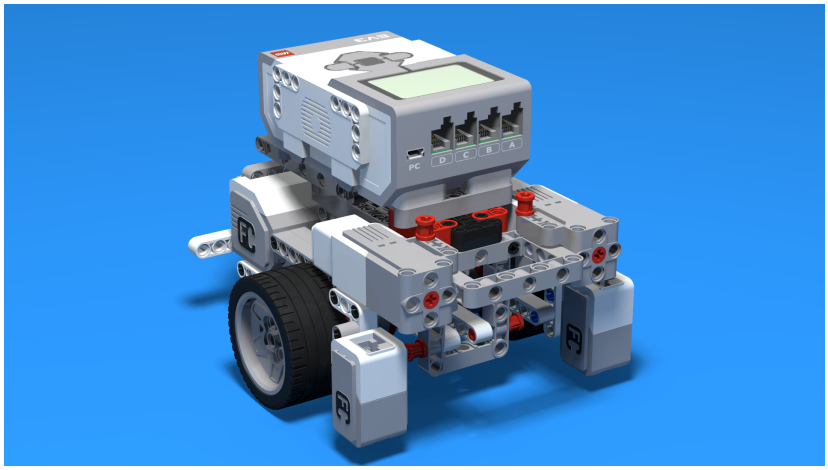
1.1.4 Tretia generácia

Tretia generácia nesie meno LEGO MINDSTORMS Evolution, alebo skrátene LEGO EV3. Táto verzia robota dovoľovala takzvané datalogovanie informácii, programovanie cez dátový kábel a Bluetooth, či za použitia iPad alebo Android zariadenia, notebookov, či desktopov. Bola programovaná vo vynovenom grafickom systéme, ktorý používala druhá generácia ale všetky informácie a nastavenia jednotlivých blokov programu boli viditeľné bez ďalšieho klikania. Ukážka programu je na obrázku 1.7. Disponovala oveľa väčším výpočtovým výkonom a jej počítač bežal na operačnom systéme Linux - čo zároveň spôsobovalo dlhý štart a vypínanie. Podporovala aj externý USB modul pre komunikáciu cez WiFi. Na obrázku 1.8 je ukážka robota zloženého z komponentov tejto generácie.

Od roku 2003, kedy vznikol jazyk Scratch na MIT sa postupne rozšíril a získal si priazeň používateľov i didaktikov informatiky. Je populárny pre svoju nenáročnosť a jednoduchosť. Dokonca existoval doplnok na programovanie LEGO robotov cez klasický Scratch. To viedlo aj spoločnosť LEGO Education, aby vydali novú verziu prostredia na programovanie svojich robotov v jazyku, ktorý vychádzal zo syntaxe a vizuálnej podoby



Obr. 1.7: Program pre LEGO MINDSTORMS EV3. Prevzaté z [3].



Obr. 1.8: Ukážka modelu zo stavebnice LEGO MINDSTORMS EV3. Prevzaté z [4].

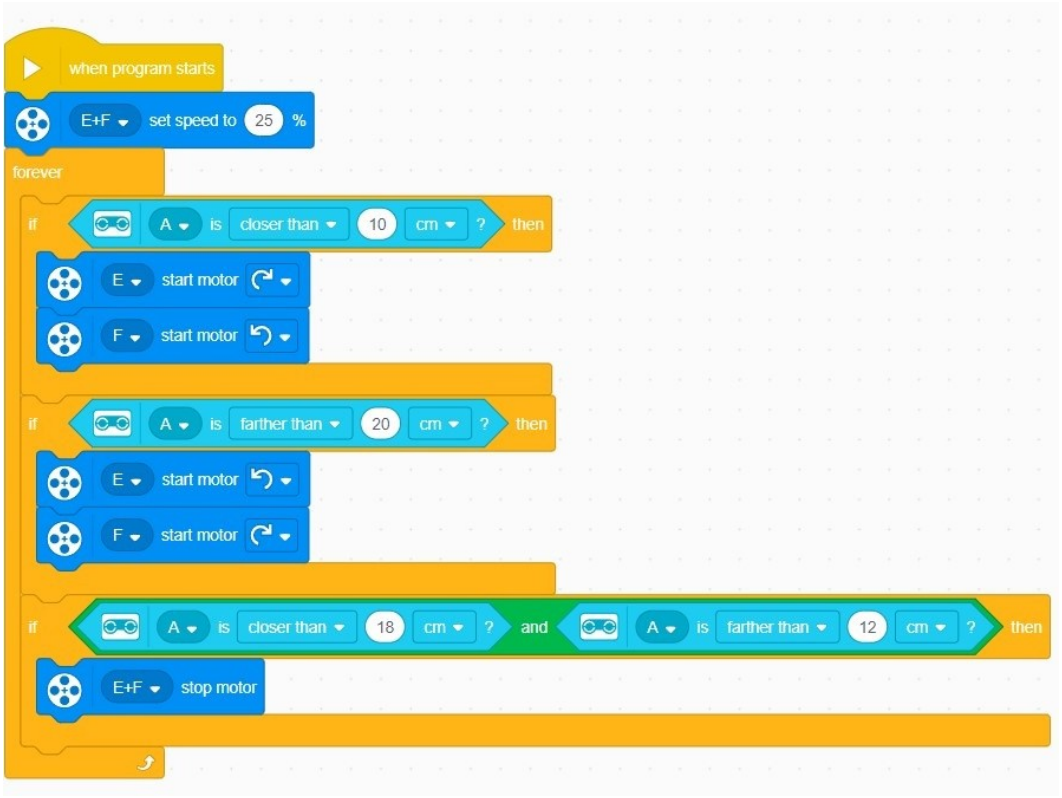
jazyka Scratch. Na popularite medzi žiakmi narástol aj jazyk Python, ktorý bol tiež zakomponovaný medzi možnosti, ako programovať stavebnice LEGO Education. Na obrázku 1.9 je ukážka programu v jazyku WordBlocks pre neskoršie stavebnice Spike Prime, ale aj pre EV3 vydalo LEGO Education podobný softvér a programovací jazyk.

1.1.5 Štvrtá generácia

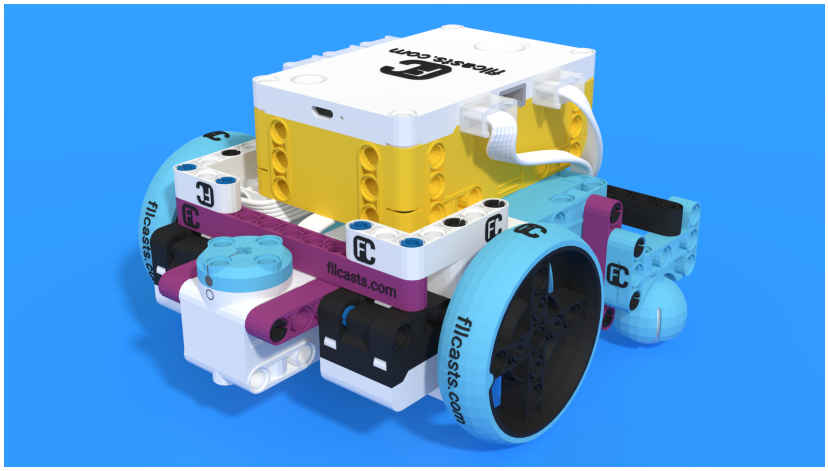
Táto generácia nesie meno LEGO Spike Prime. Líši sa tým, že nemá displej, porty sú univerzálne, čiže už nie sú vyhradené iba pre motor alebo senzor, má 5x5 LED maticu a bezchybný gyroskop v troch osiach. Na programovanie slúžia už len jazyky Scratch a Python. Robot je programovateľný cez dátový kábel alebo použitím Bluetooth a nejakú dobu disponoval aj modulom pre WiFi ako EV3, ktorý nakoniec tiež vypli. Vzorový robot zložený z komponentov stavebnice je na obrázku 1.10.

1.2 Jazyky na programovanie robotov

Ako sme spomenuli v úvode, pre poslednú generáciu robotických stavebníc LEGO Education Spike Prime LEGO Education ponúka tri jazyky – Icon Blocks - bez textov, pre žiakov 1. stupňa, Word Blocks – podobný jazyku Scratch a Python – plne textový



Obr. 1.9: Ukážka programu v jazyku Scratch. Program som vytvoril v systéme LEGO Education Spike.



Obr. 1.10: Ukážka modelu zo stavebnice LEGO Spike Prime. Prevzaté z [4].

programovací jazyk. Interpretované jazyky, ako napr. jazyk Python, sú vhodné pre začiatočníkov ako úvod do programovania, ale nie sú vhodné pre nízkoúrovňové programovanie robotov, kde nám záleží na každej milisekunde. Každá zbytočná operácia, ktorú musí riadiaca jednotka urobiť navyše, nás len vzďaľuje od toho, čo sme v programe napísali. Python je veľmi dynamický, alokuje a dealokuje si pamäť kedykoľvek to on uzná za vhodné a vtedy je v rýchlosti nekonzistentný. Prvýkrát mu nejaká operácia trvá určitý počet milisekúnd a keď tú istú operáciu zavoláme o chvíľu neskôr, tak to môže trvať aj niekoľko násobne dlhšie, lebo práve v tej chvíli spustil vnútorný *garbage collector*. Ďalšou nevýhodou je, že programátor musí ovládať celý rad príkazov z knižnice, ktorá mu sprístupňuje ovládanie hardvéru. Jazyk Scratch môže byť nepraktický pre programovanie zložitejších robotických úloh a podobne ho obmedzuje rýchlosť ako Python, keďže jeho kód sa tiež kompiluje do MicroPythonu. Pri programovaní robotov je potrebné naprogramovať rozhodovanie pre rôzne situácie, v ktorých sa bude robot nachádzať. To pri lineárnom zápise programového toku alebo pri využití globálnych udalostí - oboje sú typické charakteristiky scratchového jazyka Word Blocks, môže byť zápis neprehľadný, komplikovaný, chybný a ťažko zrozumiteľný.

1.2.1 Konečné stavové automaty

Koncept stavových automatov sa postupne vyvinul od uvedenia Turingových strojov v roku 1936 Alanom Turingom. Konečné stavové automaty (finite-state automaton, často označovaný aj ako finite-state machine) podrobne analyzovali J. D. Ullman a J. E. Hopcroft v knihe Úvod do teórie automatov, jazykov a výpočtov [8]. Konečný automat (KA) pozostáva z konečnej množiny stavov a množiny prechodov zo stavu do stavu, ktoré sa uskutočňujú, ak je na vstupe skonzumovaný symbol zo vstupnej abecedy. Pre každý vstupný symbol existuje najviac jeden prechod z každého stavu (možno aj späť do toho istého stavu). Jeden stav, zvyčajne označený g_0, je počiatočný stav, v ktorom automat začína. Niektoré stavy sú označené ako koncové alebo akceptačné stavy. KA sa dá vizualizovať súvislým orientovaným grafom, nazývaným prechodový diagram, nasledovne. Vrcholy grafu zodpovedajú stavom KA. Ak existuje prechod zo stavu q do stavu p na vstupe a, potom v prechodovom diagrame existuje orientovaná hrana označená a spájajúca stav q so stavom p. KA akceptuje reťazec x, ak sled prechodov zodpovedajúcich symbolom x vedie z počiatočného stavu do akceptačného stavu.

Ak ale chceme, aby robot okrem pozorovania svojho prostredia a reagovania zmenou svojho stavu vykonával aj nejakú aktivitu, potrebujeme miesto, kam to zapísať. Tomuto popisu vyhovuje konečný stavový prevodník (finite-state transducer). Tento automat má nielen vstupnú pásku, ale aj výstupnú. Symboly zapisované na výstupnú pásku môžeme interpretovať ako akcie, ktoré má robot vykonávať. Stručná definícia sa nachádza napr. v článku [9]. Konečný stavový prevodník $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ je

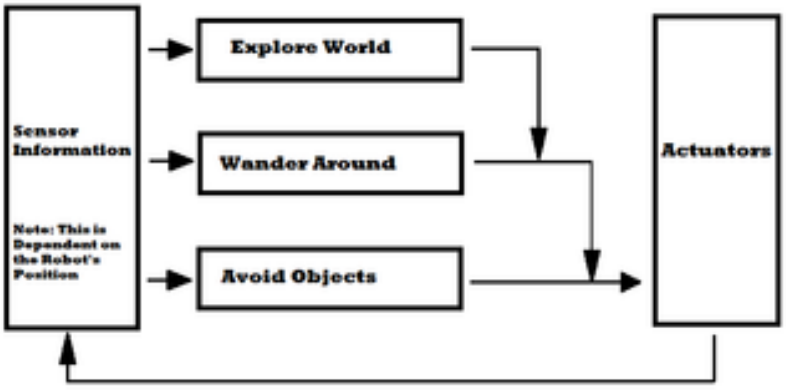
8-tica, kde:

- Σ je konečná vstupná abeceda prevodníka,
- Δ je konečná výstupná abeceda,
- Q je konečná množina stavov,
- $I \subseteq Q$ je množina počiatočných stavov,
- $F \subseteq Q$ je množina koncových stavov,
- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q$ je konečná množina prechodov,
- $\lambda : I \to \Delta^*$ je počiatočná výstupná funkcia, ktorá zobrazuje I na Δ^*, a
- $\rho : F \to \Delta^*$ je koncová výstupná funkcia, ktorá zobrazuje F na Δ^*.

Stavový prechod $e \in E$, označujeme $i[e]$ jeho vstupný symbol, $p[e]$ jeho východiskový alebo predchádzajúci stav a $n[e]$ jeho cieľový alebo nasledujúci stav, $w[e]$ jeho váhu (v prípade vážených automatov), $o[e]$ jeho výstupný symbol (v prípade prekladačov). Pre stav $q \in Q$, označujeme $E[q]$ množinu prechodov odchádzajúcich z q. V celej tejto práci budeme mať často na mysli konečné stavové prevodníky, keď budeme písať o stavových automatoch - čo je oveľa viac používaný a preto lepšie zrozumiteľný pojem. Prevodníky môžeme totiž stále chápať iba ako určitým spôsobom obohatené automaty. Obohatenie, ktoré využijeme bude spočívať nielen v pridaní akcie do stavového prechodu, ale aj vo vytváraní lokálnej pamäte, pod-automatov a nahradenie jednej akcie celou postupnosťou akcií. Práca sa nezameriava na formálne odvodzovanie a dokazovanie, kde by tieto rozdiely boli podstatné, ale využíva najmä základnú povahu štruktúry týchto formalizmov, ktorá je rovnaká.

Takýto stavový automat vhodne popisuje správanie programovaných robotov. Túto myšlienkou využil aj Rodney Brooks, ktorý sa považuje za otca Behaviour-Based robotiky, ktorej princípy sa používajú už vyše dvadsať rokov. Využil ho aj vo svojom systéme programovania s názvom Subsumption Architecture, kedy staviame správanie robota do vrstiev a až keď máme správne uchopené nižšie správanie, programujeme správanie na vyššej vrstve, ktoré interaguje s prvkami na nižších vrstvách. Pre príklad uviedem prípad, kedy chceme robota naučiť behať cez prekážky. V prvom kroku ho naučíme stáť (udržiavať rovnováhu), v ďalšom kráčať a skákať, kde môžeme interagovať so správaním na udržiavanie rovnováhy, ktoré sme naprogramovali skôr. Ďalej ho naučíme behať, kde opäť môžeme interagovať s kráčaním a napokon môžeme programovať beh cez prekážky, ktoré bude interagovať s viacerými správaniami.

Dôvody, prečo používať konečné stavové automaty pre programovanie robotov, dobre zhrnuli autori v publikácii [7], kde píšu, že stavové automaty sú jednoducho



Obr. 1.11: SubSumption Architecture z knihy [6]

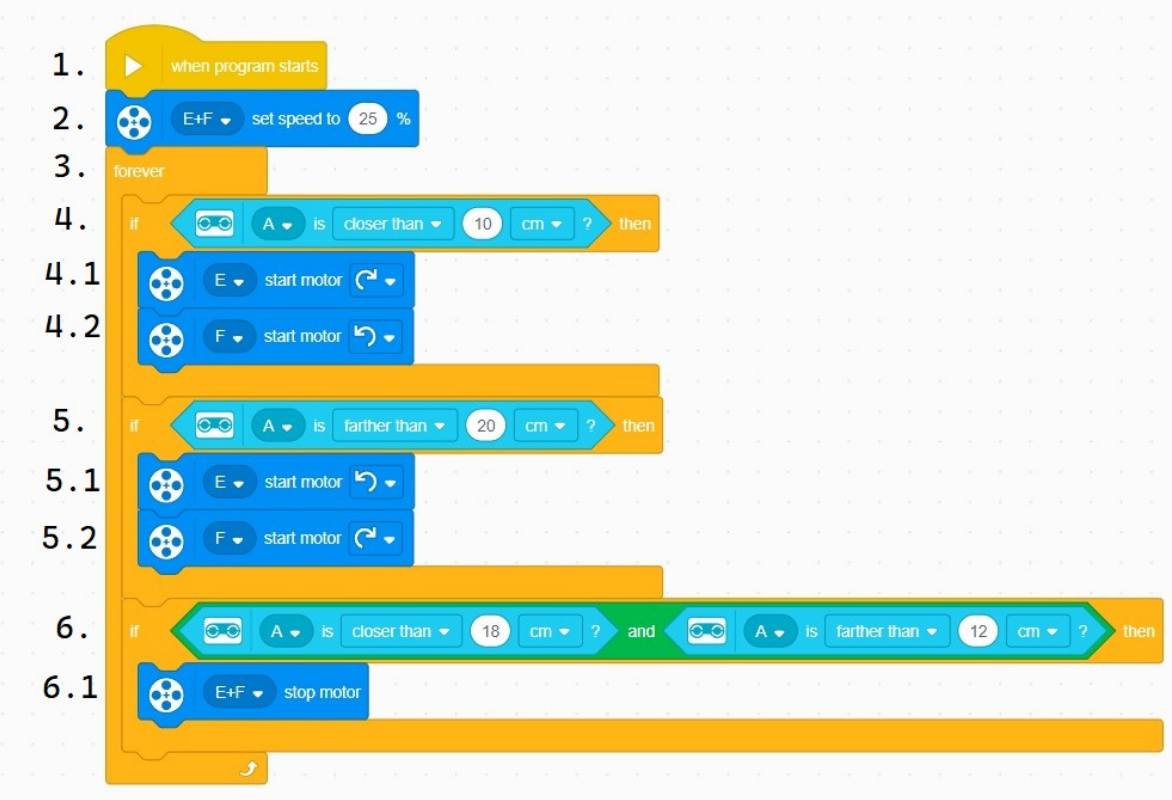
pochopiteľné, často používané medzi programátormi ako komunikácia a často už samé o sebe tvoria dokumentáciu. Je jednoduché sledovať, v ktorom stave sa daný robot nachádza a ľahko sa tak hľadajú chyby v programe.

1.3 LEGO Spike Prime

Ako sme si už popísali túto stavebnicu v časti 1.1.5, tohto robota je možné programovať pomocou rôznych jazykov. Hardvér riadiacej kocky pozostáva z procesora STM32F413 s 320KB pamäte RAM a 1MB pamäte flash, 32MB interného úložiska, šiestich portov pre vstupno-výstupné zariadenia, z čoho sú dva vysokorýchlostné a štyri obyčajné, maticovým LED displejom 5x5, konektorom pre micro-USB kábel, ktorý okrem programovania slúži aj na inštaláciu firmvérov a nabíjanie batérie (podstatné vylepšenie oproti EV3), Bluetooth Clasic a Bluetooth Low Energy, gyroskopom a akcelerometrom pre tri osi a zvukovým výstupom. Informácie prehľadne spísal Geoffrey Daniels na svojom GitHub repozitári [10]. Okrem vstavaného hardvéru stavebnica obsahuje motory na jednosmerný prúd so zabudovanými otáčkovými senzormi, ultrazvukový senzor na meranie vzdialenosti, senzor na meranie sily (a dotyku) a farebný/svetelný senzor.

Na ilustráciu práce s touto stavebnicou použijeme úlohu, kde robot má prenasledovať objekt, ktorý je pred ním. Keď sa objekt priblíži príliš blízko, tak robot začne cúvať a naopak, keď je objekt vzdialený, priblíži sa k nemu. Pre túto úlohu si zostrojíme robota s ultrazvukom, aký je vyobrazený na obrázku 1.13. Je to zjednodušená verzia príkladu z príručky [11], úloha 16. My ale použijeme iba jeden senzor. Obrázok 1.12 je fotografia funkčného programu a tu je jeho vysvetlenie:

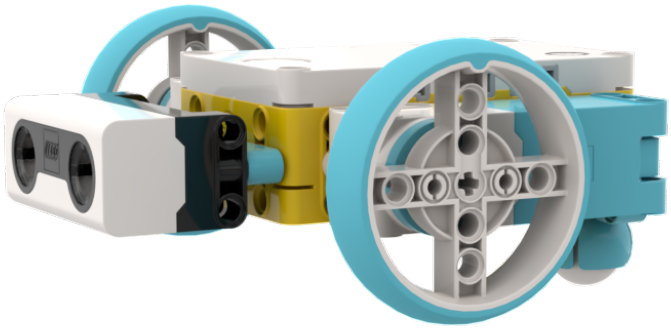
- 1. začiatok programu, ktorý sa spustí hneď po prijatí programu do riadiacej jednotky alebo po opätovnom stlačení ovládacieho tlačidla



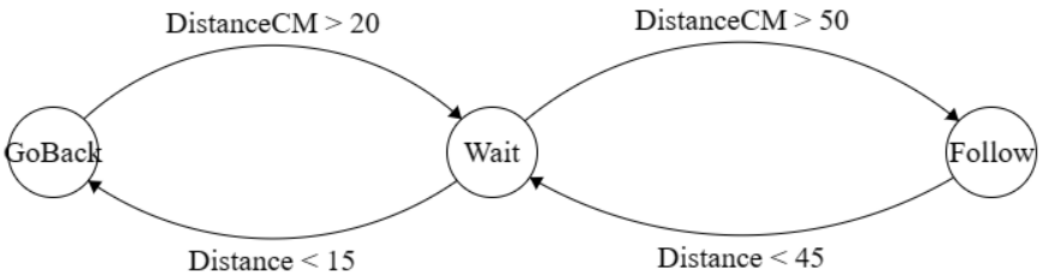
Obr. 1.12: Program, ktorého vykonávaním robot prenasleduje objekt, ktorý je pred ním.

- 2. nastavenie rýchlosti robota na 25% maximálnej rýchlosti
- 3. nekonečný cyklus programu
- 4. podmienka, v ktorej sa pýtame, či objekt pred nami je bližšie ako 10 centimetrov. Ak je táto podmienka splnená, tak:
  - 4.1 a 4.2 zapnutie motoru E v smere hodinových ručičiek a motoru F v opačnom smere
- 5. podmienka, v ktorej sa pýtame, či objekt pred nami je od nás ďalej ako 20 centimetrov. Ak je táto podmienka splnená, tak:
  - 5.1 a 5.2 zapnutie motorov E a F v opačnom smere ako v bodoch 4.1 a 4.2
- 6. podmienka, v ktorej sa pýtame, či objekt pred nami je ďalej ako 12 centimetrov a zároveň bližšie ako 18 centimetrov. Ak je táto podmienka splnená, tak:
  - 6.1 vypneme oba motory

Správanie robota s takýmto programom môžeme modelovať pomocou stavového automatu ako na obr. 1.14. Automat pozostáva z troch stavov: *Wait*, *GoBack* a *Follow*,



Obr. 1.13: Robot s ultrazvukovým senzorom. Obrázok bol vytvorený pomocou stud.io aplikácie.



Obr. 1.14: Robot pohybujúci sa závisle od vzdialenosti od objektu podľa stavového automatu vytvoreného na [12].

medzi ktorými prechádza závisle od toho, v akej vzdialenosti je od objektu. Ak si pod prechodmi predstavíme aj akcie - príkazy na ovládanie pohybu motorov, dostali by sme konečno-stavový prekladač, ktorý by mohol takýto program nahradiť.

1.3.1 Programátorské prostredia

Pre programovanie týchto stavebníc existuje softvér priamo od LEGO Education, ktorý je v online podobe aj ako aplikácia. Okrem tohto softvéru vieme používať aj softvér s názvom Pybricks [13], ktorý má tiež online aj offline verziu. Ponúka programovanie v jazyku Python a spoplatnenú verziu blokového jazyka v štýle jazyka Scratch, inštaláciu svojho firmvéru - odlišného od štandardného firmvéru LEGO a ukážky častí kódov, ktoré pracujú s knižnicou na obsluhu hardvéru v jazyku Python.

Pybricks a firmvér

Celý softvér Pybricks má verejný zdrojový kód na GitHub-e [14]. V repozitári *pybricks-code* sú zdrojové súbory aplikácie, ktorá je písaná v jazyku TypeScript nad framework-

om React. *pybricksdev* je základné nastavenie pre pokročilejších programátorov, ktorý chcú programovať v prostredí Visual Studio Code. Samotný firmvér je v repozitári *pybricks-micropython*. Tento repozitár má v stromovej štruktúre niekoľko pre nás zaujímavých častí. Prvá časť je priečinok *bricks*, ktorý obsahuje pravidlá, podľa ktorých sa vytvorí firmvér a v ktorých potom aj tento firmvér nájdeme (*/bricks/primehub/build/firmware.zip*). Druhou dôležitou časťou tohto repozitára je priečinok */lib/pbio/sys/*, v ktorom sú príkazy na ovládanie hardvéru. V *main.c* sa nachádza hlavná funkcia, v ktorej robot čaká na vypnutie alebo spustiteľný kód, ktorý následne vykoná. Ďalšou potrebnou časťou je súbor */bricks/_common/source.mk*, v ktorom sú vypísané všetky súbory, ktoré sa pri vytváraní firmvéru použijú.

Pybricks aplikácia je rozdelená na dve časti. V ľavej časti máme dve ikony, list a ozubené koleso, a navigáciu. Ozubené koleso symbolizuje nastavenia. Tu si vieme meniť farbu pozadia editora, inštalovať firmvér, nájdeme tu veľa užitočných odkazov, napríklad na úvod do tohto systému, vzorových projektov alebo miesto, kde vieme hlásiť akékoľvek problémy s týmto softvérom. Druhá možnosť, ktorú predstavuje prázdny list, slúži na správu našich programov. Jednotlivé programy vieme nahrávať, sťahovať a vytvárať nové. Pri vytváraní sa nás aplikácia opýta na jazyk, v ktorom budeme nášho robota programovať a názov programu. V pravej časti hore nájdeme štyri tlačidlá. Prvým sa pripájame na našu riadiacu jednotku pomocou Bluetooth, druhým spúšťame program, tretím ho vieme vypnúť a štvrtým spustíme konzolu, v ktorej vieme posielať robotovi príkazy postupne.

Pri tvorbe firmvéru je potrebný prekladač pre 32-bitový ARM (cross-compiller), keďže jeho kód je napísaný v jazyku C. Pre vytvorenie firmvéru pre stavebnicu LEGO Spike Prime potrebujeme byť v najvyššom priečinku repozitára a v operačnom systéme Linux zavolať príkaz *make primehub*. Na úspešnú inštaláciu potrebujeme mať aktualizované ovládače, na čo nás upozorní aj aplikácia pri neúspešnej inštalácii. Po úspešnom vytvorení firmvéru v aplikácii urobíme nasledujúci zoznam úkonov v danom poradí:

1. v ľavej časti hlavného menu otvoríme ozubené koleso, čo predstavuje nastavenia
2. v navigácii v časti *Firmware* zvolíme možnosť *Install Pybricks Firmware*
3. vyskakovacie okno v dolnej časti v strede má možnosť *Advanced*, klikneme na ňu
4. do zobrazeného rámčeka presunieme náš vytvorený firmvér alebo ak klikneme doň, tak nám to otvorí prieskumníka, v ktorom si náš firmvér vyhľadáme
5. po vložení firmvéru by ho aplikácia mala rozoznať, čo sa prejaví tým, že nám v hornej časti už neukazuje možnosti jednotlivých stavebníc
6. potvrdíme tlačidlom *Next*

7. odsúhlasíme licenciu v zaškrtávacom boxe a klikneme na *Next*
8. pomenujeme si našu riadiacu jednotku, ktorej názov sa nám bude zobrazovať pri pripájaní na ňu a potvrdíme tlačidlom *Next*
9. postupujeme podľa návodu:
   (a) odpojíme riadiacu jednotku a vypneme ju
   (b) za stáleho držania tlačidla Bluetooth pripojíme kocku dátovým káblom k počítaču
   (c) tlačidlo Bluetooth pustíme potom, čo začne blikať na rúžovo-zeleno-modrú farbu
10. klikneme na tlačidlo *Install*
11. aplikácia nám ponúkne nové vyskakovacie okno, v ktorom zvolíme našu riadiacu jednotku a potvrdíme tlačidlom *Connect*
12. úspešná inštalácia je signalizovaná rozsvietením LED matice na riadiacej jednotke alebo zmiznutím ukazovateľa priebehu inštalácie v aplikácii.

1.4 Programovanie grafickej aplikácie v C#

Programovanie aplikácií v jazyku C# je jednoduché a efektívne vďaka používaniu softvérov na to určených. Jazyk je objektovo orientovaný a o správu pamäte sa nemusíme starať, keďže ju spravuje mechanizmus *garbage collector*. Vývojové prostredia ponúkajú množstvo rôznych predprogramovaných komponentov, užitočných a efektívnych knižníc a okolo týchto aplikácií je široká komunita a podrobná dokumentácia. Aplikácie väčšinou využívajú framework .NET. Pri vytváraní novej aplikácie vytvoríme nový projekt, zvolíme možnosť *Windows Forms App* s *.NET Framework* v záložke *Deskop*. Po potvrdení tlačidlom *Create* nám softvér vytvorí prázdnu aplikáciu, ktorá po spustení otvorí našu novovytvorenú aplikáciu. Spustiť ju vieme z editora a poslednú spustenú z editora vieme spustiť v priečinku, kde je náš projekt */nazovProjektu/nazovProjektu-/bin/Debug/nazovProjektu.exe*. Automaticky vygenerovaný kód novej aplikácie môže vyzerať ako v ukážke 1.1. Príkaz *Using* znamená to isté, ako v iných jazykoch *Include*. *namespace* je totožný s príkazom, ktorý sa používa v jazyku C++, čiže namiesto volania *WindowsFormsApp1.Form1()* nám postačí *Form1()*. Funkcia *public Form1()...* je konštruktor našej triedy, ktorý inicializuje všetky grafické komponenty. Neodporúča sa do tejto funkcie čokoľvek písať, lebo ak to interaguje s nejakým grafickým komponentom, ktorý ešte nebol inicializovaný, tak to môže spôsobiť spadnutie našej aplikácie. Ak máme nejaký kód, ktorý chceme spustiť hneď po otvorení aplikácie, je potrebné na

Algoritmus 1.1: Automatický generovaný kód po vytvorení novej aplikácie C#

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Algoritmus 1.2: Automatický generovaný kód pre grafické komponenty v C#

```
private void button1_Click(object sender, EventArgs e)
{
    throw new System.NotImplementedException();
}
```

to použiť funkciu *public Form1_Load()...*, ktorá slúži presne pre tieto účely. Vytvoriť ju je potrebné cez záložku *Design* v spodnej časti softvéru.

V softvéri vieme všetko grafické ručne programovať, alebo si v záložke zvolíme namiesto *Code* záložku *Design*, čo nám otvorí celú grafickú paletu nástrojov. Komponenty vieme pridávať do našej aplikácie, odoberať, škálovať, meniť rôzne atribúty a po dvojkliku na tieto komponenty, ak sú interaktívne, sa nám vygeneruje kód 1.2, v ktorom vieme programovať správanie. Ak chceme takýto kód neskôr odstrániť, je nutné to robiť opäť cez *Design*, alebo túto funkciu nájsť v súbore *Form1.Designer.cs* a vymazať ju i odtiaľto.

1.4.1 Serializácia v aplikáciách C#

Serializácia je ukladanie objektov v pamäti do formátu, ktorý sa dá prenášať medzi počítačmi, alebo uložiť do vonkajšej pamäte počítača a neskôr pri čítaní opäť vytvoriť pôvodne objekty. Na serializáciu v C# možno využiť buď štandardné .NET API (napr. XmlSerializer/BinaryFormatter), alebo je možné použiť iné knižnice, ktoré sa dajú veľmi ľahko a rýchlo nainštalovať. Binárna serializácia je nebezpečná a spôsobuje

problémy pri aktualizácii softvéru na novú verziu. Vhodná knižnica pre naše potreby je *Newtonsoft.Json*, ktorú nainštalujeme v časti *Tools > NuGet Package Manager > Manage NuGet Packages for Solution*, kde túto knižnicu vyhľadáme a nainštalujeme. Potom stačí pridať *string jsonString = JsonConvert.SerializeObject(Object, settings);*, kde *jsonString* je textový reťazec vo formáte JSON. Ten následne môžeme zapísať do súboru. Knižnica serializuje všetko, čo je v triede *Object* prístupne metódami *get; set;* a pri tom používa nastavenia, ktoré sú v argumente *settings*. Vieme tam nastaviť počet medzier, ignorovanie rekurzívnych závislosti a mnoho ďalších parametrov. Pri ignorovaní závislosti si treba dať pozor, lebo knižnica si neurobí nejakú tabuľku podľa ktorej tieto závislosti pri deserializácii naspäť nastaví, ale všetky tieto parametre serializuje ako null.

Kapitola 2

Špecifikácia a ciele práce

V tejto kapitole nájdeme cieľ bakalárskej práce, špecifikáciu funkcionality a dôvody prečo sme zvolili práve takéto riešenie.

Tak ako sme uviedli v časti 1.2.1, stavové automaty, presnejšie konečné stavové prekladače sú vhodný, prehľadný a dokumentujúci spôsob ako správne popísať správanie robota v rozličnej situácii. Cieľom práce bude umožniť užívateľom programovať robotov takýmto spôsobom. Úloh bude v tejto práci niekoľko a to nasledovných:

- tvorba aplikácie pre vytváranie konečných stavových prevodníkov
- písanie firmvéru na čítanie a vykonávanie takýchto automatov
- úprava aktuálnej webovej aplikácie od Pybricks pre posielanie nami vytvorených automatov

.

2.1 Softvér na tvorbu automatov

Takýto softvér by mal užívateľovi dovoľovať tvorbu a editáciu paralelných a vnorených automatov, vytváranie stavov, ich editovanie a mazanie, medzi stavmi tvoriť prechody, ktoré budú obsahovať podmienky, za ktorých sa stav robota zmení a zoznam akcií, ktoré pri tejto zmene má robot vykonať a vytváranie dátových štruktúr pre lokálnu pamäť na ovládanie v rámci jedného automatu a globálnu pamäť, ktorá bude slúžiť na komunikáciu medzi automatmi. V softvéri by mala byť možnosť uloženia, načítania a tvorby exportu nami vytvoreného automatu.

2.2 Nový firmvér

Robot bude vedieť efektívne vykonávať naprogramované konečné stavové prevodníky. Prevodníky bude možné do robota prijať v predspracovanej podobe, ktorá sa vyexpor-

tuje z aplikácie na tvorbu automatov.

2.3 Úprava webovej aplikácie

Webová aplikácia Pybricks slúži ako komunikačný kanál medzi riadiacou jednotkou a počítačom. Upravená verzia bude umožňovať posielať robotovi exportované vytvorené stavové automaty za účelom ich následného vykonávania modifikovaným firmware. Zároveň bude umožňovať vypisovať správy z bežiaceho automatu do svojej konzoly.

Kapitola 3

Návrh

V nasledujúcej kapitole sa budeme zaoberať návrhom modelu jazyka, návrhom aplikácie pre tvorbu stavových automatov, modifikáciou webového rozhrania a rozšírením súčasného firmvéru pre vykonávanie stavových automatov. Niektoré z týchto návrhov vizualizujeme obrázkom pre lepšie vysvetlenie danej funkcionality alebo na zobrazenie vzťahu medzi jednotlivými časťami.

3.1 Návrh modelu stavových automatov

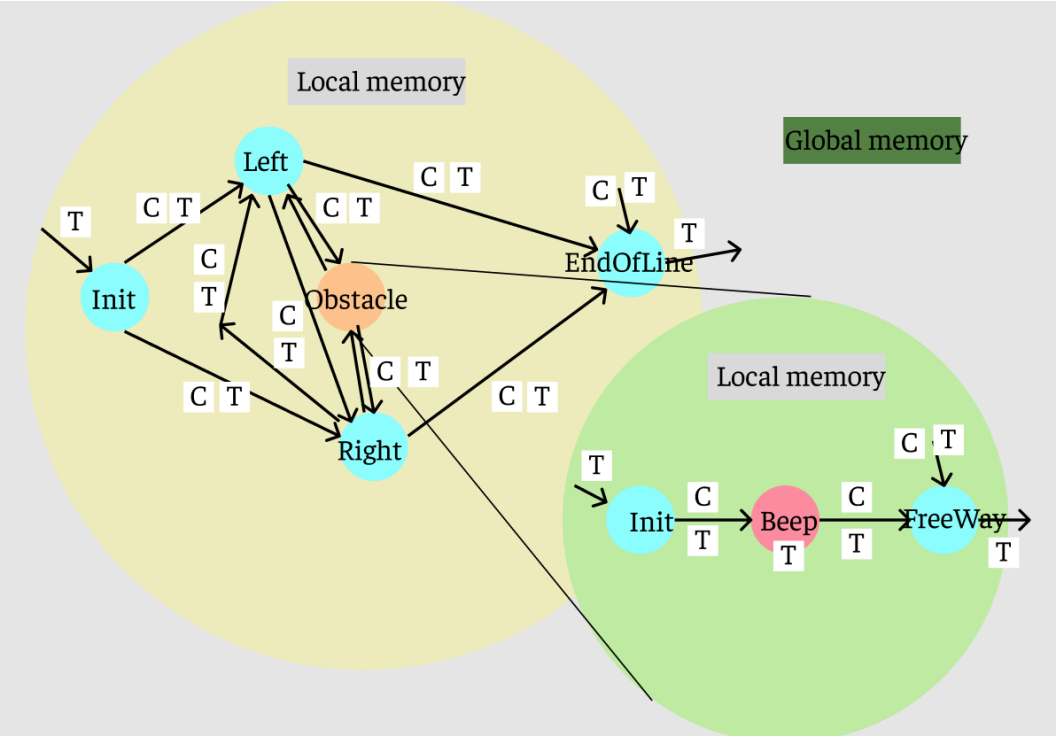
Stavový automat, ktorý bude môcť užívateľ vytvoriť, bude o čosi zložitejší od konečnostavových prevodníkov, ktoré sme spomínali v časti 1.2.1, ale o to viac môžu obohatiť stavovo definované správanie robotov. Ukážku návrhu je možné vidieť na obrázku 3.1. Samotné správanie bude možné rozdeliť do automatov (paralelných automatov), ktoré sa po spustení začnú vykonávať súbežne. Každý automat bude obsahovať práve jeden inicializačný stav, neprázdnu množinu konečných stavov a ďalšiu množinu stavov rôzneho typu. Tieto typy sú:

1. prázdny stav, ktorý je základný a po tom, čo sem program vstúpi, tak sa nič navyše nedeje

2. aktívny stav, ktorý bude obsahovať zoznam príkazov, ktoré sa budú v určitom časovom intervale dookola vykonávať pokiaľ program tento stav neopustí

3. stav s vnoreným automatom, do ktorého keď program príde, tak sa spustí vykonávanie vnoreného automatu, ktoré sa ukončí vtedy, keď vnorený automat vojde do jedného z konečných stavov. Tu bude mať užívateľ možnosť nastaviť takzvaný *Exit code*, ktorý bude môcť použiť v podmienkach prechodových funkcií.

Medzi jednotlivými stavmi bude možné vytvárať prechody, ktoré budú obsahovať podmienky, za ktorých prechod z jedného stavu do druhého nastane a zoznam



Obr. 3.1: Návrh grafického zobrazenia automatu za použitia nástroja Figma [15].

akcií, ktoré sa pri danej zmene majú vykonať. Žiaden prechod nemôže smerovať do inicializačného stavu a žiaden prechod nemôže smerovať z konečných stavov do iného stavu. Medzi každými dvoma stavmi môže existovať nanajvýš jeden stavový prechod. Je možné zadať inicializačný skript (funkciu), ktorá sa vykoná predtým, ako automat prejde do svojho počiatočného stavu, čiže zoznam inštrukcií, ktoré sa majú vykonať pred spustením daného automatu. Konečné stavy môžu mať dva výnimočné prechody:

1. Super prechod, ktorého podmienka sa kontroluje neustále počas vykonávania aktuálneho automatu (čiže ak je splnená hneď po vstupe do tohto automatu, tak vykonávanie sa hneď ukončí, keďže program prejde do konečného stavu) a obsahovať môže zoznam inštrukcií na vykonanie

2. a konečný prechod, ktorý obsahuje zoznam inštrukcií a zároveň slúži na záverečné uloženie dát do globálnej pamäte alebo nastavenie *Exit code* pred odchodom z daného automatu.

Automaty budú obsahovať lokálne a globálne premenné, ktoré budú čísla, znakové reťazce a boolovské hodnoty. Lokálna premenná bude dostupná pre daný automat, ktorému bola priradená, zatiaľ čo globálna bude viditeľná všetkým paralelným aj vnoreným automatom. Medzi premenné patria i inštancie alebo odkazy na nainicializované zariadenia (motory a senzory).

Algoritmus 3.1: Ukážka modelu jazyka

```
define(x, int, 2)
define(y, int, 5)
define(z, int 0)
sum(z, x, y)
print_value("z=", z)
```

3.2 Návrh modelu jazyka pre akcie v automatoch

Syntax príkazov, ktoré budú reprezentovať akcie vykonávané na stavových prechodoch, alebo skripty vnútri aktívneho stavu vychádza zo syntaxe jazyka Python, čo bude zjednodušením pre veľkú časť užívateľov. Operáciu sa zapisujú pomocou názvu funkcie a vstupných argumentov v zátvorkách oddelených čiarkami, pozri ukážku 3.1. Prvé tri riadky deklarujú a definujú nové premenné x, y a z, ktoré sú typu *Integer* s rôznou iniciálnou hodnotou. Následne skript vykoná súčet, ktorého výsledok sa uloží do premennej z a sčítance sú x a y. V poslednom kroku programu sa volá funkcia pre výpis hodnoty do konzoly, ktorá má dva argumenty. Prvým je textový reťazec a druhým je samotná hodnota, ktorú chceme vypísať. Takýmto spôsobom bude užívateľ vedieť volať funkcie, ktoré sa starajú o obsluhu motorov, senzorov a ďalších zariadení. Jazyk umožňuje:

1. definovať premenné typu *Integer*, *String* a *Boolean*, operácie s nimi ako napríklad priradenie, sčítanie, odčítanie, nové definovanie

2. jednoduché logické operácie ako sú logické *And*, logické *Or*, logické *Not*

3. ovládanie a volanie funkcií nad hardvérom

4. jednoduché cykly *For* a *While*

5. komunikáciu s počítačom pomocou Bluetooth pre výpisy do konzoly.

.

3.2.1 Vzorové programy

Na demonštrovanie vyjadrovacej sily a jednoduchosti takého jazyka využijeme nasledujúce ukážky. Každú ukážku si prejdeme krok po kroku a vysvetlíme si, čo daný program robí, na čo slúži.

Sledovanie objektu K tejto časti patrí program v ukážke 3.2. Na začiatku je definícia a deklarácia piatich premenných:

1. *distance* bude slúžiť na uchovávanie si vzdialenosti, ktorú budeme získavať z ultrazvukového senzora

2. *condition* bude slúžiť na uchovávanie si logického výsledku porovnania

3. *follow* je premenná, ktorú využijeme na opakovanie cyklu *While*

4. *sensor* je premenná alebo odkaz na inštanciu senzora, ktorý bude pripojený na porte A

5. *base*, je tiež premenná alebo odkaz na inštanciu dvoch synchronizovaných motorov, kde ľavý motor je na porte B a pravý motor je na porte C

Na ďalšom riadku začína cyklus *While*, ktorý je nekonečný, keďže premenná *follow* sa nikde nemení. Vo vnútri cyklu je funkcia, ktorá do premennej *distance* uloží hodnotu vzdialenosti zo senzora *sensor*. Následne sú tri porovnávania, kde sa výsledok uloží do premennej *condition*. Porovnáva sa hodnota premennej *distance* s konštantami 400, 100, 150 a 350. Ak sa splní jedna z prvých dvoch podmienok, tak sa zavolá funkcia, ktorá ovláda motory a pohyb spustí s rýchlosťou 200 milimetrov za sekúndu a uhlom 0 stupňov za sekundu. Ak je splnená tretia podmienka, tak sa motory zastavia.

Gyroskop a LED matica Táto časť opisuje program v ukážke 3.3. Na začiatku je definícia a deklarácia šiestich premenných:

1. x a y použijeme na uchovanie si informácie o polohe bodu v ypsilonovej a xovej osi

2. $x_rotation$ a $y_rotation$ budeme využívať na uchovanie si informácie o naklonení riadiacej jednotky v daných osiach

3. $condition_x$ a $condition_y$ využijeme pre uchovanie si logických kontrol, či sa riadiaca jednotka dostatočne veľa naklonila v danej osi

V ďalšom riadku je nekonečný cyklus *While*. Vo vnútri cyklu sa na začiatku získava veľkosť náklonu v osiach x a y, ktoré sa priradia do premenných. Potom sa kontrolujú všetky možnosti, ktoré môžu nastať. Kocka je naklonená hore, dole, vpravo, vľavo alebo nie je naklonená, a podľa toho sa rozsvieti bod v maticovej mriežke so stopercentnou silou podsvietenia.

Algoritmus 3.2: Program pre sledovanie objektu

```
define(distance, int, 0)
define(condition, bool, false)
define(follow, bool, true)
define(sensor, Sensor, A)
define(base, DriveBase, B+C)
while(follow)
{
    get_low_distance(distance, sensor)
    more(condition, distance, 400)
    if(condition)
    {
        base_run_forever(base, 200, 0)
    }
    less(condition, distance, 100)
    if(condition)
    {
        base_run_forever(base, -200, 0)
    }
    between(condition, distance, 150, 350)
    if(condition)
    {
        base_stop(base)
    }
}
```

Algoritmus 3.3: Program pre prácu s gyroskopom a LED svetelným displejom

```
define(x, int, 2)
define(y, int, 2)
define(x_rotation, int, 0)
define(y_rotation, int, 0)
define(condition_x, bool, false)
define(condition_y, bool, false)
while(true)
{
    matrix_clear()
    get_x_rotation(x_rotation)
    get_y_rotation(y_rotation)
    more(condition_x, x_rotation, 40)
    between(condition_y, y_rotation, -30, 30)
    if(and(condition_x, condition_y))
    {
        redefine(x, 0)
        matrix_set_pixel(x, y, 100)
    }
    less(condition_x, x_rotation, -40)
    if(and(condition_x, condition_y))
    {
        redefine(x, 5)
        matrix_set_pixel(x, y, 100)
    }
    between(condition_x, x_rotation, -30, 30)
    if(and(condition_x, condition_y))
    {
        redefine(x, 2)
        redefine(y, 2)
        matrix_set_pixel(x, y, 100)
    }
    ...
}
```

3.3 Návrh aplikácie

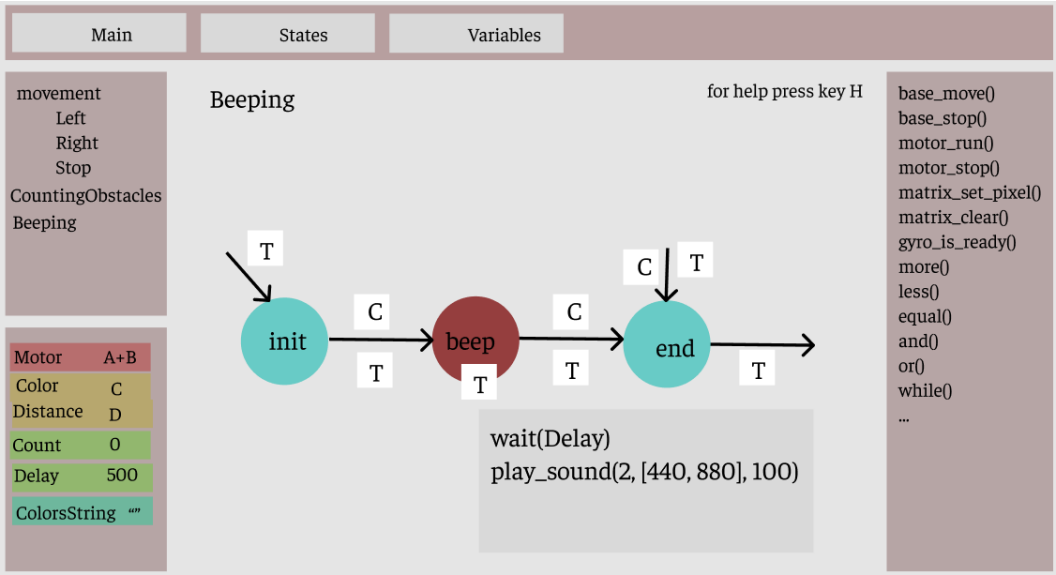
Ako sme uvideli v podkapitole 1.4, jazyk C# je používaný na rýchlu tvorbu grafických aplikácií pre platformu Windows pre takzvané Windows Forms Applications. Aplikácia bude musieť byť prehľadná a ľahko ovládateľná, na čo nám poslúžia rôzne komponenty, akými sú napríklad tlačidlá, zoznamy, oznamovacie okná a ďalšie predprogramované funkcionality. Ukážku návrhu používateľského rozhrania je možné vidieť na obrázku 3.2.

V hornej časti obrazovky sa bude nachádzať lišta, ktorá bude obsahovať niekoľko sekcií podľa funkcie, ktoré budú ponúkať. V prvej sekcii budú tlačidlá pre tvorbu nových paralelných automatov, možnosť uložiť alebo exportovať aktuálny stavový automat, načítať automat, ktorý užívateľ niekedy v minulosti uložil a možnosť zavrieť aplikáciu. Čítanie a ukladanie bude zabezpečené serializáciou. Export bude špeciálny formát, kedy všetky názvy funkcií budú prekonvertované na čísla. V druhej sekcii budú tlačidlá zamerané na tvorbu stavov, ktoré pre množstvo používania bude možné zavolať i klávesovými skratkami. Posledná časť sa bude venovať správe, presnejšie tvorbe premenných.

Ľavá časť obrazovky bude vyhradená pre sumarizáciu aktuálneho správania robota, ktoré predstavujú všetky paralelné a vnorené automaty a všetky premenné, s ktorými bude robot vedieť pracovať, rozdelené to bude horizontálne, kde v hornej časti bude takzvaný *tree view*, kde užívateľ bude prehľadne vidieť, ktorý vnorený automat patrí ktorému automatu, jednotlivé automaty bude vedieť zbaliť a rozbaliť, po kliknutí naň bude mať možnosť editovať ho alebo vymazať. V spodnej časti bude obyčajný zoznam, ktorý bude obsahovať všetky premenné, ktoré užívateľ vytvoril. Pre prehľadnosť, jednotlivé typy premenných budú farebne rozlíšené. Tak ako pri kliknutí na nejaký automat má užívateľ možnosť ho editovať, túto možnosť bude mať aj tu. Po kliknutí na meno premennej bude mať možnosť meniť meno, hodnotu alebo premennú odstrániť.

V pravej časti sa bude nachádzať textové pole obsahujúce všetky možné funkcie, ktoré môže užívateľ použiť. Keďže sú vypísané v textovom poli, užívateľ ich bude môcť jednoducho kopírovať do stavových automatov.

V strednej časti obrazovky, ktorej bude vyhradené najviac priestoru, sa budú stavové automaty vykresľovať. Pre jednoduchosť a prehľadnosť sa bude vykresľovať vždy iba jeden automat a to ten, ktorý bude zvolený v ľavej časti obrazovky a jeho meno bude vypísané v hornej časti. Keďže automat môže obsahovať viac druhov stavov, tieto stavy sa budú vykresľovať rôznou farbou podľa typu akého sú. Medzi jednotlivými stavmi budú prechodové funkcie alebo prechody, ktoré bude možné kresliť oblúkom alebo lomenou čiarou, aby to bolo pre užívateľa prehľadné, a smer prechodu bude označený šípkou. Keďže automat obsahuje výnimočné prechody, tak tie budú zvýraznené odlišnou farbou. Interakciu so stavmi a prechodmi bude uskutočňovaná pomocou myši a



Obr. 3.2: Návrh grafickej aplikácie za použitia nástroja [15]

kláves. Všetky klávesové skratky budú vypísané na pracovnej ploche, alebo bude vypísaná klávesová skratka, po stlačení ktorej sa všetky skratky a celé ovládanie zobrazí užívateľovi. K podmienkam prechodov, zoznamom inštrukcií ktoré sa majú pri zmene stavu vykonať a ďalším obdobným funkciám sa používateľ bude vedieť dostať pomocou tlačidiel, ktoré budú na takom mieste, aby bolo užívateľovi jasné, k čomu patria.

3.4 Návrh firmvéru

Nebudeme tvoriť nový firmvér ale rozširovať existujúci od spoločnosti Pybricks, ktorý sme už spomínali v časti 1.3.1. Firmware je určený na prijímanie a štartovanie programov v jazyku Python, ktoré sú preložené do bajtkódu dialektu MicroPython. Celú túto časť nevyužijeme. Firmvér však obsahuje množstvo užitočného kódu na riadenie hardvérových komponentov a to v rôznych módoch a tú časť využijeme. Napríklad pre farebný senzor vieme získať priamo farbu, alebo RGB, alebo HSV a mnoho ďalších typov výstupov. V prvom kroku zjednodušíme volanie týchto funkcií a to tým, že naprogramujeme API (podľa návrhového vzoru Adaptér), kedy užívateľ napríklad zavolá použije akciu *get_low_distance()*, čo na pozadí zavolá funkciu *get_data()* s parametrom módu, ktorý bežný užívateľ nepotrebuje vedieť. Takto vieme používateľovi sprístupniť množstvo užitočnej funkcionality dostupnej priamo vo firmvéri.

Druhým krokom bude vytvorenie globálnej pamäte pre presun parametrov medzi funkciami. Napríklad vyššie spomenutá funkcia potrebuje dva parametre. Prvým je senzor, z ktorého bude čítať dáta a druhým je smerník do pamäte, kam sa majú dáta uložiť. Vďaka tomu sa zo všetkých funkcií stanú funkcie bez parametra a bude možné smerníky na tieto funkcie zaradiť do jednej tabuľky a efektívne ich podľa čísla fun-

kcie/akcie pri interpretovaní automatu zavolať v čase O(1) potom, čo sa jej parametre pripravia do globálnej štruktúry parametrov.

Tretím krokom bude tieto funkcie uložiť do poľa, pre jednoduchšiu interpretáciu a čítanie stavových automatov. Napríklad vyššie spomenutá funkcia bude mať v poli index 54. Čiže pri tvorbe exportu automatu sa nepoužije názov funkcie ale číslo 54.

Štvrtý krok predstavuje príprava štruktúr, ktoré budú slúžiť na uschovanie dát ohľadom stavových automatov, Využijeme dynamické alokovanie pamäte a knižnice na správu dynamických štruktúr. To však nie je také jednoduché, keďže pôvodný firmware je navrhnutý tak, aby dynamická pamäť bola dostupná predovšetkým micropythonovským programom. Celkovo je bežné (a je to aj tento prípad), že program bežiaci na embedded ARM architektúre nemá dostupnú dynamickú pamäť pomocou tradičných volaní malloc() / free(). Systém bol vytvorený využitím knižníc ContikiOS, ktorý umožňuje 3 spôsoby alokovania pamäte: 1) bežná dynamická pamäť pomocou rozhrania MEMB - informácie o pamäťových blokoch nie sú uložené priebežne na začiatku blokov ako to robí malloc(), ale v inej časti pamäte, 2) špeciálna manažovaná pamäť podporujúca defragmentáciu a 3) malloc(), ktorý ale nie je odporúčaný a v tomto firmvér jeho použitie ani nie je možné. Využijeme teda alokovanie v štýle MEMB, čo si vyžiada určitú agendu naviac.

V ďalšom kroku bude vytvorenie metódy, ktorá bude čítať vstup odoslaný cez konzolu webového rozhrania a z neho vytvárať vnútornú reprezentáciu dátových štruktúr tak, ako to používateľ naprogramoval v našej aplikácii, ktorá bude potom pripravená na interpretovanie automatu.

No a v poslednom kroku sa po odštartovaní programu z webového prostredia automat odštartuje, t.j. bude interpretovať prijatý kód, spúšťať požadované zoznamy akcií, kontrolovať podmienky stavových prechodov, udržiavať lokálne aj globálne premenné použité v automate a realizovať stavové prechody, keď budú podmienky splnené.

Všetky tieto kroky bude potrebné neustále kontrolovať a testovať aby sme neskôr neobjavili nežiadúce správanie, ktoré bude ťažké odstrániť. Firmvér má predprogramované testy, ale nie sú k ním návody, ako ich spustiť a ako pripraviť robota na takéto testovanie.

3.5 Návrh úprav webového prostredia

Webová aplikácia od spoločnosti Pybricks prejde taktiež zmenami. Momentálne je možné vytvárať dva druhy programov, ktorými sú blokový program a Python. My budeme implementovať možnosť tretieho typu a tým je stavový automat. Automat exportovaný do špeciálneho interného formátu (v textovej reprezentácii) sa bude do robota posielať tým istým tlačidlom ako sa posielali programy doteraz cez sériovú

linku BlueTooth spojenia s robotom.

Kapitola 4

Implementácia

Táto kapitola sa zameriava na priebeh implementácie niektorých častí návrhu a to tých, ktoré boli problémové alebo zaujímavé.

4.1 Čítanie dát zo senzorov

V celom firmvéri je jediná funkcia, ktorá dostáva dáta zo senzorov a takýto je jej pseudokód: *error get_ data(legodev_ dev*, uint8_ t, void**)*. Prvým parametrom je smerník na zariadenie, z ktorého chceme čítať, druhým je mód, v ktorom ich bude zariadenie spracovávať a tretím je smerníková referencia na premennú v pamäti, ktorá ukazuje na štruktúru, do ktorej sa uložia dáta. Je pri tom použitý typ *void ** a to preto, že každý mód vracia iný typ dát. Tento typ dát nájdeme pri definícii módov hneď vedľa. Líšia sa počtom bitov alebo aj počtom parametrov, preto treba pri každom čítaní pracovať s dátami osobitne. Preto sme rozšírili náš interface, kde ponúkame užívateľovi čítanie týchto dát v rôznych módoch bez potreby údržby a spracovania výsledkov.

4.2 Ovládanie motorov

Pre pohyb je vo firmvéri množstvo funkcií a väčšina z nich obsahuje mód, čo má robiť potom, čo pohyb ukončí. Firmvér nám nedovolí zavolať dva po sebe idúce funkcie, ktoré ovládajú pohyb motora, pokiaľ sa prvý pohyb neukončil. Pre predstavu si môžeme uviesť spustenie motora, ktorý otočí servomotorom daný počet stupňov danou rýchlosťou. Pseudokód je takýto: *error servo_ run_ angle(servo*, int, int, control_ on_ completion)*. Posledným parametrom určujeme mód, ktorý definuje to, čo má robiť motor po tom, čo splní úlohu.

Jednou z možností je, že motor bude pokračovať v pohybe, ale uvoľníme ho, aby mohol prijať a vykonať ďalšiu inštrukciu. Inou možnosťou je, že motor zastaví a bude držať aktuálny uhol bez toho, aby mohol vykonať novú inštrukciu. Pri programovaní

by bolo pre užívateľa náročné sa zorientovať vo všetkých týchto módoch a tak všetky pohyby budú mať jeden a ten istý mód. Po vykonaní inštrukcie sa motor zastaví a dovolí vykonávanie ďalších inštrukcii.

4.3 Bluetooth a komunikácia

Po nainštalovaní firmvéru cez kábel je jediná možnosť ako s robotom komunikovať Bluetooth. Cez Bluetooth vieme nahrať celý program alebo cez konzolu písať príkaz po príkaze. Pri testovaní ako robot prijíma programy sme zistili, že webová stránka má zabudovanú knižnicu, ktorá z programu v Pythone vyrobí bajtový kód, ktorý následne posiela pomocou Bluetooth-u.

Riešením je posielať nami vytvorené stavové automaty exportované aplikáciou pomocou konzoly webového prostredia, pretože údaje odoslané cez túto konzolu sa posielajú kompletne a bez žiadnych úprav. Na strane robota je funkcia, ku ktorej máme jednoduchý prístup, ktorá číta z automaticky napĺňaného bufra BlueTooth komunikačnej linky. Jej pseudokód je takýto: *error bluetooth_rx(uint8_t *, uint32_t*)*. Vstupné parametre sú dva. Prvým je smerník na pole, ktoré bude obsahovať vstupný reťazec. Druhým je smerník na číslo, ktoré bude popisovať množstvo prijatých znakov. Bufer nie je nekonečný ako ani pamäť robota a tak keď je prenášané veľké množstvo dát, ktoré chceme spracovávať v robotovi, tak je na mieste použiť cyklus *while*, ktorý sa bude opakovať pokiaľ hodnota, kam ukazuje smerník na počet dát nebude nulová.

Takýto istý problém má aj vysielanie dát z robota. Zásobník, ktorý plníme, má veľkosť približne 25 znakov, čiže ak chceme vysielať nejaký dlhší reťazec, tak je to potrebné taktiež urobiť pomocou *while* cyklu. Funkcia, ktorá zapisuje do bufra, má dva parametre, kde prvým je pointer na začiatok reťazca, ktorý chceme odvysielať a druhým je dĺžka tohto reťazca. Výstupom okrem kódu chyby je druhý parameter, ktorý po návrate z funkcie obsahuje množstvo dát, ktoré zapísal do výstupného bufra.

Kapitola 5

Výsledky

Táto kapitola obsahuje dve časti. Prvou časťou sú ukážky programov vytvorených v systéme navrhnutom v bakalárskej práci a v druhej časti sú spôsoby, ako sme testovali funkčnosť a stabilitu programov.

5.1 Ukážky hotových programov

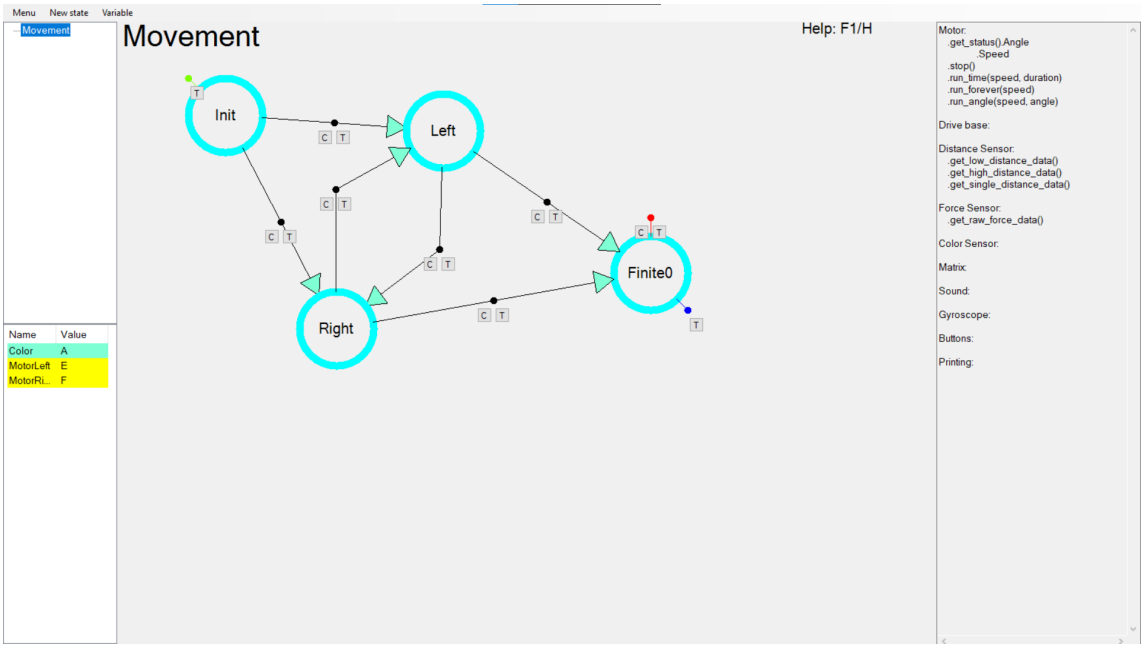
Tu sa nachádzajú výsledky implementácií, ktoré sme si popísali v kapitole 4. Každú časť popisujeme samostatne s tým, že hodnotíme, čo sa nám podarilo, na aké problémy sme pri tom narazili a ako sme ich riešili.

5.1.1 Aplikácia na tvorbu automatov

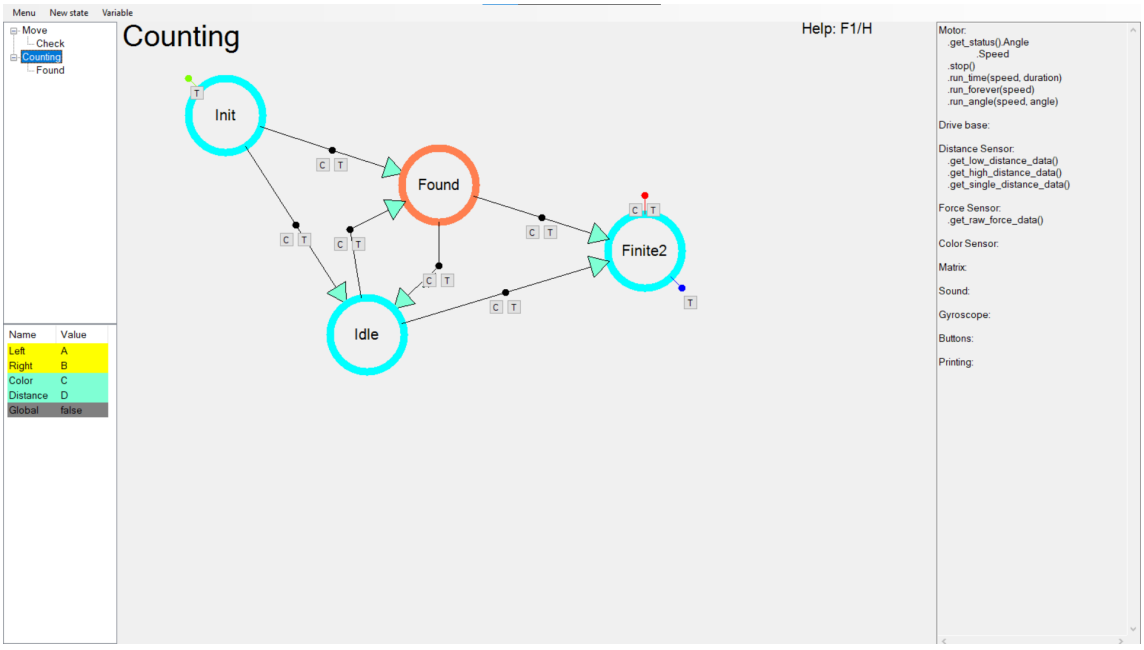
Táto aplikácia spĺňa všetky základne požiadavky, ktoré nám vznikajú pri tvorbe stavových automatov. Vieme vytvárať paralelne spúšťajúce sa automaty, vnorené automaty a množstvo stavov. Okrajové prípady sú ošetrené a chyby, ktoré sa objavili počas testovania boli analyzované a taktiež odstránené.

Na obrázku 5.1 môžeme vidieť jednoduchý prototyp správania robota, ktorý sleduje čiaru. Pri pohybe využíva dva motory a farebný senzor, ktoré môžeme vidieť v ľavej dolnej časti.

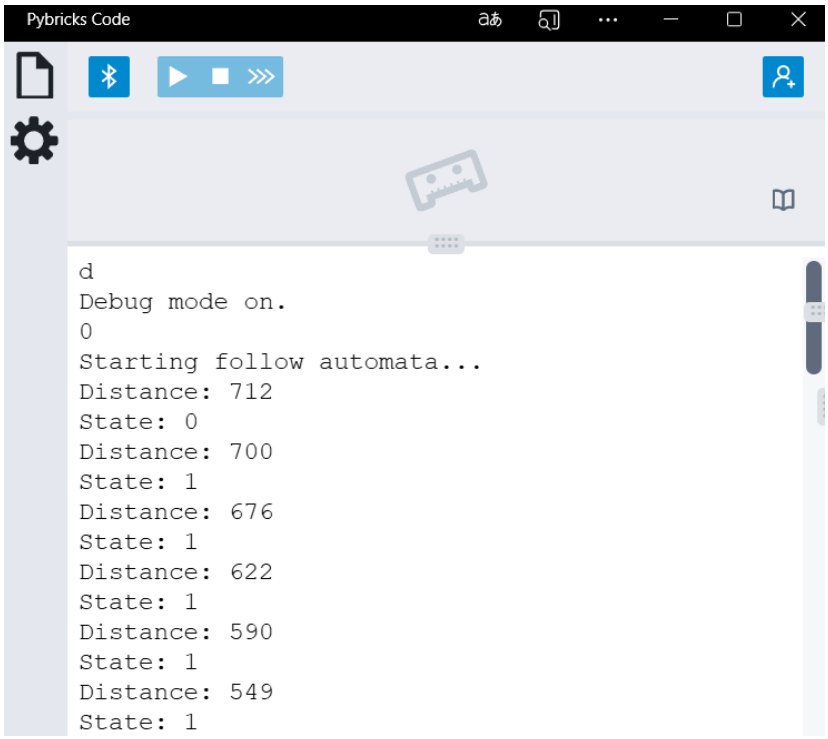
Na druhom obrázku 5.2 môžeme vidieť komplexnejšie správanie robota. Tento program popisuje robota, ktorý sa pohybuje po priestore a počas pohybu počíta objekty troch farieb. Okrem motorov a senzorov využíva globálnu premennú, lebo robot sa má zastaviť po tom, čo napočíta určitý počet objektov. Hoci je zobrazený iba jeden automat, v ľavom paneli možno vidieť, že program pozostáva z viacerých automatov.



Obr. 5.1: Ukážka stavového automatu vytvoreného našou aplikáciou



Obr. 5.2: Ukážka komplexného stavového automatu vytvoreného našou aplikáciou



Obr. 5.3: Ukážka webového rozhrania

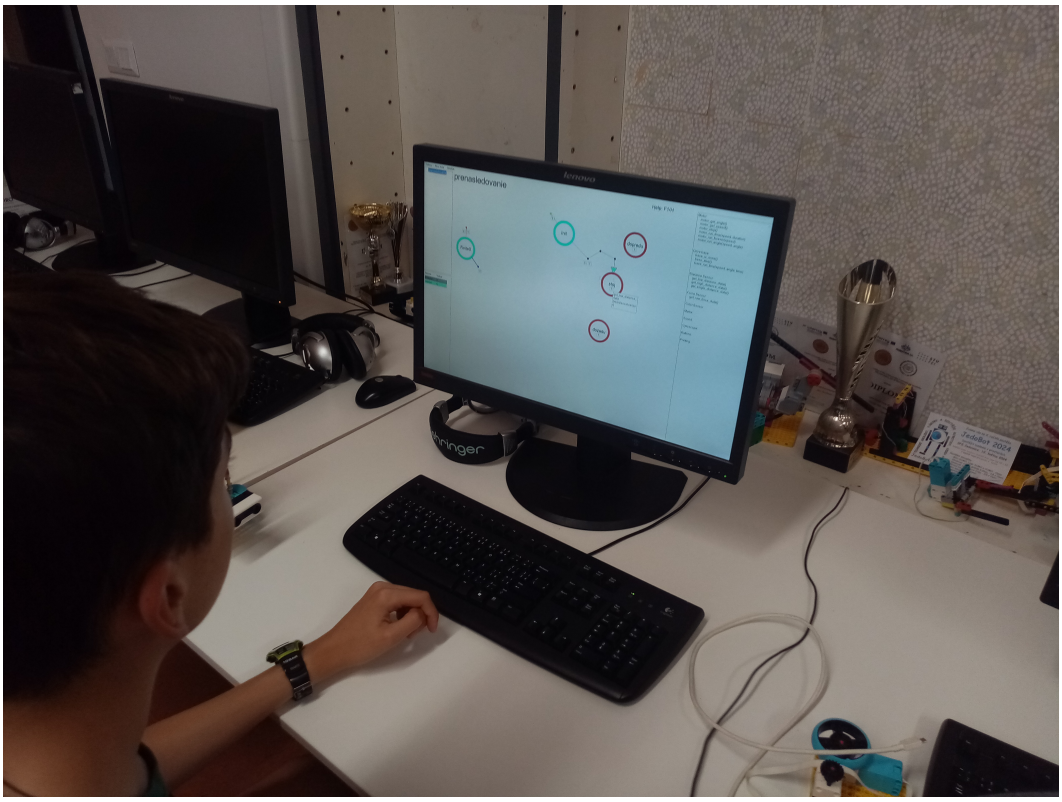
5.1.2 Webová stránka

Ako sme uviedli pri implementácii webového prostredia v časti 4.3, posielanie textových súborov cez tlačidlá by si vyžadovalo rozsiahlejšie zásahy do webového prostredia, na ktoré nemáme kapacitu a potrebovali by sme viac času na jeho úpravu.

Na komunikáciu preto momentálne používame časť webu, ktorá s robotom komunikuje pomocou konzoly, čo môžeme vidieť na obrázku 5.3. V strednej časti obrázka vidíme konzolu, v ktorej je písmeno "d", ktoré sme tam napísali pre zapnutie módu. Tento mód zapol výpis hodnôt senzorov a aktuálneho stavu, v ktorom sa robot nachádza. Výpis nastáva každých 200 milisekúnd. Ďalší riadok potvrdzuje vypnutie alebo zapnutie tohto módu. V treťom riadku je číslo 0, ktoré spúšťa prvý z troch predprogramovaných automatov. Tento automat predstavuje správanie, kedy sa robot snaží udržiavať vzdialenosť s objektom, ktorý je pred ním v rozmedzí on 150 do 350 milimetrov. Pri spustení a ukončení jednotlivých automatov nás o tom informuje výpis v konzole.

5.1.3 Firmvér

Firmvér od Pybricks je veľký a robustný. Pri implementácii sme narazili na množstvo problémov, ktorými boli napríklad chýbajúce znalosti hardvéru, nevysvetlených skratiek v názvoch funkcii a konštánt, slabá znalosť používania a operácie so smerníkmi. To zapríčinilo niekedy veľké problémy, ktoré si vyžadovali väčšie množstvo času na



Obr. 5.4: Fotodokumentácia z priebehu testovania na Robotickom krúžku.

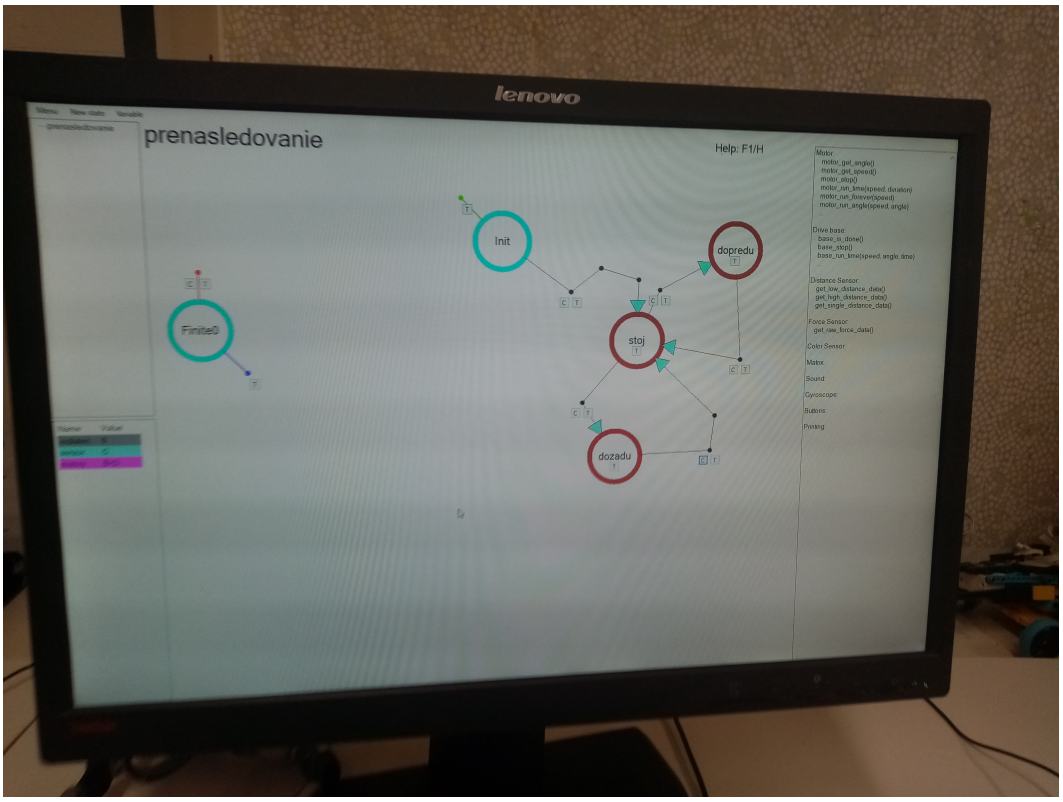
implementáciu a testovanie, než sa na začiatku očakávalo.

Úspešne sa nám podarilo vytvoriť interface pre volanie funkcií na obsluhu hardvéru, Vytvorenie globálnej pamäte, pomocou ktorej si funkcie vymieňajú parametre, s ktorými pracujú, správu funkčnosti komunikácie prostredníctvom Bluetooth-u ako pri vysielaní tak i pri prijímaní. Nakoniec sme implementovali spracovanie a vykonávanie nami vytvorených stavových automatov. To sme na konci podrobili taktiež testovaniu a nájdené chyby sme odstránili.

5.2 Testovanie

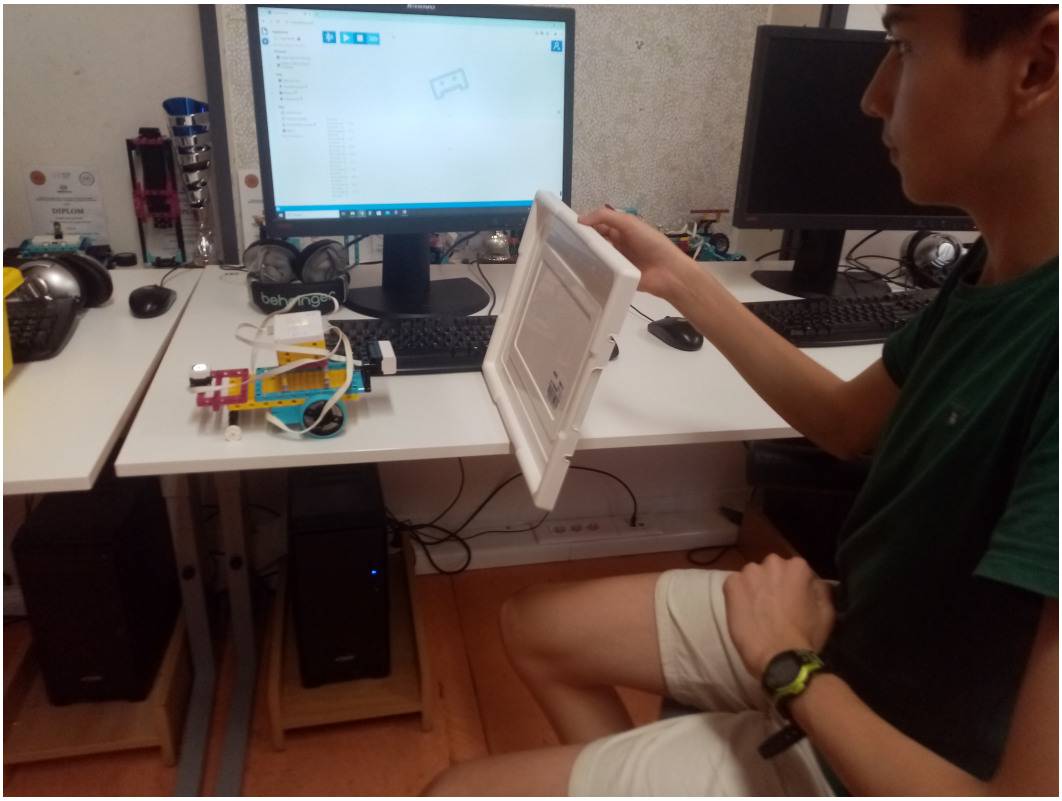
Tak ako sme si už spomínali vyššie, testovanie prebiehalo počas celej implementácie. Niekoľkokrát sme dali programy vyskúšať kolegom a na základe ich spätnej väzby sme chyby opravovali a náročné alebo zlé ovládanie upravovali a menili. Výsledkom bola aplikácia, ktorú sme dali otestovať účastníkovi Robotického krúžku pri Spojenej škole svätého Františka z Assisi v Bratislave, obr. (5.4).

V prvom kroku bol žiakovi vysvetlený a na príklade popísaný koncept stavových automatov. Následne mu bola ukázaná práca v aplikácii, aby získal prehľad čo všetko v aplikácii dokáže naprogramovať. Po tomto kroku dostal zadania, ktoré v asistencii riešil. Jednou z úloh bolo vytvoriť správanie robota, ktorý má udržiavať určitú vzdialenosť



Obr. 5.5: Na fotografii je program, ktorý vytvoril účastník testovania.

od prekážky, ktorá je pred ním. Účastník pochopil, že takéto správanie potrebuje po prvotnom vstupnom stave ďalšie tri aktívne stavy v ktorých bude získavať vzdialenosť od prekážky a tie si relevantne pomenoval 'Stoj', 'Dopredu' a 'Dozadu'. V ďalšom kroku ich prepojil stavovými prechodmi, kde nastavil podmienky prechodu a všetky úlohy, ktoré má robot pri zmene stavu vykonať. Výsledný stavový automat (5.5) následne spustil v robotovi, ktorý sa správal presne tak, ako zadanie popisovalo. Testovanie sme zachytili na fotografii na obr. 5.6. V písomnej spätnej väzbe účastník napísal: „*Tak páčilo sa mi, že všetko fungovalo a že je to taký zaujímavý spôsob programovania.*“



Obr. 5.6: Na fotografii je zaznamenané testovanie programu.

Záver

Hlavným cieľom tejto bakalárskej práce bolo vytvoriť nový spôsob, ako programovať správanie robotov zostrojených zo stavebnice Spike Prime od spoločnosti LEGO Education. Týmto spôsobom sú stavové automaty, ktoré vhodne vystihujú a popisujú správanie robotov. Pri tvorbe návrhu a implementácie sme sa pokúsili zachovať nenáročnosť programovacieho jazyka a priblížiť sa grafickému programovaniu, akým je Scratch. Dôvodom potreby tohto nového spôsobu je aj to, že pri komplexnejších problémoch sa programy v jazykoch Python a blokových jazykoch typu Scratch stávajú neprehľadné a ťažko modifikovateľné.

Aplikácia na tvorbu stavových automatov umožňuje užívateľom vytvárať programy popisujúce správanie robotov skĺbením grafického a textového programovania. Aplikácia je prehľadná a jednoduchá na používanie.

Webové rozhranie, ktoré slúži na komunikáciu medzi počítačom a robotom nebolo nijako upravované, ale zmenil sa spôsob používania. Namiesto doterajšieho nahrávania programov alebo ovládania robota pomocou konzoly, sa teraz konzola využíva na posielanie reprezentácii stavových automatov robotovi a interakciu s bežiacim programom.

Firmvér robota bol modifikovaný postupne v niekoľkých krokoch. Prvým bolo vytvorenie ľahšieho prístupu k ovládaniu hardvéru, ďalším bolo vytvorenie globálnej pamäte, ktorá slúži na posúvanie parametrov, ktoré potrebujú funkcie pre správne fungovanie. Tretím krokom bolo sprevádzkovanie Bluetooth komunikácie pre posielanie a prijímanie textových reťazcov. To je potrebné pre výpis premenných do konzoly, alebo pre prijímanie textových reprezentácií automatov, ktoré budú vložené do konzoly. Štvrtým krokom bolo spracovanie vstupu a vytvorenie vnútornej reprezentácie stavového automatu. Posledným bolo otestovanie behu programu a vyladenie všetkých chýb.

Implementované boli i moderné technológie ako napríklad knižnica slúžiaca na serializáciu alebo aplikácia vytvorená ako 'Windows Form Application'. Všetky tri časti boli priebežne testované a každá nájdená chyba programu bola opravená.

Navrhnutá aplikácia spĺňa stanovené ciele bakalárskej práce. Výsledkom je aplikácia, v ktorej užívateľ programuje roboty pomocou stavových automatov, webové rozhranie pomocou ktorého vie tieto programy posielať robotovi a firmvér, ktorý tieto stavové automaty spracováva a vykonáva.

V rámci budúcich možných vylepšení práce by bolo možné implementovať a rozšíriť

ju o ďalšie funkcionality. Jednou z možných úprav by mohlo byť v aplikácii vylepšiť grafickú stránku programu alebo implementovať rôzne zlepšenia používania tohto nástroja, napríklad kontrolu textového kódu. Iným rozšírením by mohlo byť webové prostredie, ktoré by sme modifikovali tak, aby bolo možné nahrávať celé súbory s reprezentáciou stavových automatov.

Literatúra

[1] BatteryPoweredBricks: WIP In depth Lego Dacta retrospective video series, dostupné online: https://www.eurobricks.com/forum/index.php?/forums/topic/190311-wip-in-depth-lego-dacta-retrospective-video-series/ (naposledy navštívené 17.05.2024)

[2] BatteryPoweredBricks: Lego Dacta Test, YouTube video, dostupné online: https://youtu.be/0pHY7IJ5T-o?si=Y-19RfcTDB0d5R6Y (naposledy navštívené 17.05.2024)

[3] Luciano F de Medeiros, Luiz Roberto Cuch: Robótica Educacional como Recurso Pedagógico para Alunos de Baixo Rendimento: Relato de Experiência, EDUCERE - XIII Congresso Nacional de Educação, 2017.

[4] Reduco Ltd.: FLL Casts, vzdelávací portál, dostupné on-line: https://www.fllcasts.com/ (naposledy navštívené 26.05.2024)

[5] LEGO Education, webstránka, dostupné on-line: https://education.lego.com/ (naposledy navštívené 18.05.2024)

[6] Brooks R., Cambrian Intelligence: The Early History of the New AI, A Bradford Book, 1999.

[7] Balogh, R, & Obdržálek, D., Using Finite State Machines in Introductory Robotics: Methods and Applications for Teaching and Learning, Robotics in Education, 2019.

[8] Hopcroft J. E., Ullman J. D., Introduction to automata theory, languages and computation, Addison-Wesley, 1939.

[9] Allauzen C., Mohri M., Efficient algorithms for testing the twins property, Journal of Automata, Languages, and Combinatorics, vol. 8, s. 117-144, 2003.

[10] Daniels, G., Mindstorms / SPIKE Prime / SPIKE Essential, GitHub repozitár, dostupné on-line: https://github.com/gpdaniels/spike-prime (naposledy navštívené 19.05.2024)

[11] Petrovič P., Stavebnice LEGO Mindstorms Education EV3, materiály ku školeniam, dostupné on-line: robotika.sk/events/18Skolenia/priruckaEV3.pdf (naposledy navštívené 19.05.2024)

[12] Wallace, E., Finite State Machine Designer, 2010. dostupné on-line: https://www.madebyevan.com/fsm/ (naposledy navštívené 19.05.2024)

[13] Pybricks, Robotics made easy, on-line aplikácia, dostupné on-line: https://pybricks.com/ (naposledy navštívené 19.05.2024)

[14] Pybricks, Robotics made easy, Github repozitár, dostupné on-line: https://github.com/pybricks (naposledy navštívené 19.05.2024)

[15] Figma, Figma Design, on-line aplikácia, dostupné on-line: https://www.figma.com/design (naposledy navštívené 05.06.2024)

[16] Chung C. J., Introduction to Lego Robotics with RCX Code Programming Language. Introductory workbook for Robofest Competition Teams, 2006. dostupné on-line: https://www.robofest.net/academy/rcxcode4robofest.pdf (naposledy navštívené 19.05.2024)

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu i samotná aplikácia pre tvorbu stavových automatov a zdrojový kód firmvéru. Tieto súbory sú taktiež zverejnené aj na stránke `https://davinci.fmph.uniba.sk/~vikiszaly1/bc/source.html`. Je možné ich nájsť v časti: *Zdrojové súbory tejto práce.*