Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Výučbový robot vytvorený na 3D tlačiarni

Bakalárska práca

2021
Martin Stolárik

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Výučbový robot vytvorený na 3D tlačiarni

Bakalárska práca

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

Bratislava, 2021
Martin Stolárik


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

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

**Názov:** Výučbový robot vytvorený na 3D tlačiarni
*3D Printed Educational robot*

**Anotácia:** Pre denné tábory vo Fablabe a tábor IT Akadémie boli použité dva lacné roboty, ktoré sú vhodné na vyučovanie programovania a 3D technológií: robot Otto a robot Mokrarosa. K obom existuje pomerne rozsiahly program na vytváranie a prehrávanie choreografií a obsluhu ich ostatných funkcií. Tento softvér však funguje len priamo na jednočipovom mikropočítači a komunikuje sa s ním cez znakový terminál (napr. Putty). Cieľom tejto práce je vyvinúť desktopovú aplikáciu s GUI, ktorá by komunikovala s takýmto robotom, graficky vizualizovala jeho stav, umožňovala ho programovať a spúšťať vytvorené programy v jednoduchej simulácii. Súcasťou práce je vytvorenie ukážkových programov.

**Kľúčové slová:** otto, mokrarosa, 3D tlač, výučbový robot

**Vedúci:** Mgr. Pavel Petrovič, PhD.
**Katedra:** FMFI.KAI - Katedra aplikovanej informatiky
**Vedúci katedry:** prof. Ing. Igor Farkaš, Dr.

**Dátum zadania:** 04.10.2020

**Dátum schválenia:** 06.10.2020 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

**Poďakovanie:** Ďakujem pánovi školiteľovi Mgr. Pavlovi Petrovičovi, PhD. za cenné rady, podporu, konzultácie i motivačnú kritiku počas tvorby aplikácie a textu.

# Abstrakt

V práci sa zaoberáme vývojom desktopovej aplikácie s grafickým používateľským rozhraním, modulárnym systémom, ktorý umožňuje programovať jednoduché edukačné roboty riadené mikropočítačom Arduino. Vzniká aplikácia primárne určená pre začínajúcich, mladistvých programátorov vo veku 10 až 15 rokov. Riadiaci program robota je možné tvoriť prostredníctvom navrhnutého vizuálneho programovacieho jazyka, ktorý je následne transformovaný do reprezentácie v jazyku C (používateľ jazyk C ovládať nemusí). Prepojenie s kompilátorom Arduino IDE umožňuje jednoduché nahranie vytvoreného programu priamo do mikropočítača. So spusteným riadiacim programom je následne, po pripojení rozhrania USB možné v termináli obojsmerne sériovo komunikovať. Súčasťou aplikácie je modul pre simuláciu, v ktorom možno zobraziť sériu pohybov jednoduchých choreografií vytvorených pre robota Otto.

**Kľúčové slová:** výučbový robot, arduino, otto, vizuálny programovací jazyk

# Abstract

In this work, we look into a development of a modular desktop application with a graphical user interface, which allows programming of educational Arduino–controlled robots. The application is primarily intended for beginners — youth programmers aged from 10 to 15. The control program for the robot can be created in a custom–designed visual programming language, subsequently translated into its equivalent representation in language C. This feature allows users to program the robot without prior knowledge of the C programming language. We utilized Arduino IDE compiler to easily upload program directly to the microcontroller. After upload, the application provides a terminal for two–way serial communication with the created program via a USB connection. The enclosed simulation module allows users to test simple choreographies for the robot Otto.

**Keywords:** educational robot, arduino, otto, visual programming language

# Obsah

# Zoznam obrázkov

# Úvod

Informatika je v dnešných časoch rozmáhajúcou sa vedou. Aplikáciou do praxe nám často uľahčuje prácu, šetrí čas. Pokrok v tejto oblasti je však (ako snáď aj v každej inej) podmienený neprestajným výskumom a objavovaním nových možností. Neoddeliteľnou súčasťou tohto procesu sú najmä ľudia investujúci doň čas, energiu a poznatky. Kde však takých nájsť? O svoju priazeň v očiach žiakov súperia dnes v edukačnom procese mnohé zaujímavé disciplíny.

Našim zámerom je podporiť výučbu robotiky a programovania u mladších, začínajúcich programátorov. Tieto odvetvia, veľmi príbuzné informatike, sú na edukačné účely priam ideálne — vo vhodnej forme umožňujú pomerne jednoducho, hravou formou a interakciou vzbudiť záujem a motiváciu pre ďalšiu prácu [47]. Cieľom je vytvorenie bádateľského prostredia umožňujúceho žiakom objavovať svet v tvorivom a konštrukcionistickom vzdelávacom procese, platformy, ktorá podporí ich osobnostný rozvoj po rozličných stránkach — z hľadiska priestorovej predstavivosti, abstraktného a formálneho myslenia, ale i získania praktických skúseností s artefaktmi z reálneho sveta. Používateľ tak nadobudne cenné poznatky pre štúdium teoretických princípov všetkých zapojených oblastí, najmä fyziky, informatiky, matematiky ale i iných predmetov.

Východiskom sú dva existujúce roboty vytvorené na 3D tlačiarni, Otto a Mokrarosa, ktorých prednosťami sú predovšetkým cenová dostupnosť a modularita. Momentálna konštrukcia umožňuje pohyb končatín, čím je možné vykonávať chôdzu, otáčanie (zmenu smeru) a tvoriť rôzne choreografie. Roboty sú tiež osadené jednoduchým mp3 prehrávačom a ultrazvukovým senzorom, umožňujúcim merať vzdialenosť od prekážky. Riadenie prístroja zabezpečuje mikropočítač Arduino Nano Strong.

Úskalím pre nováčikov v oblasti je zadávanie príkazov a programovanie robota, čo bolo doposiaľ možné vykonávať len cez znakový terminál. Znaky sa odosielajú cez sériový port do riadiacej jednotky prístroja, kde sú dekódované a interpretované napríklad ako pohyb konkrétneho motora. Cieľom práce je túto prekážku odstrániť vytvorením desktopovej aplikácie s grafickým používateľským rozhraním, ktoré by nahradilo rolu terminálu v procese interakcie s robotom. Hlavným cieľom je umožniť programovanie v prehľadnom vizuálnom jazyku. Tento typ jazyka pripomína puzzle, kde jednotlivými dielikmi sú komponenty bežného programovacieho jazyka (ako napríklad podmienka if alebo cykly či konštanty), ktoré do seba zapadajú, a tak je možné tvoriť kód. Viac o

ňom uvádzame v kapitole 4.1. Zámerom je tiež vytvoriť v aplikácii modul vizualizácie robota, v ktorom by bolo možné spúšťať vytvorené programy v jednoduchej simulácii bez prítomnosti robota samotného a modul ponúkajúci ukážkové programy, uľahčujúce prvotné zoznámenie s možnosťami aplikácie a robota.

V nasledujúcej kapitole uvádzame krátky prehľad vybraných historických udalostí vo vývoji robotiky a stručné predstavenie inštitúcií podporujúcich rozvoj tejto oblasti u nás na Slovensku. V kapitole 2 sa venujeme špecifikácii robotov Otto (časť 2.1.4) a Mokrarosa (časť 2.1.5), najmä ich konštrukcii a možnostiam riadenia. Kapitola 3 bližšie konkretizuje cieľ našej práce. O voľbe technológií vhodných pre implementáciu stanovených cieľov pojednáva nasledujúca kapitola 4. V ďalšom texte (kapitola 5) je predstavený návrh riešenia, implementačné detaily a rozhodnutia. Kapitola 6 je venovaná návrhu vizuálneho programovacieho jazyka, v ktorom je v aplikácii možné tvoriť riadiace programy robotov. Prácu uzatvára kapitola 7 predstavujúca niekoľko ukážkových programov vytvorených vo vzniknutej aplikácii.

# Kapitola 1

# Robotika

V kapitole uvádzame základné pojmy z oblasti, prehľad vybraných historických udalostí vedúcich k vzniku dnešných robotov a krátke predstavenie ich súčasného využitia. Zameriavame sa i na výučbu robotiky u nás na Slovensku, predovšetkým na možnosti štúdia a dostupnosť potrebných zdrojov pre mladších záujemcov.

## 1.1 Definície pojmov

***Robot*** je definovaný Medzinárodnou organizáciou pre normalizáciu (angl. International organization for standardization - ISO) ako „spustiteľný mechanizmus programovateľný v dvoch alebo viacerých osiach pohybu, so stupňom autonomity, pohybujúci sa v prostredí s účelom vykonať plánované úkony“. Norma ďalej uvádza, že súčasťou robota je riadiaci systém s rozhraním umožňujúcim obsluhu a tiež definuje základné rozdelenie na priemyselné a obslužné roboty [18]. Existuje viacero definícií, prienikom je požiadavka na programovateľnosť, autonomitu a schopnosť interakcie s prostredím. Niektoré kladú dôraz na podobnosť človeku, iné na viacúčelovosť [44]. Interakcia s prostredím znamená komunikáciu medzi robotom a prostredím. Robot získava informácie z prostredia použitím receptorov, rôznych senzorov merajúcich veličiny ako teplota, vzdialenosť, tlak či rýchlosť. Komunikácia opačným smerom prebieha pomocou efektorov, ktorými robot vykonáva akcie, zväčša sú efektormi motory rôznych typov.

***Robotika*** je veda zaoberajúca sa robotmi.

***Stupeň voľnosti*** (angl. degree of freedom - DOF) označuje pohyblivosť. Plne pohyblivé teleso má šesť stupňov voľnosti, posun a rotáciu v každej z osí 3D priestoru. DOF možno ľahko ilustrovať kĺbmi ľudského tela, napríklad koleno má jeden stupeň voľnosti. V robotike sa zvykne udávať DOF pre robota ako súčet DOF jeho kĺbov.

## 1.2 História

S veľkou pravdepodobnosťou sa môžeme domnievať, že prvý zrod robota, ako ho poznáme a definujeme dnes, bol len fikciou, nápadom či snom v mysli jedného z našich predkov [44]. Výskyt prvého zariadenia (stvorenia) pripomínajúceho robota možno hľadať i nájsť v mytológií, ktorá priam prekvitá postavami záhadne sa preberajúcimi k životu, pripomínajúcimi ľudí plniacich najrôznejšie úlohy. Slovo *robot* ako také sa objavuje až v roku 1920, v diele Karla Čapka — Rossum's Universal Robots, údajne odvodené zo slova „robotník" alebo „robota". Dielom autor upozorňuje na zneužitie robotiky, ale tiež ukazuje jej silu a potenciál. Robot sa krátko na to, v roku 1926, vyskytne i v kinematografii, konkrétne v sci-fi filme Fritza Langa s názvom Metropolis. Robotiku v literatúre neskôr reprezentoval známy Isaac Asimov, ktorý v svojich dielach zaviedol tri zákony robotiky podriaďujúce robotov človeku. Poukázal však aj na nedostatky a možné riziká spolužitia ľudstva a spoločnosti robotov.

Za dôveryhodnejších predchodcov dnešných robotov možno považovať rôzne mechanické vynálezy. Za jedno z prvých komplexnejších zariadení, ktoré sa môže uchádzať o toto označenie vzniklo v roku 270 p.n.l., keď v Grécku fyzik Ctesibus vytvoril prístroj na meranie času s vodným pohonom [44]. Vynálezov vzniklo pochopiteľne veľa, presuňme sa ale do doby, keď sa do oblasti prvýkrát vniesla možnosť programovateľnosti, nakoniec, je to jedna z podmienok stanovených v definícii pojmu „robot" [18]. Zlatý čas pre pokrok a inovácie, najmä v mechanizácií, známy tiež ako priemyselná revolúcia, priniesol ďalší výrazný míľnik na začiatku devätnásteho storočia. Tentoraz v odvetví spracovania hodvábu, keď vo Francúzsku Joseph Jacquard prvýkrát použil dierne štítky na ovládanie vzoru produkovaného šijacím strojom. Neskôr princíp štítkov použil Charles Babbage pri konštrukcií automatickej kalkulačky a analytického stroja, pre ktorý je často označovaný ako otec počítačov. Pre zaujímavosť dodáme, že mimo šijacieho stroja patrili k prvým oblastiam aplikácie automatizácie napríklad výroba skrutiek alebo konštrukcia žeriavu. Robotike výrazne pomohol i preslávený elektrotechnický inžinier Nikola Tesla, najmä pre svoje prínosy v súvise s objavom striedavého prúdu.

Po druhej svetovej vojne prispeli k vývoju v oblasti robotiky najmä nové vynálezy ako servomotor ale i pokroky v rozvoji digitálnej technológie. V roku 1954 si George C. Devol jr. patentoval vynález umožňujúci cyklickú kontrolu zariadení, ktorú po strete s odborníkom Josephom Engelbergerom využili pri vývoji prvého priemyselného robota. Robot s názvom „Unimate" vznikol v roku 1961, nahradil prácu robotníkov pri obsluhe stroja tvoriaceho odliatky a neskôr bol model použitý aj pri výrobe automobilov. Unimate sa na svojej pozícii osvedčil a začal tak éru automatizácie výrobných procesov pomocou robotiky. Oblasti aplikácie priemyselných robotov zahŕňali najmä zváračstvo ale známe boli aj roboty vykonávajúce maliarske práce.

V ďalšom období bolo v robotike snahou umožniť prístrojom vyššiu mieru interakcie s prostredím. Jeden z prvých pokusov o dosiahnutie tohto cieľa vznikol v Centre umelej inteligencie v Kalifornii. Centrum vyvinulo v rokoch 1966 až 1972 robota Shakey. Robot bol riadený počítačmi, kolesá mu umožnili pohyb v priestore, okolie vnímal videokamerou a niekoľkými senzormi. Okrem jednoduchého pohybu bol teda schopný aj základnej interakcie s prostredím a tým sa výrazne odlíšil od dovtedy známych zariadení. V tejto dobe sa o rozvoj vo veľkej miere pričinili i japonské spoločnosti, ktoré začali roboty nasadzovať do výrobných procesov.

Prvé roboty schopné komplexnejšej interakcie s prostredím narazili na komplikácie v súvislosti s potrebou náročnejších výpočtov, čo sa prejavilo na rýchlosti ich reakcií. Prvý systém pohybujúci sa na základe vyhodnotenia pozícií prekážok v okolí zo záberov z kamier sa pohyboval rýchlosťou jeden meter za 10 až 15 minút [44]. Zrejmé prednosti a potenciál robotov však nezostali bez povšimnutia a s rozvojom sa pokračovalo. Vznikli rôzne inštitúcie zaoberajúce sa robotikou, pomohol i rozsiahly vývoj v oblasti informatiky. Zvyšovala sa presnosť a rýchlosť, na riadenie sa začali používať dnes celkom bežné mikropočítače.

V snahe vysporiadať sa so stúpajúcimi nárokmi na výpočty vznikli i nové oblasti, napríklad „robotika založená na správaní“ (angl. behavior–based robotics). Riadenie v „klasickej“ robotike je založené na vopred definovaných (komplexných) modeloch, každý príkaz znamená vykonanie preddefinovanej postupnosti akcií efektorov, existuje v nej teda akýsi „centrálny modul riadenia“ [40]. Nový prístup riadenia je založený na „správaniach“ — jednoduchších, paralelne vykonávaných, interagujúcich moduloch, neustále vyhodnocujúcich situáciu (vstupy), ktoré transformujú na adekvátny výstup. Priamejšie prepojenie vstupu a výstupu na nižšej úrovni a upustenie od vytvárania zložitých modelov okolia prinieslo žiadanú pamäťovú i výpočtovú úsporu.

Zaujímavou sférou robotiky je takzvaná „sociálna robotika“, zameraná na vyššiu úroveň interakcie prístrojov a človeka [42]. Sociálny robot môže byť vnímaný viac ako spolupracovník, nie ako nástroj. Ich uplatnenie je najmä v oblasti vzdelávania či zdravotníctva. Za zakladateľa sociálnej robotiky je považovaný americký neurofyziológ William Grey Walter [17]. Výsledkom jeho snažení bol mimo iného vznik jedných z prvých autonómnych robotov „Machina Speculatrix“, známych tiež ako „korytnačky“, ktoré sa samostatne pohybovali, nasledujúc zdroj svetla.

Odvetvím, v ktorom možno využitie robotov a pokrokov v ich vývoji sledovať sa stal vesmírny výskum. Prvenstvo patrí Sovietskemu zväzu, ktorý vyslal prvú robotickú vesmírnu loď už v roku 1951. Súboj o ďalšie úspechy, najmä so Spojenými štátmi, upriamil na robotiku pozornosť. Roboty zmapovali najbližšie planéty, ich životnosť na povrchu cieľových planét sa predlžovala z desiatok minút na roky. Dnes sa prvenstvom — prvým humanoidným robotom vo vesmíre pýši americká agentúra NASA (angl. National Aeronautics and Space Administration — Národný úrad pre letectvo a vesmír).

## 1.3 Súčasnosť

Roboty sú dnes integrované v mnohých oblastiach ľudskej činnosti. Od jednoduchších systémov až po komplexnejšie, či tie ovládané umelou inteligenciou, roboty a s nimi spätá automatizácia nám pomáhajú šetriť náklady, zvyšujú produktivitu, eliminujú chyby. Pokročila i vízia robotov v umeleckej tvorbe, ktorá veští robotom do budúcna zlatú éru. V praxi možno najväčší prínos a najrozsiahlejšiu aplikáciu robotov vidieť v odvetviach priemyslu a medicíny. Roboty sú tiež vo veľkom využívané armádou, kde sa ako aj v iných oblastiach uplatňujú najmä pre presnosť a neúnavnosť. Rozšírila sa i teoretická časť robotiky a s pribúdajúcimi odvetviami aplikácie robotov vznikla aj potreba ich kategorizácie podľa možností mobility, účelu i technického vybavenia. Pomerne novou sférou sú lietajúce roboty, drony.

S nástupom robotiky vznikla tiež nutnosť výchovy expertov v tejto oblasti. Na tvorbe a prevádzke robota sa podieľajú odborníci z rôznych odborov, dôležitá je zložka dizajnu, konštrukcie i softvérového vybavenia obslužného programu. Učebné plány našich škôl sa však žiaľ nie vždy adaptujú na nové požiadavky v praxi s prijateľným oneskorením a v dostatočnej miere. Podpora robotiky je v úzadí najmä v prípade študentov základných škôl a gymnázií, vznikajú preto rôzne inštitúcie a projekty s cieľom motivovať potenciálnych záujemcov. Tu predstavujeme dva projekty, ktorých snažením je inšpirovaná i naša práca.

### IT Akadémia

IT Akadémia je jedným z národných projektov Centra vedecko–technických informácií SR (CVTI SR) [12]. CVTI SR je organizácia koordinujúca činnosť interdisciplinárnych výskumno-vývojových centier a národných infraštruktúr pre výskum, vývoj, inovácie a vzdelávanie [11]. Mimo iného sa venuje aj popularizácii vedy a techniky, s čím súvisí projekt IT Akadémie. Ako názov napovedá, jedným zo zámerov je podpora vzdelávania v oblasti informačných a komunikačných technológií (IKT) [19]. Cieľovú skupinu tvoria žiaci základných a stredných škôl. Podpora spočíva v modernizácii obsahu a metód výučby predmetov súvisiacich s IKT, rozširovaní palety dostupných študijných materiálov, ale i v doplňaní vedomostí vyučujúcich. Mladých majú k ďalšiemu štúdiu oblasti motivovať i krúžky, semináre, súťaže či tábory.

### Fablab

Fablab je celosvetová sieť laboratórií — dielní, voľne dostupných verejnosti, poskytujúcich prístup k moderným technológiám ako 3D tlač, CNC frézovanie, alebo 3D skenovanie [16]. Cieľom je poskytnúť prístup k prostriedkom komukoľvek, kto má záujem o rozvoj alebo vzdelanie v oblasti. Priestory sú k dispozícii aj pre testovanie prototypov. Fablab Bratislava je organizačnou súčasťou CVTI SR, v prevádzke je od roku 2014 [36]. Dielňa podporuje rôzne projekty a v rámci činnosti tiež organizuje tvorivé dielne, sústredenia a tábory pre mladších účastníkov vo veku 11-15 rokov, kde je im umožnené zoznámiť sa s novou technológiou, programovaním i robotikou hravou formou [14]. Pod názvom Denný tábor digitálnych technológií (DTDT) je jedna z takýchto akcií organizovaná každoročne od leta 2018.

# Kapitola 2

# Súčasný stav

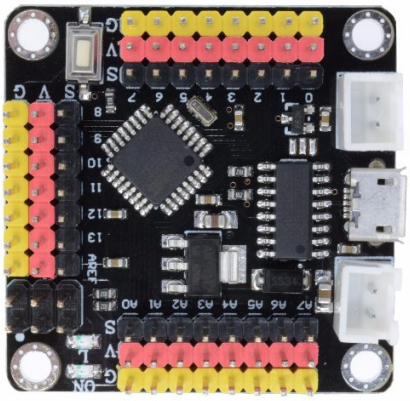
Kapitola predstavuje roboty Otto a Mokrarosa, na ktorých riadenie sa v práci zameriavame. Uvádzame možnosti a obmedzenia plynúce z aktuálnej implementácie ich riadiacich programov, ako aj z ich fyzickej konštrukcie. Súčasťou je prehľad podobných riešení, ktorými sa možno pri práci inšpirovať.

## 2.1 Robot Otto a Mokrarosa

Cieľom práce nie je stavba robota, zameriavame sa na riadenie existujúcich robotov, najmä modelu Otto a Mokrarosa. Tieto roboty boli použité vo výučbe v rámci denných táborov v bratislavskom Fablabe a tiež v tábore IT Akadémie. Sú teda testované v praxi, poznáme ich nedostatky ale i potenciál, ktorý máme v pláne rozvíjať. Na prvý pohľad možno badať podobu danú výrobným procesom, nakoľko oba roboty sú tlačené 3D tlačiarňou, rozmermi sú kompaktné, konštrukciou relatívne jednoduché. Modularita konštrukcie sľubuje rozšíriteľnosť v mnohých smeroch, roboty sú principiálne podobné no majú aj svoje špecifiká. Treba zdôrazniť, že ich primárnym určením nie je zdolávanie prekážok alebo fyzická manipulácia s nejakými predmetmi, aj keď nie je vylúčená. Primárne sa jedná o edukačné, demonštračné zariadenia.

### 2.1.1 Riadiaca jednotka Arduino

Arduino je open–source platforma, používaná vo viacerých projektoch na riadenie rôznych zariadení. Existuje viacero verzií, líšia sa výkonom i funkciami [38]. Niektoré ponúkajú integrované pripojenie k ethernetovej sieti či grafickým a zvukovým výstupným zariadeniam, rozdiely sú tiež v parametroch osadených procesorov a pamäte. Multifunkčnosť platformy je však skrytá najmä v možnosti softvérovej obsluhy analógových a digitálnych pinov. Použiteľné sú ako vstup alebo výstup a možno nimi zabezpečiť i obsluhu prerušení. Pomocou pinov je Arduino schopné ovládať motory, načítavať merané hodnoty zo senzorov i komunikovať s inými zariadeniami.



Obr. 2.1: Riadiaca jednotka Arduino Nano Strong

Hlavným komponentom, ktorý využívajú oba spomínané roboty ako riadiacu jednotku, je jednočipový mikropočítač Arduino Nano Strong (obrázok 2.1), klon originálu Arduino Nano V3.0. Mikropočítač riadi všetky funkcie robota, vysiela riadiace impulzy pre ovládanie pohybu servomotorov i vysielanie zvukových signálov a je tiež zodpovedný za komunikáciu so senzormi. Pre účely tejto práce je najdôležitejším parametrom daným použitím tohto komponentu pamäť. Konkrétne Arduino Nano Strong disponuje 32KB flash pamäte (z čoho 2KB využíva bootloader), pamäťou SRAM o veľkosti 2KB a 1KB pamäte typu EEPROM [1] [8]. Je teda zrejmé, že je žiaduce s pamäťou pri implementácii riadiacich programov pokiaľ možno čo najviac šetriť. Komunikácia s mikropočítačom prebieha pripojeným USB káblom, ktorým je možné nahrať do pamäte riadiaci program. Po spustení je možné s programom komunikovať buď to prostredníctvom už spomínaného rozhrania USB, alebo bezdrôtovo, čo umožňuje pripojený modul rozhrania Bluetooth (modul HC–05). Piny umožňujúce riadenie motorov a obsluhu senzorov je na obrázku 2.1 možno vidieť na hornej, ľavej a spodnej časti dosky.

### Programovanie mikropočítača Arduino

Programovací jazyk Arduino je založený na jazyku C++, sú v ňom však navyše dostupné rôzne funkcie a knižnice umožňujúce ovládanie špecifických prvkov mikropočítača, najmä riadenie periférií, senzorov a motorov pomocou pinov [7]. Periférie je teda možné jednoducho ovládať volaním príslušnej funkcie. K dispozícii je tiež voľne dostupný Arduino Software (IDE), v ktorom možno v textovom editore písať program, jedným kliknutím ho kompilovať, po pripojení USB kábla načítať do pamäte mikropočítača Arduino a spustiť. Program tvoria dve hlavné časti, metódy *setup* a *loop*. Metóda *setup* sa vykoná pri spustení programu práve raz, slúži predovšetkým na inicializáciu premenných, prípadne kalibráciu motorov, senzorov, určenie parametrov

[1] Electrically Erasable Programmable Read–Only Memory — elektricky zmazateľná pamäť ROM

pinov a podobne. Metóda *loop* je následne vykonávaná v nekonečnom cykle, obsahuje hlavný program. S ohľadom na obmedzenia plynúce z veľkosti dostupnej pamäte je kód písaný v jazyku C++ zvyčajne čo najjednoduchší, s preferenciou jednoduchých štruktúr a pamäťovo úspornejších prvkov jazyka C.

### Komunikácia — princíp

S mikropočítačom možno komunikovať napríklad pripojením rozhrania USB. Výmena informácií je založená na asynchrónnej sériovej komunikácii. Bity v sériovej linke sú prenášané postupne, v každom smere v jednom kanáli, čo oproti paralelnej komunikácii, vyžadujúcej niekoľko vodičov, prináša výraznú úsporu zdrojov (dostupných pinov) mikropočítača. Asynchrónnosť tiež znižuje nároky na komunikačný kanál, nakoľko komunikujúce zariadenia nezdieľajú pripojenie zabezpečujúce zosúladenie hodín [34]. Úspory v tomto smere nie sú bez následkov, prejavujú sa najmä v kvantite prenášaných dát, ku ktorým pribúda množstvo riadiacich informácií. Ich objem však nepredstavuje v tejto aplikácii prekážku, nakoľko ani samotné prenášané dáta nie sú zvyčajne rozsiahle.

Základom úspešnej asynchrónnej sériovej komunikácie je použitie rovnakého protokolu s rovnakými parametrami na oboch stranách. Ide najmä o prenosovú rýchlosť, počet bitov dát v pakete, počet synchronizačných (stop) bitov a určenie použitia bitu pre kontrolu parity. Pri komunikácii s mikropočítačom je štandardom rýchlosť 9600 bps, 8 bitov dát v pakete, jeden stop bit a informácia o parite (počtu kladných bitov v pakete) sa neprenáša [3].

### Komunikácia zo strany mikropočítača

Procesor mikropočítača rozpoznáva sériovú komunikáciu pomocou špecializovaného hardvéru — UART [31]. UART (angl. Universal Asynchronous Receiver/Transmitter) prevádza paralelné dáta (zo zbernice CPU) do sériovej podoby [9], pričom dve UART zariadenia sú schopné vzájomnej priamej sériovej komunikácie s použitím len dvoch prepojovacích vodičov. Činnosť UART možno softvérovo simulovať (napríklad použitím knižnice *SoftwareSerial* [2]) a tak pripojiť i viacero takto komunikujúcich zariadení.

V mikropočítači je sériový výstup UART vyvedený i na piny, takže možno ľahko priamo pripojiť i iné zariadenia (periférie) podporujúce tento štandard. Pre sériovú komunikáciu s počítačom prostredníctvom rozhrania USB je výstup z UART prepojený s „USB–to–serial“ čipom, ku ktorému existujú prislúchajúce ovládače pre OS. Činnosť tohto „prevodníka“ nemožno v mikropočítači softvérovo simulovať, v prípade potreby ich však možno pripojiť viacero (k simulovaným UART). V kóde je prístup k funkciám sériovej komunikácie umožnený objektom *Serial* a jeho metódami (napríklad *Serial.print()*, *Serial.read()*, *Serial.available()*). Komunikovať s perifériami možno i po-

užitím protokolov ako SPI alebo I2C, ku ktorým sú tiež k dispozícii obslužné funkcie.

#### Rozhranie Bluetooth

Komunikovať s mikropočítačom možno aj prostredníctvom rozhrania Bluetooth, o ktoré bolo použité Arduino rozšírené [45]. Jeho zapojenie a implementácia ilustruje prípad softvérovej simulácie ďalšieho UART modulu. Pri konštrukcii robota vznikla táto nutnosť, nakoľko použitý mikropočítač disponuje jediným UART modulom, ktorý je využívaný pre komunikáciu s počítačom pripojením USB. Zapojenie modulu rozhrania Bluetooth (samostatná súčiastka) nemožno zlúčiť s existujúcim modulom UART, nakoľko potom dochádza k interferencii so sériovou komunikáciou prebiehajúcou cez USB (napríklad pri nahrávaní riadiaceho programu) [45].

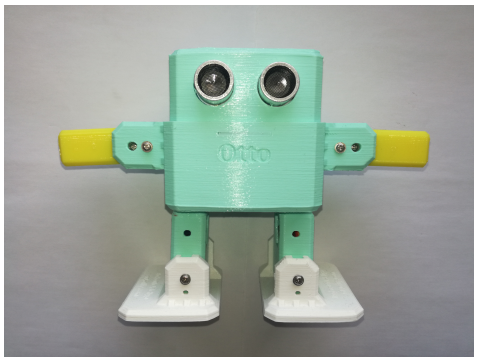
### 2.1.2 Riadiaci program

Riadiaci program robota je vždy možné vytvoriť priamo ako program pre Arduino popísaný vyššie. V prípade denných táborov bolo ale zvolené iné riešenie, umožňujúce riadiť robota aj bez znalosti C++. Princíp je nasledovný. Riadiaci program napísaný v C++ prijíma počas behu, v časti loop, cez sériový port (pripojenie rozhraním USB alebo bezdrôtovou technológiou Bluetooth) postupnosť znakov, ktoré majú predurčený význam. Znaky rozpoznáva, a interpretuje ako príkazy. Na strane užívateľa je teda riadenie robota vykonávané zadávaním reťazca do znakového terminálu (napríklad prostredníctvom programu Putty v OS Windows). Popísaným spôsobom je možné riadiť robota manuálne ale i vytvárať jednoduché choreografie. Viac o programovaní a ovládaní uvádzame v častiach venovaných samostatným robotom, nakoľko ich možnosti nie sú totožné. Výhodou prístupu je mimo iného fakt, že pri ladení choreografie nie je potrebné zakaždým kompilovať program pre Arduino. Ústredným nedostatkom je na druhú stranu výraznejšie obmedzená dĺžka choreografie, keďže je nutné všetky jej prvky reprezentovať a uchovať v operačnej pamäti.

### 2.1.3 Interakcia s prostredím

Roboty interagujú s prostredím použitím receptorov, ktorými získavajú údaje z prostredia a efektorov, ktorými prostredie ovplyvňujú. K mikropočítaču Arduino možno pripojiť rôzne periférie. Aktuálna konštrukcia robotov Otto a Mokrarosa zdieľa použitie komponentov servomotor a ultrazvukový senzor.

Servomotor je typ motoru, ktorého úlohou väčšinou nie je kontinuálny pohyb (i keď nie je vylúčený) ale presné natočenie osi. Motor teda disponuje zariadením na určenie polohy, a na základe riadiacej informácie vždy prispôsobí svoje natočenie. V prípade robotov bolo pri stavbe použité mikroservo SG-90, umožňujúce nastavenie polohy v




Obr. 2.2: Robot Otto

rozmedzí 180 stupňov. Pre riadenie je dostupná knižnica a nastavenie polohy jednotlivých motorov je tak v kóde len jednoduchým volaním príslušnej funkcie s parametrom definujúcim požadované natočenie.

Ultrazvukový senzor umožňuje bezkontaktne určiť vzdialenosť od prekážky, zisťuje ju podobne ako v prírode netopier. Princíp je založený na sonare, zariadenie pozostáva z vysielača vysokofrekvenčného (pre človeka nepočuteľného) zvuku a prijímača. Na začiatku merania je odvysielaný signál a následne sa čaká na jeho prijatie po odraze od prekážky. Dôležitý je práve časový rozstup vysielania a prijatia „odrazeného“ zvuku. Keďže rýchlosť šírenia zvuku vzduchom poznáme, vieme ľahko vypočítať prekonanú vzdialenosť za daný čas. Model použitý pri stavbe robotov nesie označenie HC-SR04.

### 2.1.4 Robot Otto

Vznikol v rámci projektu Otto DIY v susednej Českej republike, s cieľom vytvoriť dostupného robota pre širokú verejnosť. Dnes je vyrábaných niekoľko verzií, robot je modulárny a všetok softvér a hardvér s ním spojený voľne dostupný v duchu open–source [30]. Otto bol ústrednou témou Denného tábora digitálnych technológií v roku 2018, kde sa ukázalo, že je vhodný pre demonštračné a edukačné účely pre začínajúcich programátorov. Modularita a open–source princíp sľubujú potenciál aj do budúcnosti, a aj preto je súčasťou našej práce.

#### Konštrukcia, možnosti, schopnosti

Vzhľadom sa snaží napodobniť človeka, má štyri končatiny, dve „nohy“, každá poháňaná dvoma servomotormi, mu umožňujú pohyb do všetkých strán roviny. Môže tiež pohybovať „rukami“ v rozmedzí 180 stupňov, „oči“ tvorí ultrazvukový senzor schopný merať vzdialenosť od prekážky, zhora na hlave sú k dispozícii dve dotykové tlačidlá, ktoré možno použiť ako ďalšie vstupné zariadenie. V tele je osadený reproduktor a mp3 prehrávač. Robota Otto môžete vidieť na obrázku 2.2.

**Riadiaci program, interakcia, programovanie**

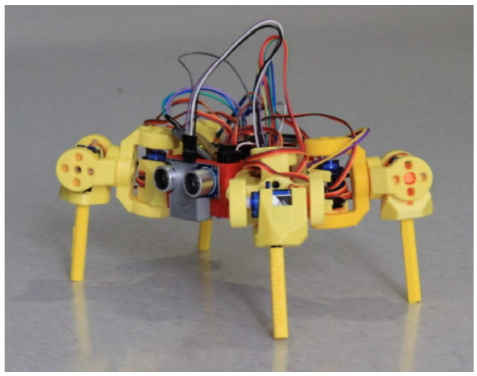
Program vykonávaný mikropočítačom Arduino nie je prebraný od vynálezcov pôvodnej verzie robota. Vytvoril ho špeciálne pre účely denného tábora školiteľ tejto práce, Mgr. Pavel Petrovič, PhD. Ako spomíname v úvode, jedná sa o program umožňujúci programovanie robota bez znalosti C++. V najnovšej verzii (leto 2020) sú možné dva varianty interakcie s robotom, priama interakcia a jednoduché programovanie.

Robota možno ovládať priamo. V tomto režime zariadenie vykonáva príkazy (znaky) prijaté cez sériový port ihneď. (Príkazy sú v komunikácii reprezentované znakmi a riadiaci program ich po prečítaní interpretuje ako príkazy.) Takéto povely sú charakteru „posuň motor A do polohy X“ alebo „pípni“. Možno tiež spustiť prehrávanie niektorej z predprogramovaných melódií alebo choreografií, ovládať mp3 prehrávač, kalibrovať pozície servomotorov a vytvorenú kalibráciu uložiť do pamäte EEPROM.

Druhým prístupom je programovanie choreografií a ich následné spúšťanie. Jedným z interpretovaných znakov — príkazov je aj špeciálny znak na prechod do režimu programovania. Tu je možné zadať program ako postupnosť trojíc (doba čakania v milisekundách, číslo motora, rotácia motora v stupňoch) a niekoľkých špeciálnych príkazov (tiež ako trojice), ktoré umožňujú napríklad „skoky“ v programe (v podstate inštrukcia *jump* — vykonávanie programu pokračuje zadaným riadkom kódu, za čo sa považuje každá takáto trojica). Súčasťou sú tiež trojice — inštrukcie na prehranie melódie, prípadne zvukového efektu. Krátku choreografiu zadanú v tomto formáte môžete vidieť v ukážke 2.1. Takto vytvorený program je ukladaný do pamäte RAM, v riadiacom programe vykonávanom v mikropočítači sú spomínané trojice reprezentované ako prvky poľa celých čísel. Programovanie je teda značne limitované veľkosťou pamäte a často neprehľadné. Upraviť nahraný program nie je možné priamo v mikropočítači, je nutné nahrať nový, čo v prípade ladenia choreografie môže znamenať časté, nepraktické prepínanie medzi terminálom a textovým editorom. Výhodou však stále bezpochyby zostáva možnosť interakcie s robotom a tvorby choreografie bez znalosti C++ a tiež bez nutnosti kompilácie a nahratia riadiaceho programu pri zmene kompozície pohybov.

```
1 @1  10  3
2 1  11  1
3 1  12  1
4 1000  5  0
5 0  0  0
```

Ukážka kódu 2.1: Príklad choreografie tobota Otto zadanej ako trojíce čísel



Obr. 2.3: Robot Mokrarosa

### 2.1.5 Robot Mokrarosa

Robot MoKraRosA (MOdulárny KRAhulský RObot S Arduinom) bol témou Denného tábora digitálnych technológií v roku 2019 [46]. Vyvinutý je od základu v dielni Fablab Bratislava kde bolo toho roku skonštruovaných 80 kusov s podobným cieľom ako predtým robot Otto [33].

#### Konštrukcia, možnosti, schopnosti

Telo robota Mokrarosa pripomína pavúka (obrázok 2.3). Aj tu bol použitý rovnaký modul pre senzor merania vzdialenosti od prekážky, ktorý v dizajne imituje „oči“ prístroja. Pavúk je navyše osadený gyroskopom, ktorý mu umožňuje reagovať na preklopenie, prípadne pád. Konštrukcia dovoľuje nohami robota prevrátiť alebo vykonať kotúľ, pričom na základe údajov z gyroskopu môže vždy nasmerovať" nohy smerom k zemi, a tak sa nikdy nedostane do polohy, z ktorej by nebol možný pohyb (na rozdiel od robota Otto). Štandardne je tiež osadený mp3 prehrávačom a reproduktorom.

#### Riadiaci program, interakcia, programovanie

Riadiaci program pavúka je o niečo zložitejší. Umožňuje jednak robota riadiť „po znakoch“, v tomto móde ihneď reaguje na symbol prijatý cez sériový port, obdobne ako v prípade robota Otto. V režime programovania sa výraznejšie líši. Program je daný ako postupnosť n-tíc kde každá n-tica vyjadruje v akej polohe (v stupňoch) sa má v daný moment daný motor nachádzať. Je tak možné hýbať s viacerými motormi naraz. Podstatný rozdiel je tiež v postupe programovania. Program je možné tvoriť tak, že robota postupne navádzame v režime „po znakoch“, manuálnym pohybom jednotlivými motormi do požadovanej polohy a následne uložíme stav — natočenie každého z motorov do spomínanej n-tice. Takto postupne vytvárame „pózy“ — konfigurácie, cez ktoré robot následne plynule prechádza. Vytvorenú choreografiu možno opakovane ladiť krokovaním a úpravou vybranej konfigurácie.

## 2.2 Existujúce riešenia

Jediné nám známe dostupné príbuzné riešenie súvisí s robotom Otto. Pochádza priamo od jeho tvorcov, ktorí vyvinuli pomerne rozsiahlu aplikáciu na tvorbu riadiacich programov. V prípade robota Mokrarosa podobné riešenia pochopiteľne neexistujú, nakoľko je len nedávnym produktom dielne Fablab Bratislava. Uvádzame však i iné produkty súvisiace s výučbou robotiky a programovaním robotov.

### 2.2.1 Otto Blockly

Ide o desktopovú aplikáciu z produkcie konštruktérov robota Otto, umožňujúcu programovať robota pomocou vizuálneho jazyka, vytvoreného pomocou knižnice Google Blockly [29]. Produktom sa možno vo viacerých aspektoch inšpirovať. Aplikácia je profesionálne graficky spracovaná, podporuje rôzne funkcie robota, ponúka okamžitý preklad vytvoreného kódu riadiaceho programu do C++, ktorý možno ľahko kompilovať a nahrať priamo do pripojeného mikropočítača Arduino. Vo vizuálnom jazyku sú k dispozícii prvky ovládajúce pokročilé funkcie rôznych verzií robota. Súčasťou sú predpripravené ukážkové kódy a tiež možnosť komunikácie cez sériový port. V čase vzniku tohto textu je k dispozícii verzia 1.3.0.

Nevýhodou pre nás je predovšetkým naviazanosť na robota z výroby autorov. Nami používaný klon robota Otto je modifikovaný, a tak by pre použitie s aplikáciou bolo potrebné s každou zmenou zapojenia zmeniť aj logiku generátora kódu v aplikácii. Rovnako aplikácia nepodporuje štýl programovania kde sú vytvárané choreografie ukladané bez potreby kompilácie do pamäte RAM. Tento prístup je pritom mimoriadne výhodný pri častom ladení jednoduchých choreografii. Našim cieľom je vytvoriť alternatívny, nezávislý softvér, ktorého životný cyklus, dynamika a rozšíriteľnosť nie je naviazaná na spomínaný projekt.

### 2.2.2 LEGO Mindstorms

Jedná sa o komplexnú stavebnicu umožňujúcu stavbu a programovanie robota [24]. Základom je riadiaca jednotka, do ktorej možno pripájať senzory a motory, avšak ich počet je v základnej verzii limitovaný dostupnými portami na maximálne štyri motory a štyri senzory, možno však dokúpiť (neoficiálne) „rozbočky“ a zapojiť i väčší počet. K stavebnici možno pripojiť rôzne receptory, gyroskop, kompas, svetelný či ultrazvukový senzor. Všetky periférie sú s riadiacou jednotkou spojené vodičmi zakončeniami pripomínajúcimi telefónne káble. Modularita systému je ohraničená dostupnými produktmi ponúkanými firmou LEGO, výber je rozmanitý, no komponenty nie sú až tak cenovo prívetivé. Pre porovnanie uvádzame cenu ultrazvukového senzora — LEGO senzor 34,99€ [25] a senzor použitý pri stavbe robota Otto 0,70€ [35].

K hardvéru ponúka LEGO i prislúchajúci softvér na tvorbu riadiacich programov. Oficiálna aplikácia umožňuje programovanie vo vizuálnom jazyku, kde používateľ v grafickom rozhraní vytvára kód ako postupnosť ikon, prvkov reprezentujúcich senzory, motory ale napríklad i cykly. Ikony možno konfigurovať, a tak nastavovať parametre ikonou reprezentovaného úkonu. Ikona „pohyb motora“ má napríklad parametre ako port, v ktorom je zapojený motor, rýchlosť, smer a miera otočenia.

Plynulý prechod programátorov od vizuálnych jazykov k bežným textovým v prípade robotov rady Mindstorms umožňujú rôzne alternatívne programovacie platformy, kde možno riadiaci program tvoriť v jazyku Python (EV3Python) či C (RobotC) [1].

# Kapitola 3

# Cieľ

Tu je opísaný predmet riešenia, ciele práce a požiadavky na výsledný produkt.

## 3.1 Primárny cieľ

Ústredným cieľom tejto práce je vytvorenie desktopovej aplikácie s grafickým používateľským prostredím (angl. GUI — graphical user interface), ktoré uľahčí programovanie robotov Otto a Mokrarosa, opísaných v časti 2.1. Ako prioritu stanovujeme podporu pre robota Otto, ktorý je konštrukciou o čosi jednoduchší. Programovanie zariadení máme v úmysle zjednodušiť najmä použitím vizuálneho programovacieho jazyka (VPJ), o ktorom viac uvádzame v kapitole 4.1.

Inšpiráciou je nám predovšetkým aplikácia vytvorená konštruktérmi pôvodnej verzie robota Otto (Otto Blockly, časť 2.2.1), ktorá spomínaný typ jazyka využíva. V základnej verzii našej aplikácie je žiaduce vytvoriť prvky podporujúce funkcionality dostupné v aplikácii Otto Blockly. Jedná sa najmä o prvky obsiahnuté vo VPJ, umožňujúce jednoduchý pohyb motormi, obsluhu senzorov, ale i komplexnejšie funkcie. V ďalšom je tiež potrebné umožniť kompiláciu vytvoreného kódu a jeho nahratie do riadiacej jednotky robota. Nezabúdame ani na funkciu umožňujúcu jednoduchú sériovú komunikáciu s riadiacou jednotkou pripojením rozhrania USB a možnosť načítania predpripravených ukážkových programov.

Výraznejšie sa odlíšiť od existujúcej aplikácie máme v pláne tvorbou modulu vizualizácie, v ktorom by bolo možné spúšťať vytvorené programy v jednoduchej simulácii. Pod „jednoduchosťou“ si predstavujeme zobrazenie tela robota v trojrozmernom priestore, umožnenie pohybu jednotlivými časťami (končatinami) na základe vytvoreného programu, a pôsobenie gravitácie na zobrazovaný model. V simulácii je prirodzene snahou čo najvierohodnejšie napodobniť realitu.

K odlíšeniu od aplikácie Otto Blockly prispeje tiež podpora tvorby choreografií ako postupnosti pohybov uchovávanej čisto v operačnej pamäti bez nutnosti kompilácie.

## 3.2 Požiadavky

Okrem základných (implicitných) požiadaviek na spoľahlivosť a funkčnosť je dôležité zdôrazniť ďalšie, plynúce z určenia cieľovej skupiny používateľov novovzniknutej aplikácie. Používateľmi má byť najmä veková kategória 10-15 rokov, čo je pomerne široké rozpätie. Potvrdila nám to i doc. PaedDr. Monika Tomcsányiová, PhD. z katedry didaktiky matematiky, fyziky a informatiky. Problémom je, že interval zahŕňa rôzne kognitívne vývinové štádiá žiakov, ktoré sa výrazne líšia napríklad v schopnosti spracovať abstrakciu. Je teda potrebné, aby aplikácia poskytla vo VPJ prvky zohľadňujúce túto skutočnosť a mladším programátorom umožnila jednoduchšiu tvorbu kódu.

# Kapitola 4

# Technológie

Kapitola pojednáva o voľbe vhodných technológii pre implementáciu jednotlivých cieľov práce. Detailom implementácie s použitím tu opísaných technológií je venovaná nasledujúca kapitola 5.

## 4.1 Editor kódu

Nakoľko je možnosť tvorby riadiaceho programu robota vo vizuálnom programovacom jazyku (VPJ) našim ústredným cieľom, podriaďujeme mu výber ostatných technológií.

Koncept VPJ nie je novinkou. Začiatky rozvoja v tejto oblasti možno badať už v šesťdesiatych rokoch minulého storočia s nástupom počítačovej grafiky samotnej [41]. Myšlienka je to pritom takmer prirodzená, obrazové vnímanie je človeku blízke, vizuálne vnemy si ľahšie pamätáme, sú intuitívne. Ako označenie napovedá, tento typ jazyka sa vyznačuje práve tým, že nie je tvorený textovými prvkami, ako mnohé bežné jazyky. Elementy, z ktorých možno tvoriť program sú grafické, ide teda zväčša o rôzne ikony, šípky, obrazce či diagramy, ktorých rozmiestením a prepojením tvoríme kód.

Dnes je táto oblasť rozvinutá a vizuálne jazyky vieme klasifikovať do niekoľkých kategórií. Zaujímavou je kategória takzvaných „čistých“ vizuálnych jazykov, ktoré sú kompilované z grafickej formy priamo do strojového kódu. Pre našu aplikáciu je však podstatná druhá z hlavných kategórií, obsahujúca „hybridné textovo–vizuálne systémy“, do ktorej spadajú i tie, umožňujúce tvorbu programu vo vizuálnej podobe, ktorá je ale následne preložená do textovej reprezentácie v niektorom z textových jazykov. Práve tento koncept je pre našu aplikáciu vhodný, vytvoríme VPJ s adekvátnymi prvkami, preložiteľnými to jazyka C, ktorému „rozumejú“ riadiace jednotky robotov.

Potenciál VPJ je predovšetkým v možnosti interaktívnej tvorby kódu (napríklad formou drag and drop), ich názornosti a jednoduchosti umožňujúcej v prípade potreby vysokú abstrakciu od textovo orientovaného jazyka, do ktorého sú prekladané. V mnohých prípadoch sú tak ideálnym riešením pre začínajúcich programátorov aj ako istý

medzistupeň pri výučbe programovania v textovom jazyku. V porovnaní s textovou formou sú vizuálne jazyky v nevýhode hlavne v komplexite, obrazová reprezentácia viac zaťažuje pamäť i procesor. S narastajúcou dĺžkou kódu môžu byť tiež menej prehľadné, ťažko udržiavateľné i rozšíriteľné.

### LabVIEW

Laboratory Virtual Instrument Engineering Workbench (LabVIEW) je výtvorom spoločnosti National Instruments, zaoberajúcou sa tvorbou softvéru a zariadení pre automatické testovanie. Produkt má širšie spektrum použitia, mimo iného je s jeho pomocou možno vytvárať riadiace programy pre vnorené systémy v jazyku G [39].

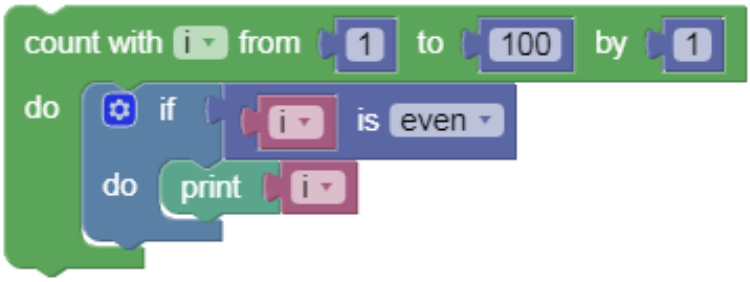
Základným prvkom jazyka je takzvaný VI — „virtuálny prístroj“ (angl. Virtual Instrument). VI môže predstavovať celý program alebo byť súčasťou väčšieho programu. Pre každý VI používateľ definuje prvky „predného panela“, zobrazujúceho stav prístroja. Pomocou panela je tiež možné definovať vstupy a po výpočte získať výstupné hodnoty. VI ďalej tvorí „diagram blokov“, definujúci správanie VI vo vizuálnej podobe. Bloky reprezentujú funkcie, k dispozícii sú rôzne, od jednoduchých aritmetických operácií až po komplexnejšie. V diagrame je umožnený prístup k vstupom definovaným prostredníctvom predného panela. Bloky sú prepájané „vodičmi“ reprezentovanými čiarami, znázorňujúcimi tok dát. Postup vyhodnocovania vstupu je založený na dostupnosti dát — blok sa vyhodnotí v momente keď sú preň dostupné všetky vstupné hodnoty a výsledok je následne dostupný pre ďalšie prepojené bloky alebo odoslaný na výstup (predný panel).

Na LabVIEW je založená aplikácia firmy LEGO na tvorbu riadiacich programov robota Mindstorms (kapitola 2.2.2). Nevýhodou pre našu aplikáciu je najmä spoplatnenie tohto softvéru, napriek tomu je to jedno z možných riešení.

### Blockly

Blockly je produktom firmy Google. Ide o knižnicu vytvorenú v jazyku JavaScript ako nástroj na tvorbu VPJ. Dostupná je v duchu open–source bez poplatkov a nesie licenciu Apache 2.0, ktorá ju umožňuje použiť k akémukoľvek účelu aj v modifikovanej verzii. Jazyky tvorené touto knižnicou spadajú do kategórie hybridných VPJ, kód v nich tvorený je prekladaný do textového jazyka a až následne kompilovaný do strojového kódu.

Základným stavebným prvkom jazykov vytvorených pomocou Blockly je „blok“, ktorý možno prirovnať k dieliku puzzle alebo akejsi kachličke [48]. Bloky sú rôznych typov a zvyčajne reprezentujú prvky cieľového textového jazyka ako cykly, premenné, procedúry, aritmetické operácie a iné. Dôležitou súčasťou novovzniknutých jazykov sú ale bloky vytvorené špecificky pre danú oblasť. V našom prípade to môžu byť dieliky




Obr. 4.1: Použitie blokov v knižnici Blockly

sprístupňujúce používateľov rôzne funkcie robota ako ovládanie motorov či vydávanie zvukových signálov. Výsledný kód je tvorený spájaním blokov do väčších celkov. Príklad bloku vykonávajúceho (neefektívny) výpis párnych prirodzených čísel menších alebo rovných ako sto môžete vidieť na obrázku 4.1.

Blockly pozostáva z dvoch hlavných častí, definícií blokov a definícií generátorov, zabezpečujúcich preklad do cieľového textového jazyka. Od vývojárov sú k v knižnici k dispozícii „od výroby“ bloky a generátory pre jazyky JavaScript, Lua, PHP, Dart, a Python. I keď našim cieľom je jazyk C, existujúce časti nám môžu byť nápomocné.

Výhodou je vysoká prispôsobiteľnosť blokov po dizajnovej stránke a tiež možnosť konfigurácie spojov medzi dielikmi tak, aby vynucovali syntaktické pravidlá cieľového jazyka a vykonávali typové kontroly. Benefitom je i neutíchajúci aktívny vývoj a dostupnosť dokumentácie. Integrovanie knižnice do aplikácie rovnako netvorí prekážku, nakoľko JavaScript možno spúšťať v prehliadačoch, ktoré sú bežnou súčasťou vývojových platforiem ako komponent „WebView“. Knižnica je nasadená v mnohých kontextoch, osvedčená je v oblasti výučby (napríklad projektom Scratch) ale i robotiky, tu je dôkazom napríklad aplikácia RoboBlockly alebo už spomínaný projekt Otto Blockly (kapitola 2.2.1) a aj preto sme sa rozhodli pre jej použitie v našej práci.

## 4.2 Implementačný jazyk aplikácie

Výber knižnice Blockly ako nástroja pre tvorbu vizuálneho jazyka nabáda k vývoju internetovej aplikácie a nie desktopovej. Napriek tomu, že ide o JavaScript knižnicu, podriaďujeme ju desktopovej aplikácií, nakoľko bežný internetový prehliadač nedisponuje funkciami potrebnými pre splnenie ďalších cieľov našej práce. Problematickou je napríklad komunikácia s robotom prostredníctvom sériového portu, ku ktorému nezvykne byť v prehliadačoch možný prístup pre bezpečnostné riziko.

Rovnako môže byť komplikované spúšťať iné desktopové aplikácie (napríklad prístupom k príkazovému riadku), čo je nevyhnutné pre spustenie Arduino kompilátora

po vytvorení riadiaceho programu. Implementácia týchto funkcionalít v rámci internetovej aplikácie nie je nemožná, no vyžadovala by napríklad vývoj a inštaláciu rozšírení webových prehliadačov.

Zohľadňujúc požadované funkcionality, popularitu jazykov a skúseností autorov práce, je voľbou pre implementáciu jazyk Java. Java poskytuje rozsiahly systém na tvorbu grafických používateľských rozhraní — JavaFX. Ten umožňuje oddeliť v kóde definície logiky a vzhľadu (rozloženia), na ktoré slúži formát XML. Vzhľad a správanie jednotlivých grafických prvkov je pritom možné ľahko ovládať i v Java kóde. Súčasťou systému JavaFX je komponent WebView slúžiaci ako prehliadač, v ktorom možno spustiť grafické rozhranie knižnice Blockly a tiež dovoľuje obojsmernú komunikáciu medzi jazykom JavaScript a Java.

Java tiež umožňuje ľahko spúšťať externé programy a komunikovať s nimi prostredníctvom ich CLI. Nechýbajú v nej ani mechanizmy na komunikáciu cez sériový port.

## 4.3 Vizualizácia

Spúšťanie vytvorených programov v jednoduchej simulácii je jednou zo zamýšľaných funkcionalít našej aplikácie. Priamočiarym postupom implementácie je použitie API ako Vulkan alebo OpenGL pre komunikáciu s GPU a návrh grafických prvkov od základu. Takto by však bolo nutné vytvoriť pomerne rozsiahly modul na nízkej úrovni, čo nie je pre našu jednoduchú simuláciu nutné. Existuje hneď niekoľko riešení vyššej úrovne, ponúka sa použitie rôznych knižníc, simulátorov či herných programov.

### 4.3.1 Simulátor

Prirodzeným prístupom k riešeniu modulu simulácie je použitie existujúceho robotického simulátora. K dispozícii je ich hneď niekoľko, keďže sa bežne používajú ako pomôcka na testovacie účely pred skonštruovaním zariadení.

*Webots* je jedným z dostupných simulátorov. Ide o open–source desktopovú aplikáciu umožňujúcu navyše i modelovanie a programovanie prístrojov [37]. Program robota môže byť vytvorený v rôznych jazykoch (C, C++, Python , Java, MATLAB, ROS), prostredie simulácie možno upravovať v jazyku VRML97 (Virtual Reality Modeling Language). Súčasťou je podpora senzorov a simulácia fyzikálnych javov.

Ďalšími známymi simulátormi sú Gazebo, Visual Components, RoboDK, V-REP, RobotStudio, WorkcellSimulator či RoboLogix. Jedná sa však o programovacie nástroje, zväčša komplexné a hlavne samostatné aplikácie s vlastným GUI. Umožňujú detailnú simuláciu a pre účely jednoduchej vizualizácie nie sú vhodné. Od ich použitia nás odrádza nie len zbytočne vysoká komplexita ale i komplikácie spojené s integráciou do našej aplikácie.

### 4.3.2 Herný program

Herné programy (angl. game engines) poskytujú iný prístup k riešeniu modulu simulácie. Zvyčajne sú v nich k dispozícii všetky potrebné funkcionality bežne prítomné v už spomínaných simulátoroch (niektoré simulátory sú dokonca založené na hernom programe). Rovnako poskytujú nástroje pre konštrukciu a vykreslenie trojrozmerného virtuálneho prostredia, do ktorého možno pridávať ďalšie objekty. V celom prostredí možno uplatňovať a simulovať fyzikálne zákony a javy ako gravitácia, trenie či zrýchlenie. „Herný“ charakter týchto produktov je zvyčajne viditeľný najmä v poskytovaných funkcionalitách ako podpora sieťovej komunikácie či rozoznávanie gest v GUI.

Najväčšou výhodou je pomerne jednoduchá integrácia do aplikácie, keďže herné programy majú tiež formu softvérových rámcov a knižníc. Po stanovení implementačného jazyka je vhodné nájsť „hernú knižnicu“ vytvorenú v rovnakom jazyku. Inšpiráciou sú nám odporúčania uvedené na stránke LWJGL (Lightweight Java Game Library), knižnice, ktorá umožňuje v Java aplikáciách prístup k spomínaným API ako OpenGL či Vulkan [27]. Tu autori pre začínajúcich vývojárov v oblasti sami odporúčajú použitie herných programov založených na tejto knižnici. Spomínané sú dva — LibGDX a jMonkeyEngine.

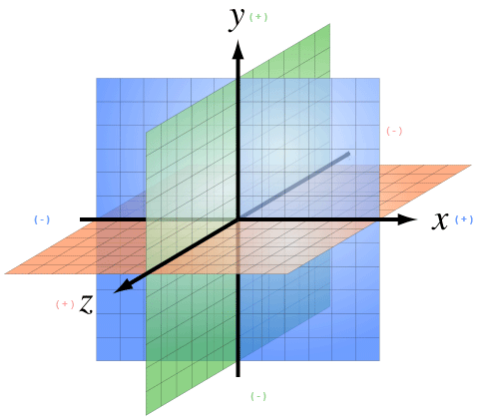
#### LibGDX

LibGDX je softvérovým rámcom (angl. framework) umožňujúcim vývoj hier pre viacero platforiem súčasne. Založený je na OpenGL, kód je zverejnený v duchu open–source a licencia Apache 2.0 nás od jeho použitia rovnako neodrádza. Pre používateľov — začiatočníkov je k dispozícii rozsiahla dokumentácia (Javadoc) a rôzne návody. Zaujímavosťou je aplikácia na vytvorenie základnej kostry výslednej „hry“ (u nás simulácie), ktorá po zadaní niekoľkých parametrov výrazne uľahčí úvodné nastavenie a možno tak rýchlo prejsť k vývoju hry samotnej (ako herná logika či rozmiestnenie objektov) [26].

#### jMonkeyEngine

jMonkeyEngine (ďalej ako „jME“) je možné k projektu ľahko pridať vo forme knižníc pomocou nástroja Maven. Podobne ako LibGDX nás v jeho použití neobmedzuje ani licencia (pripojená BSD 3–Clause licencia umožňuje použitie produktu s modifikáciou i na komerčné účely) ani poskytnuté funkcionality. Nechýba dokumentácia, návody pre začiatočníkov, vlastný SDK a IDE.

Oba herné programy pokrývajú naše požiadavky a na internetových fórach je ich možné veľakrát nájsť ako ekvivalenty, pre ktoré sa pri vývoji hry ťažko rozhodnúť. Častým odporúčaním však je voľba jME, ak má byť výsledný produkt zameraný prevažne na trojrozmernú grafiku, čo platí aj v našom prípade.




Obr. 4.2: Súradnicový systém scény v jMonkeyEngine

K výberu jME, ako technológie pre implementáciu simulácie, prispel i článok priamo opisujúci simulátor robotov vytvorený pomocou jME — jmeSim [43]. I keď je údajne tento simulátor open–source, nedopátrali sme sa k jeho zdrojovému kódu. Napriek tomu považujeme článok za dôkaz, že má zmysel snažiť sa o simuláciu v hernom programe, a jME je pre tento účel použiteľnou technológiou.

Integrovať simuláciu do našej aplikácie pomôžu existujúce riešenia ako JME3-JFX [21], ktoré umožňujú vykresľovanie grafického obsahu simulácie v jME priamo do komponentu JavaFX scény.

### Princíp a možnosti jMonkeyEngine

Ústredným určením jME je v podstate sprostredkovanie nízkoúrovňového prístupu k funkciám grafickej karty na úrovni vyššej. Programátorovi navyše poskytuje niekoľko užitočných abstrakcií a dátových štruktúr [20]. Základom hry (simulácie) je *scéna*, trojrozmerné prostredie, v ktorom sú umiestňované *uzly* usporiadané v stromovej štruktúre. Prázdnu scénu tvorí jediný uzol — *root*, ku ktorému sú pri výstavbe scény priraďované ďalšie. Uzol je abstraktným, neviditeľným bodom v priestore, má svoju polohu, rotáciu a mierku (angl. scale). Umožňuje logické rozmiestnenie viditeľných prvkov zlučovaním do skupín (viacero uzlov môže mať spoločného rodiča) a tým i hromadnú manipuláciu s mu podriadenými uzlami. Poloha je určená súradnicovým systémom (obrázok 4.2).

Viditeľnými časťami scény sú *geometrie*. Súčasťou scény sa stávajú priradením do už spomínaného stromu uzlov, a podobne ako uzlom je im možno určiť transláciu, rotáciu a mierku. Viditeľnú časť geometrie tvorí 3D tvar a materiál. Tvar môže byť jednoduchým modelom geometrického telesa (kocky, gule alebo iného) alebo komplexným modelom — *sieťou* (angl. mesh) tvorenou drobnými polygónmi (zvyčajne trojuholníkmi) aproximujúcimi komplikovanejší tvar povrchu zobrazovaného objektu. Materiál určuje vlastnosti povrchu modelu ako svetlosť, priehľadnosť či lesklosť, a pomocou neho môže byť na model aplikovaná textúra. S materiálmi úzko súvisí svetlo. V scéne možno

definovať zdroje svetla (umiestnenie, prípadne smer žiarenia), podľa ktorých pôsobenia sú následne zobrazované viditeľné prvky scény.

Dôležitou súčasťou scény je *kamera*. Tá určuje, čo sa zobrazí na výstupe, v priestore má definovanú polohu, orientáciu a dohľad. Výhodou definície dohľadu sú hlavne úspory výpočtových zdrojov, nakoľko pre prvky mimo záber nie je nutné počítať detaily zobrazenia.

Pre manipuláciu s uzlami a geometriami sú v jME k dispozícií rôzne dátové štruktúry. Používajú sa najmä vektor a kvaternión (angl. quaternion), pomocou ktorých možno meniť polohu a rotáciu objektov. K dispozícii sú tiež funkcie umožňujúce konštrukciu zložitejších štruktúr (kvaternión) z jednoduchších reprezentácií (matica alebo množina uhlov pri definícii rotácie). Dostupné sú i operácie nad týmto štruktúrami.

Kód simulácie sa skladá z dvoch hlavných častí, metód pre *inicializáciu* a *aktualizáciu*. Metóda inicializácie je volaná práve raz, pri spustení aplikácie. Jej určením je príprava scény, vykonávajú sa tu úkony ako počiatočné rozloženie uzlov, načítanie modelov, nastavenie osvetlenia. Metóda aktualizácie je zodpovedná za manipuláciu s prvkami scény počas behu aplikácie. Je cyklicky volaná obslužným vláknom aplikácie s parametrom určujúcim čas od posledného volania (angl. time per frame — TPF). Údaj je nápomocný napríklad pre zabezpečenie plynulého pohybu modelov v scéne, nakoľko jednotlivé volania metódy nemusia prebiehať v uniformných intervaloch. Po každom volaní metódy aktualizácie nasleduje vykresľovanie scény podľa aktuálneho rozloženia.

Simulácia fyzikálnych procesov je v jME možná pre spojenie s knižnicou Bullet, ktorá zodpovedá za monitorovanie stavu a prepočet polohy fyzikálnych objektov. V programe sú geometriám, ktoré majú byť súčasťou fyzikálnej simulácie vytvorené *fyzikálne ovládače* a následne je riadenie ich pohybu podriadené výsledkom výpočtov fyzikálneho modulu. Fyzikálnym objektom môžu byť priradené doplňujúce parametre ako hmotnosť, trenie povrchu, aktuálne pôsobiaca sila, rýchlosť a podobne. Dôležitým konceptom pri simulácii fyzikálnych procesov (najmä v reálnom čase) je zjednodušovanie modelu simulovaného objektu. Nemusí pritom ísť o zjednodušenie viditeľnej časti, fyzikálny modul môže pre účely výpočtu používať iný, jednoduchší tvar telesa ako je zobrazovaný používateľovi, takzvaný *kolízny tvar*. Bullet zohľadňuje zotrvačnosť telies a umožňuje definovať vzťah medzi dvoma uzlami pomocou neviditeľného „kĺbu“, obmedzujúceho rotáciu v definovaných osiach a rozmedzí pri pôsobení síl na ním spojené uzly.

# Kapitola 5

# Návrh a implementácia

V kapitole opisujeme priebeh vzniku aplikácie, implementačné detaily, postupy a rozhodnutia. Základné členenie obsahu je podľa modulov, za ktoré považujeme celky aplikácie ako editor kódu vo vizuálnom jazyku, modul pre sériovú komunikáciu, modul umožňujúci kompiláciu vytvoreného programu či modul pre simuláciu.

Vývoj začíname návrhom systému, ktorý umožní v aplikácii sprístupniť rôzne programovacie verzie pre rôzne modely robota a rôzne typy riadiaceho programu. Pokračujeme návrhom kostry a vizuálnej podoby aplikácie, v ktorej neskôr vzniknú jednotlivé moduly. V prvej fáze implementujeme časti podporujúce komunikáciu s aktuálnou verziou riadiaceho programu robota Otto a jeho programovanie. Nasleduje etapa rozširovania funkcionalít (hlavne vizuálneho jazyka) a práca na module simulácie.

## 5.1 Systém verzií

Systém umožňuje definovať správanie jednotlivých modulov aplikácie (editor kódu, modul sériovej komunikácie, modul pre kompiláciu a nahranie riadiaceho programu, modul simulácie) pomocou konfiguračných súborov. Interakcia s aplikáciou začína vyhľadaním, výberom a načítaním takéhoto súboru — verzie, ktorá určuje pre jednotlivé moduly ich počiatočnú konfiguráciu (nastavenie po načítaní konfiguračného súboru verzie, pred interakciou používateľa s aplikáciou) a ich správanie počas interakcie s užívateľom (kým nie je načítaná iná verzia alebo ukončená aplikácia). Úlohou verzií je zabezpečiť nenáročnú rozšíriteľnosť a konfigurovateľnosť aplikácie parametrizáciou modulov. Počas tvorby aplikácie vznikli dve verzie, „Otto 2020 Robotická liga“ a „Otto 2021 Procedural“.

### 5.1.1 Konfiguračný súbor

Súbor reprezentujúci verziu je tvorený množinou dvojíc kľúč–hodnota. Zoznam použitých kľúčov a význam priradenej hodnoty definuje nasledovná tabuľka.

| **Všeobecné informácie o verzii** | |
|---|---|
| *kľúč* | *hodnota* |
| name | textový identifikátor verzie pre používateľa |
| robotMaxMemory | číslo, limit pre súčet veľkostí vybraných logických modulov |
| moduleCount | počet logických modulov |
| exampleCount | počet ukážkových programov |

**Informácie pre modul editor kódu VPJ**

| *kľúč* | *hodnota* |
|---|---|
| toolbox | XML definícia obsahu ponuky prvkov VPJ |
| workspace | XML definícia iniciálneho obsahu pracovnej plochy editora VPJ |
| categories | zoznam názvov kategórií v ponuke prvkov VPJ |
| generator | označenie generátora kódu („prekladača“ VPJ) |

**Informácie pre modul kompilácie a modul sériovej komunikácie**

| *kľúč* | *hodnota* |
|---|---|
| programInRam | pravdivostná hodnota, určenie, či má byť pre použitie verzie nahraný do mikropočítača statický riadiaci program |
| programLocation | (voliteľné) relatívna cesta k statickému riadiacemu programu |
| codeToConsole | pravdivostná hodnota, určenie, či preložený program z VPJ možno odostať priamo cez sériovú komunikáciu, bez kompilácie |

**Informácie pre modul načítania @ choreografie robota Otto**

| *kľúč* | *hodnota* |
|---|---|
| codeLoader | označenie parsera pre vytvorenie prvkov VPJ z *@ choreografie* |

**Zoznam logických modulov, pre modul X (kde X je prirodzené číslo)**

| *kľúč* | *hodnota* |
|---|---|
| moduleX.name | názov, textový identifikátor pre používateľa |
| moduleX.description | opis, textový identifikátor pre používateľa |
| moduleX.required | pravdivostná hodnota, určenie, či je logický modul povinným (ak áno, vždy je súčasťou výberu) |
| moduleX.size | veľkosť podporných súborov modulu |
| moduleX.categories | zoznam názvov súvisiacich kategórií v ponuke prvkov VPJ modulu editor kódu, podmnožina zoznamu pod kľúčom *categories* |
| moduleX.header | (voliteľné) označenie podporného súboru, časť *header* |
| moduleX.setup | (voliteľné) označenie podporného súboru, časť *setup* |
| moduleX.footer | (voliteľné) označenie podporného súboru, časť *footer* |

**Zoznam ukážkových programov, pre ukážku X (kde X je prirodzené číslo)**

| *kľúč* | *hodnota* |
|---|---|
| exampleX.name | názov, textový identifikátor pre používateľa |
| exampleX.description | opis, textový identifikátor pre používateľa |
| exampleX.modules | zoznam poradových čísel použitých logických modulov |
| exampleX.workspace | XML definícia (iniciálneho) obsahu pracovnej plochy editora VPJ |

### 5.1.2 Princíp

Systém verzií sprístupní po spustení aplikácie používateľovi možnosť vyhľadať dostupné verzie. Po tomto procese môže byť verzia zvolená výberom zo zoznamu názvov verzií (hodnota priradená kľúču *name*). Následne je podľa hodnoty *programInRam* určené, či verzii prislúcha „statický“ riadiaci program mikropočítača.

Ak verzii neprislúcha statický riadiaci program, nasleduje po voľbe verzie výber *logických modulov*. Jedná sa o (tematické) celky funkcionalít, určujú, ktoré prvky budú v editore VPJ k dispozícii a ktoré podporné súbory modulov (časti *header, setup a footer*) riadiaceho programu budú kompilované. Používateľovi je pre tento účel zobrazené dialógové okno, pre každý modul zobrazujúce názov, opis a veľkosť. Na základe informácie *robotMaxMemory* je počet vybraných modulov limitovaný tak, aby súčet veľkostí vybraných modulov túto hodnotu nepresiahol. Logické moduly tiež umožňujú zjednodušiť programovanie začiatočníkom obmedzením množstva dostupných funkcií.

Ak verzii prislúcha statický riadiaci program, je po jej zvolení zobrazená ponuka na jeho nahratie (modul kompilácie a nahratia kódu je spustený so vstupom definovaným parametrom *programLocation* a následne je znefunkčnený, nakoľko kód vytváraný vo VPJ tejto verzie nie kompilovateľný C++ kód). V tomto prípade nie je používateľovi umožnená voľba logických modulov, automaticky sú zvolené všetky dostupné.

Po voľbe logických modulov alebo nahraní statického riadiaceho programu nasleduje nastavenie ostatných modulov aplikácie. Pre modul editora VPJ je vo verzii uvedené, ktoré prvky VPJ majú byť k dispozícii. Obsah ponuky prvkov VPJ (obr.5.1, priestor A) je inicializovaný načítaním XML definície pod kľúčom *toolbox*. Na základe voľby logických modulov sú potom zobrazené len kategórie (komponenty XML), ktorých názvy sú súčasťou zoznamu názvov kategórií niektorého z vybraných logických modulov (kľúč *moduleX.categories*). Ostatné kategórie zo zoznamu (zoznam pod kľúčom *categories*) sú skryté. Pre modul editora VPJ je vo verzii tiež definované počiatočné rozmiestnenie prvkov VPJ v editore (XML reprezentácia obsahu pod kľúčom *workspace*), ktoré používateľ následne upravuje a určenie generátora (pod kľúčom *generator*) (kapitola 5.3), ktorý sa použije pri preklade VPJ.

V module sériovej komunikácie je na základe hodnoty *codeToConsole* umožnené alebo znefunkčnené priame odoslanie generovaného kódu cez pripojený sériový port.

Modul kompilácie je po voľbe logických modulov konfigurovaný na základe hodnôt *moduleX.header*, *moduleX.setup* a *moduleX.footer* zvolených modulov. K logickému modulu môžu prislúchať súbory definujúce podporne funkcie (v jazyku C++), ktorých volania sú súčasťou prekladu prvkov VPJ sprístupnených daným logickým modulom. Na základe voľby logických modulov je vytvorený zoznam takýchto súborov, ktoré pri kompilácií spracuje modul kompilácie (kapitola 5.5).

Súčasťou definície verzie je tiež informácia, či je možné vygenerovať obsah editora

VPJ z užívateľom vloženého kódu vo formáte trojíc celých čísel, reprezentujúcich choreografiu robota Otto (kľúč *codeLoader*).

K verzii môžu byť tiež priložené definície ukážkových programov. Definujú prednastavené rozloženie prvkov v editore VPJ, ktoré možno načítať a ďalej upravovať. Nevyhnutné je pre každý ukážkový program definovať zoznam použitých modulov, aby bolo možné nastaviť kompilátor a kategórie prvkov VPJ zobrazené v ponuke editora.

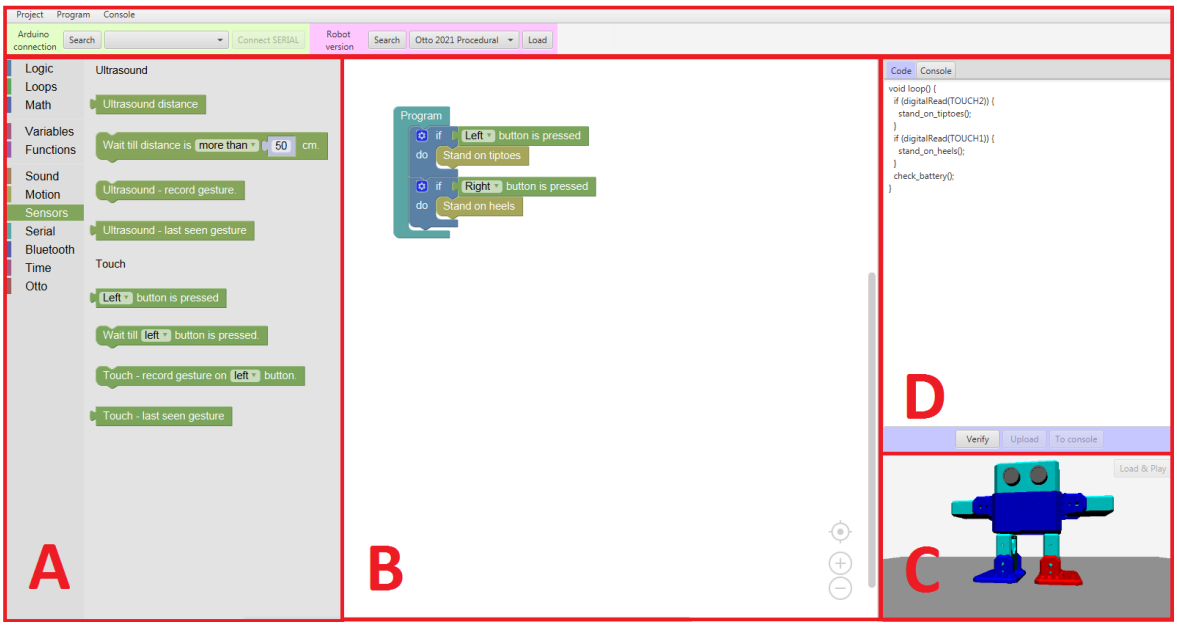
### 5.1.3 Verzia Otto 2020 Robotická liga

Verzia predstavuje konfiguráciu aplikácie, v ktorej je možné tvoriť kód pre robota Otto ako trojice celých čísel (spôsob vyvinutý pre účely denných táborov, kapitola 2.1.4). Tento prístup vyžaduje, aby bol v Arduino nahraný „statický“ riadiaci program, ktorý následne interpretuje znaky prijaté cez sériovú komunikáciu. Vo verzii je teda definované, že pre jej použitie je nutné do robota nahrať takýto program a používateľ je po jej voľbe vyzvaný, aby tak učinil. Modul kompilácie je následne (na základe informácie vo verzii) znefunkčnený, nakoľko ďalšie programovanie robota prebieha priamym odosielaním znakov cez sériovú linku (kapitola 2.1.4).

Vo verzii *Otto 2020 Robotická liga* sú v editore VPJ k dispozícii len prvky prezentovateľné trojicami celých čísel, ktorými je tvorená choreografia a prvky na obsluhu senzorov sú skryté, nakoľko tie riadiaci program nepodporuje. Generátor pre verziu *Otto 2020 Robotická liga* prekladá prvky VPJ na trojice celých čísel.

### 5.1.4 Verzia Otto 2021 Procedural

Verzia *Otto 2021 Procedural* predstavuje konfiguráciu aplikácie, v ktorej je možné tvoriť kód pre robota Otto prvkami VPJ reprezentujúcimi prvky jazyka Arduino (C++). Po načítaní tejto verzie je užívateľovi umožnená voľba *logických modulov*, ktoré sú vo verzii definované. Príkladom je logický modul *senzory*, ktorý po výbere používateľovi sprístupní vo VPJ bloky umožňujúce čítať aktuálne merané hodnoty alebo čakať na nameranie konkrétnej hodnoty. Možnosť odoslať kód vytvorený prekladom VPJ prostredníctvom sériovej komunikácie je znefunkčnená, namiesto toho je k dispozícii možnosť kód kompilovať a vzniknutý riadiaci program odoslať do riadiacej jednotky robota použitím modulu kompilácie (kapitola 5.5).




Obr. 5.1: Rozloženie používateľského rozhrania

## 5.2 Grafické rozhranie

GUI aplikácie tvoríme pomocou JavaFX. Cieľom je vytvoriť prehľadné prostredie, v ktorom sa ľahko zorientuje i požívateľ nižšieho veku. Pri rozmiestňovaní jednotlivých prvkov sú nám nápomocné existujúce aplikácie, no zohľadňujeme i rady autorov knižnice Blockly [10], keďže komponent umožňujúci prostredníctvom nej programovať robota je v našej aplikácii dominantným. Z odporúčaní je zrejmé, že plochu pre manipuláciu s prvkami vizuálneho jazyka je žiaduce maximalizovať a neoddeľovať ju od komponentu umožňujúceho tvorbu blokov (prvkov jazyka).

Ďalšie prvky vyžadujúce v aplikácii väčšiu plochu sú najmä výstupné, predovšetkým ide o výstup pre modul simulácie a výstup „prekladača“ vizuálneho jazyka na kód následne kompilovaný a (alebo) odosielaný do riadiacej jednotky robota. Taktiež je nutné zakomponovať do GUI modul pre komunikáciu s robotom, konzolu sériovej komunikácie.

Výsledné rozloženie môžete vidieť na obrázku 5.1. Tvoriť kód vo vizuálnom jazyku možno v časti *A*, umožňujúcej výber blokov, a ich následným umiestňovaním v priestore *B*. Sektor *C* je vyhradený pre výstup modulu simulácie. Časť *D* je zdieľaná modulom sériovej komunikácie a modulom umožňujúcim náhľad do kódu generovaného prekladom VPJ, medzi ktorými možno prepínať. Nad spomínanými oblasťami sa nachádza lišta nástrojov, sprístupňujúcich funkcie ako vyhľadanie a pripojenie sériovej komunikácie či voľbu verzie programovaného riadiaceho programu robota.

## 5.3 Editor kódu

Modul umožňujúci tvorbu (riadiaceho) programu pomocou vizuálneho programovacieho jazyka (VPJ) je v podstate samostatnou webovou JavaScript aplikáciou. V GUI našej aplikácie je zobrazený jej výstup tvorený knižnicou Blockly v komponente WebView (webový prehliadač). Interakcia medzi vonkajšou Java aplikáciou a vnorenou JavaScript aplikáciou prebieha v Java pomocou inštancie WebEngine. Editor sa navonok skladá z dvoch hlavných častí — ponuky prvkov VPJ (obr.5.1, priestor A), z ktorej používateľ vyberá požadované „diely“ VPJ a pracovnej plochy (obr.5.1, priestor B), v ktorej sú vybrané komponenty VPJ používateľom umiestňované, je nimi tvorený kód.

Konfiguráciu modulu editora kódu zabezpečuje systém verzií (kapitola 5.1), ktorý využíva rozhranie poskytnuté knižnicou Blocky pre úpravu momentálne zobrazeného obsahu pracovnej plochy a ponuky prvkov VPJ. Obsah a rozloženie oboch týchto častí možno určiť načítaním ich definície vo formáte XML (súčasť verzií, kapitola 5.1.1).

Rozhranie definuje i funkcie, ktorými je možné volať generátory VPJ. Ich úlohou je preklad prvkov VPJ na kód riadiaceho programu (prípadne jeho časti), ktorý je následne spracovávaný Java aplikáciou a kompilátorom. Usporiadaním v editore kódu sú použité bloky VPJ organizované v grafovej štruktúre, generátor je funkcia, ktorej parametrom je blok. Má prístup k poliam bloku (polia sú prvky pre výber možností, prípadne textový alebo číselný vstup. Zároveň má generátor prístup k vnoreným blokom (napríklad blok cyklu *while* má prístup k blokom v tele cyklu) a pripojeným „nasledovníkom“ — bloku pod ním. Návratovou hodnotou je vhodne spracovaný kód so zakomponovaným obsahom potomkov a polí. Kód je generovaný rekurzívne, zvyčajne začínajúc na najvyššom bloku umiestnenom v editore.

Rozhranie tiež umožňuje získať aktuálny obsah pracovnej plochy zapísaný v XML reprezentácii a naopak, obnoviť obsah pracovnej plochy z poskytnutej XML reprezentácie. Tento mechanizmus využívame pre načítanie iniciálneho stavu pracovnej plochy (po voľbe a načítaní verzie) ale i načítanie ukážkových programov. Samozrejmosťou je implementácia funkcií umožňujúcich uloženie a znovu načítanie celého projektu.

Detailom implementácie prvkov a logiky vzniknutého vizuálneho programovacieho jazyka je venovaná kapitola 6.

## 5.4 Komunikácia s robotom

Možnosť komunikácie s robotom je jednou zo základných požiadaviek na našu aplikáciu. Umožňuje vyššiu interakciu s používateľom a je tiež nápomocná pri ladení programov (pomocné výpisy). Komunikácia s robotom na úrovni nahrania kompilovaného programu je predmetom časti 5.5, tu je opísaná sériová komunikácia už bežiaceho programu v mikropočítači Arduino s našou aplikáciou. Jedná sa o komplementárnu časť systému k časti implementovanej v mikropočítači (sekcia 2.1.1).

V aplikácii je cieľom vytvoriť integrovaný terminál pre sériovú komunikáciu. Mikropočítač Arduino komunikuje po pripojení rozhraním USB a nainštalovaní príslušného ovládača v počítači s OS, ktorý nám toto spojenie sprostredkuje. Podobne pracuje i terminál v Arduino IDE (oficiálne IDE pre vývoj riadiacich programov Arduino) alebo program Putty (klient pre ssh, Telnet, Rlogin a sériovú komunikáciu [32]). Autori mikropočítača odporúčajú pre implementáciu sériovej komunikácie v Java použitie knižníc [4]. V článku o prepojení Java — Arduino sú spomínané dve, údajne nespoľahlivá knižnica RXTX a možná alternatíva, knižnica jSerialComm, ktorú použijeme.

jSerialComm je platformovo nezávislá knižnica, ľahko integrovateľná do našej aplikácie [22]. Umožňuje vyhľadávať pripojené dostupné sériové porty, ako aj obojsmernú komunikáciu po pripojení. Podporuje niekoľko typov operácie, v závislosti na blokovaní čítania z (zápisu do) sériového portu. V prípade čítania sú na výber alternatívy ako *neblokovaná komunikácia*, v ktorej možno príkazom *read()* skúsiť načítať dáta, ak však nie sú k dispozícii (neboli prijaté), vrátený je prázdny reťazec. Inou možnosťou je vynútenie čakania na možný príchod správy a to buď len po nejaký čas alebo až do jej prijatia. Vo oboch týchto prípadoch by ale v našej aplikácii bolo nutné cyklicky kontrolovať, či nejaké dáta nie sú k dispozícii (ako v mikropočítači), knižnica ale poskytuje vhodnejší prístup, registráciu *callback* funkcie. Tú knižnica zavolá po každom výskyte špecifikovanej udalosti, ktorou môže byť dostupnosť dát alebo prijatie ucelenej správy. V implementácii volíme možnosť volania *callback* funkcie po prijatí ucelenej správy, pričom za koniec správy je považovaný znak konca riadku (\n — line feed). Registrovaná *callback* funkcia správu načíta a zobrazí v termináli používateľovi.

Odosielanie správ je rovnako možné nastaviť do blokujúceho režimu, kde pri pokuse o zápis do sériového portu knižnica čaká buď to určitý čas alebo kým sa podarí odoslať požadovaný počet bajtov. Pre implementáciu volíme neblokovaný prístup, ak používateľ odošle dáta, sú zapísané hneď ako je to možné. Neblokovaný prístup umožní odoslanie viacerých správ, ktoré následne v rovnakom poradí knižnica odvysiela. Implementujeme tiež možnosť odosielať správy pozostávajúce z jedného znaku ihneď po stlačení požadovaného klávesu, bez nutnosti dodatočného potvrdenia odoslania. Táto funkcionalita je vhodná napríklad pri interakcii s robotom Otto v režime priameho riadenia (kapitola 2.1.4).

## 5.5 Kompilácia a nahranie riadiaceho programu

Kompiláciu a nahranie riadiaceho programu pri vývoji programov pre Arduino zvyčajne zabezpečuje Arduino IDE s vlastným grafickým používateľským rozhraním (sekcia 2.1.1). Nahrávanie kompilovaného programu prebieha sériovou komunikáciou medzi Arduino IDE a *bootloader* programom v mikropočítači, zodpovednom za uloženie prijatého kódu do flash pamäte [31]. Overenie prenosu je uskutočnené spätným odoslaním celého programu z mikropočítača do IDE. *Bootloader* je spustený len určitý (krátky) čas po resetovaní procesora, ak v tomto časovom okne nie je zaznamenané nahrávanie riadiaceho programu, spustí sa posledný nahraný program. Procesor je resetovaný po obnove napájania ale i pri nadviazaní nového sériového spojenia.

Proces kompilácie a nahrania riadiaceho programu je možné implementovať na nižšej úrovni, samostatným volaním kompilátora a následným použitím knižnice jSerialComm pre nahranie a spätnú verifikáciu. Pre kompiláciu je v tomto prípade možné použiť napríklad *Arduino builder* [5]. Nástroj už ale nie je vývojármi udržiavaný, odporúčanou alternatívou je *Arduino CLI*, poskytujúci robustnejšie riešenie. Umožňuje kompiláciu a zároveň i nahratie riadiaceho programu do mikropočítača. Problémom tohto riešenia je naopak aktívny vývoj, autori upozorňujú na možné zásadné zmeny do vydania verzie 1.0.0 [6].

Implementovaným riešením je použitie CLI samotného Arduino IDE, ktoré poskytuje všetky nami požadované funkcionality [13]. Vstupom pre Arduino IDE je jediný súbor obsahujúci C++ kód riadiaceho programu, ktorý je pred kompiláciou nutné vyskladať.

Ku každému použitému logickému modulu aktuálne načítanej verzie (kapitola 5.1.1) môže byť v konfiguračnom súbore verzie definovaný samostatný podporný súbor reprezentujúci časť *header*, *setup* a *footer*. Tieto súbory obsahujú definície konštánt, premenných a funkcií potrebných pre fungovanie daného logického modulu, nakoľko kód generovaný prvkami VPJ sprístupnenými týmto logickým modulom obsahuje ich volania. Súbor *header* vo všeobecnosti z pravidla obsahuje definície premenných, konštánt a makier, súbor *setup* obsahuje príkazy potrebné na inicializáciu modulu (definovaných premenných) a v súbore *footer* sú uvedené definície podporných funkcií.

Riadiaci program v C++ je vyskladaný postupným zreťazením *header* súborov použitých logických modulov, následným pripojením definície časti *setup* (vznikne zreťazením *setup* súborov použitých logických modulov) , pripojením časti loop (tú vytvorí generátor VPJ) a pripojením časti definícií podporných funkcií (vznikne zreťazením *footer* súborov použitých logických modulov). Výsledný súbor je vstupom pre Arduino IDE, ktoré je spustené ako samostatný proces na pozadí. Prostredníctvom CLI sú Arduino IDE odovzdané okrem vyskladaného riadiaceho programu (cesty k nemu) parametre sériového portu, cez ktorý prebehne nahranie. Požadovaný sériový port zvolí

používateľ v GUI našej aplikácie pomocou knižnice jSerialComm. Textový výstup procesu bežiaceho na pozadí je presmerovaný do grafického prvku aplikácie, kde je možné sledovať priebeh, výsledky a prípadné chyby. Arduino IDE je tiež možné použiť bez pripojenia sériového portu na overenie vytvoreného riadiaceho programu.

## 5.6 Simulácia

Implementáciu modulu simulácie začíname integráciou grafického prvku zobrazujúceho výstup simulácie v používateľskom rozhraní našej aplikácie. Herný charakter použitej technológie jMonkeyEngine (skr. jME, kapitola 4.3.2) umožňuje jednoduché vytvorenie aplikácie — hry, ktorej jediné grafické okno pozostáva z výstupu simulácie. V našom prípade je ale simulácia len „doplnkom“, rozhodli sme sa ju preto integrovať do hlavného okna našej aplikácie vo vymedzenom priestore, ako vidno na obrázku 5.1. K tomuto účelu je nám nápomocné riešenie *JME3–JFX* [21], ktoré umožňuje vykresľovanie grafického výstupu simulácie jME priamo do komponentu *ImageView* JavaFX scény. Samotná integrácia je potom už len jednoduchým volaním metódy prepájajúcej inštanciu jME s inštanciou ImageView.

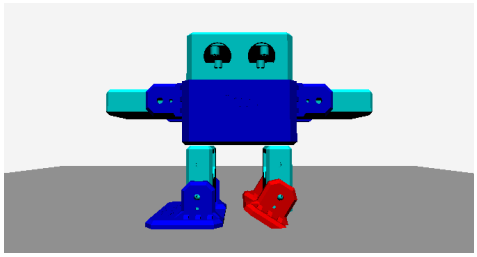
V ďalšom je potrebné vyskladať model robota Otto a samotnú scénu — prostredie, v ktorom sa bude simulovaný model pohybovať. Pre účely jednoduchej simulácie scénu v základe tvorí rovná plocha. V jazyku jME to znamená vytvorenie jedného širokého hlbokého nízkeho kvádra a jeho umiestnenie do stredu scény. Hornú stenu vzniknutej „podlahy“ sivej farby možno vidieť na obrázkoch 5.2 a 5.3 pod modelom robota.

Súčasťou scény je kamera určujúca zobrazenie pre používateľa. V GUI implementujeme ovládacie prvky, ktorými možno meniť jej polohu a rotáciu. Používateľ tak získa možnosť pohľadu na scénu z ľubovoľného uhľa a vzdialenosti.

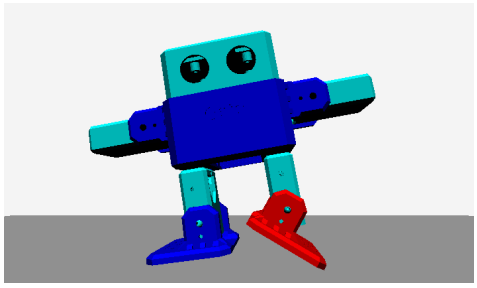
### Model robota Otto

Cieľom je vytvoriť model schopný pohybu vo vytvorenom prostredí. Rozhranie pre riadenie pohybu vytvárame analogicky rozhraniu v riadiacom programe pre Arduino, kde je pohyb motora evokovaný volaním funkcie s jediným parametrom, definujúcim požadovanú polohu v stupňoch. Pri vytváraní modelu sú nám nápomocné existujúce modely častí robota určené pre 3D tlač, z ktorých preberáme definície povrchových sietí. Model robota je následne vytvorený vhodným rozmiestnením sietí v scéne (ich zaradením do stromovej hierarchie uzlov). K zobrazeniu je nutné definovať použitie materiálov (kapitola 4.3.2).

V základnej verzii je pohyb interpretovaný ako priama zmena rotácie konkrétneho uzla (geometrie) v scéne. Pre plynulý pohyb je implementovaná časť vykonávajúca interpoláciu medzi východiskovou a cieľovou polohou končatiny. Vzniknutý model je



Obr. 5.2: Robot Otto — simulácia bez detekcie kolízií



Obr. 5.3: Robot Otto — simulácia s detekciou kolízií

statický, pohyb modelu špičky nohy robota smerom k zemi preniká modelom podlahy a neovplyvňuje pohyb ostatných častí, tento stav možno vidieť na obrázku 5.2.

Aby bolo možné simulovať fyzikálne zákony, definujeme podlahe a jednotlivým častiam robota v jME fyzikálne ovládače. Pre vysokú detailnosť povrchových sietí sú pre účely simulácie fyziky definované zjednodušené kolízne tvary, dizajn robota umožňuje použitie kvádrov. Je tiež nutné zmeniť prístup k vykonávaniu pohybu, nakoľko manuálne nastavenie rotácie nie je počas svojho priebehu brané fyzikálnou simuláciou do úvahy. Možno tak vytvoriť fyzikálne nemožný stav (obrázok 5.2), ktorý simuláciu znefunkční. Riešením pre simuláciu motorov je použitie kĺbov. Pomocou nich definujeme vzťahy medzi uzlami, osi rotácie a možnú mieru pohybu. Pohyb je následne vykonávaný periodickým nastavovaním uhlu zvieraného kĺbom, interpoláciou medzi východiskovou a cieľovou polohou končatiny. Výsledný efekt možno vidieť na obrázku 5.3.

# Kapitola 6

# Návrh vizuálneho programovacieho jazyka

V kapitole opisujeme priebeh vzniku vizuálneho programovacieho jazyka použitého v našej aplikácii. Rôzne verzie (kapitola 5.1) vyžadujú sprístupnenie rôznych komponentov. Hlavným predmetom tvorby VPJ pomocou knižnice Blockly je definícia blokov a im prislúchajúcich generátorov (kapitola 4.1).

## 6.1 Verzia Otto 2020 Robotická liga

V tejto verzii je možné tvoriť choreografie pre robota Otto ako postupnosť trojíc celých čísel (kapitola 2.1.4 a 5.1.3). Formát a význam jednotlivých čísel je uvedený na web stránke Denného tábora digitálnych technológií prezentujúcej robota Otto [28]. Základom sú trojice pre ovládanie motorov, k dispozícii sú ale i ďalšie, umožňujúce ovládanie prehrávania melódie či zvukových efektov (tabuľka 6.1). Celá choreografia, sériovou linkou prenášaná do mikropočítača, má formát „@ $x_1$ $y_1$ $z_1$ $x_2$ $y_2$ $z_2$ ... $x_n$ $y_n$ $z_n$ 0 0 0“, kde vždy trojica (príkaz) pozostáva z hodnôt $(x_i, y_i, z_i)$.

| **formát príkazu** | | | **význam** |
|---|---|---|---|
| $X$ | $Y$ | $Z$ | počkaj $X$ ms, pohni motorom $Y$ do polohy $Z$; $Y \in [1;6]; Z \in [0,180]$ |
| 1 | 8 | $X$ | nastav celkový čas prehrávania choreografie na $X$ sekúnd |
| 1 | 9 | $X$ | pokračuj príkazom na riadku $X$ |
| 1 | 10 | $X$ | nastav spomalenie pohybu, 0 = bez spomalenia |
| 1 | 11 | $X$ | začni hrať melódiu číslo $X$, 0 = vypnúť zvuk |
| 1 | 12 | $X$ | zahraj zvukový efekt číslo $X$ |
| 1 | 13 | 0 | zastav prehrávanie melódie |
| 1 | 14 | $X$ | začni hrať pesničku číslo $X$ |
| 1 | 15 | 0 | vypni / zapni reproduktor |

Tabuľka 6.1: Prípustné hodnoty v zápise choreografie robota Otto pomocou trojíc čísel


Obr. 6.1: Ukážka kódu vo VPJ pre verziu Otto Robotická liga 2020

Na základe tabuľky 6.1 vytvárame vo VPJ bloky pre jednotlivé funkcie (trojice). Cieľom je zachovať rozsah funkcionalít no ponúknuť atraktívnejší a pohodlnejší prístup k tvorbe choreografie tohto formátu. Jednotlivé bloky sú interpretované jednoducho ako prislúchajúca trojica celých čísel.

Príklad choreografie vytvorenej pomocou vzniknutých blokov možno vidieť na obrázku 6.1. Bloky vyžadujúce zadanie parametra majú v tele dostupné pole, kde je možné príslušnú hodnotu určiť. Na obrázku 6.1 ide napríklad o bloky pre nastavenie spomalenia alebo blok pre spustenie prehrávania melódie. V prípade bloku pre riadenie pohybu motora zavádzame možnosť výberu konkrétneho motora z ponuky namiesto zadania jeho čísla. Zavádzame tiež blok „Program“, do ktorého tela sú vkladané ostatné bloky — prvky choreografie. Blok „Program“ je v tejto verzii statickým prvkom pracovnej plochy editora VPJ, používateľ ho nemôže odstrániť ani duplikovať. Slúži na ohraničenie blokov, ktoré tvoria choreografiu. Bloky umiestnené v editore mimo telo tohto bloku nie sú brané do úvahy pri generovaní kódu. Výsledný program po preložení blokov môžete vidieť v ukážke 6.1.

```
1 @1  10  3
2 1  11  1
3 1  12  1
4 1000  5  0
5 0  0  0
```

Ukážka kódu 6.1: Kód generovaný z blokov na obrázku 6.1

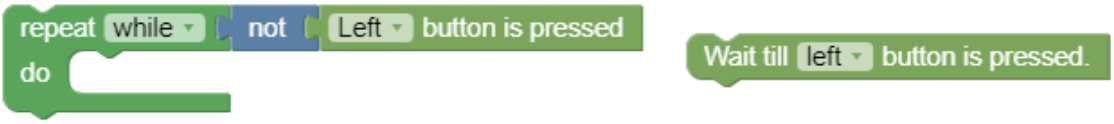
## 6.2 Verzia Otto 2021 Procedural

V tejto verzii sprístupňujeme vo VPJ prvky pre tvorbu riadiaceho programu v jazyku Arduino (C++). Použité bloky sú členené do kategórií. Každá kategória zodpovedá „tematickému“ celku, prípadne logickému modulu verzie. Podľa zavedených kategórií sú bloky zobrazené v ponuke v používateľskom rozhraní. Nižšie uvádzame ich zoznam.

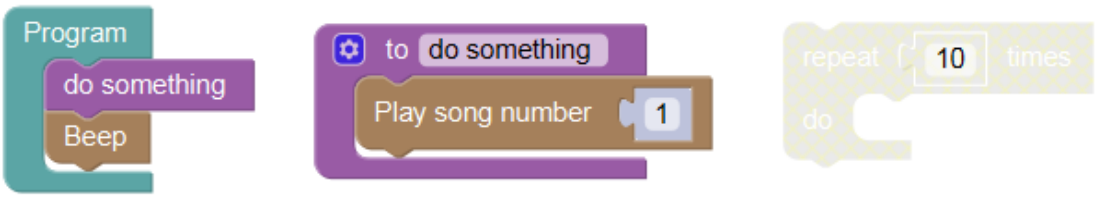
- Matematická logika (podmienka if, logické operácie, pravdivostné hodnoty)
- Cykly (cyklus for, while a blok pre príkazy *break* a *continue*)
- Matematika (číselné konštanty, aritmetické operácie, náhodné čísla)
- Premenné (tvorba premenných rôznych typov)
- Funkcie (tvorba procedúr a funkcií)
- Zvuk (ovládanie zvukových efektov a mp3 prehrávača)
- Pohyb (ovládanie servomotorov)
- Senzory (prístup k funkciám dotykových tlačidiel a ultrazvukovému senzoru)
- Sériová komunikácia prostredníctvom USB
- Sériová komunikácia prostredníctvom Bluetooth
- Práca s časom
- Kontrola batérií

Knižnica Blockly poskytuje základné, všeobecné bloky „od výroby“. K dispozícii sú bloky pre *vetvenie programu podmienkou if*, bloky pre *cykly*, bloky pre *logické a číselné konštanty* a manipuláciu s nimi (základná aritmetika a logické operácie), bloky pre *manipuláciu s textom*, *tvorbu zoznamov*, *reprezentáciu a miešanie farieb*, *tvorbu premenných* a *definíciu procedúr a funkcií*. Súčasťou sú generátory pre jazyky JavaScript, Python, Dart, Lua a PHP, z ktorých možno vychádzať. Niektoré z týchto blokov použijeme i v našej aplikácii, najmä definíciu ich vizuálnej podoby. Syntax jazyka C však vyžaduje redefiníciu generátorov viacerých prebraných blokov. K prebraným častiam patria bloky pre logické operácie, cykly, definície procedúr a funkcií, matematické operácie a premenné. Dôležitými pre našu aplikáciu sú však najmä novovzniknuté bloky, umožňujúce obsluhu špecifických funkcií robota. V knižnici dotvárame bloky umožňujúce pohyb, čítanie hodnôt meraných senzormi, sériovú komunikáciu či komunikáciu rozhraním Bluetooth.

Komplexitou funkcie poskytovanej blokom je možné nastaviť úroveň abstrakcie. Jednoduchým príkladom je blok „čakanie na stlačenie tlačidla“ (obrázok 6.2). Možno ho ľahko vyskladať z blokov *cyklu while*, *negácie* a bloku pre *načítanie stavu tlačidla* (na obrázku vľavo), no najmä pre nižšie vekové kategórie je vhodnejšia alternatíva




Obr. 6.2: Abstrakcia bloku „čakaj na stlačenie tlačidla“



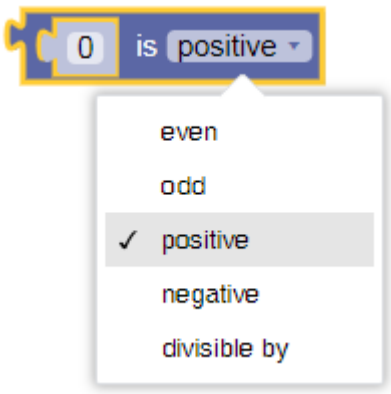

Obr. 6.3: Blok umiestnený mimo hlavného programu a definície procedúry

vyššej abstraktnej úrovne (na obrázku vpravo). Iným príkladom je blok pre načítanie gesta pomocou ultrazvukového senzora, ktorý reprezentuje v pozadí pomerne zložitú implementáciu časti riadiaceho programu, zabezpečujúcu komunikáciu s hardvérom a elimináciu chyby opakovaným meraním. V rámci jednotlivých kategórií je preto našou snahou poskytnúť bloky rôzneho charakteru, s rôznou úrovňou abstrakcie.

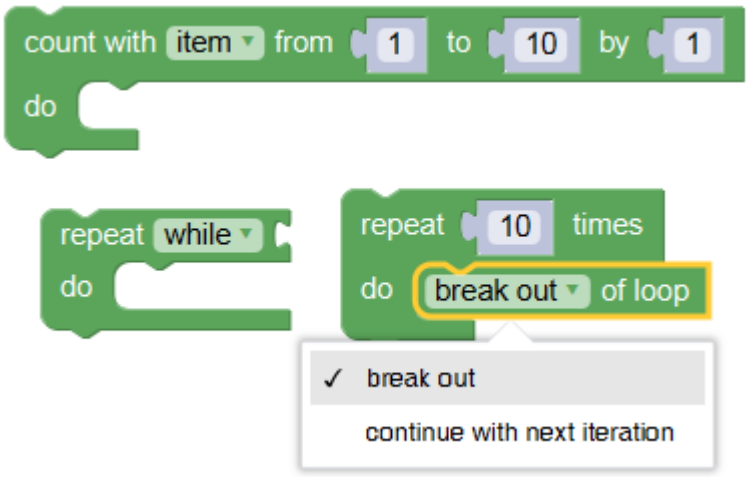
I v tejto verzii zavádzame v editore VPJ fixný, statický blok „Program“, ktorý reprezentuje časť *loop* riadiaceho programu a všetok kód je tvorený pridávaním blokov do jeho tela. Bloky bez pripojenia (priameho a zároveň nepriameho) k tomuto bloku nie sú pri vyhodnocovaní (generovaní kódu) brané do úvahy. Výnimku majú len bloky umiestnené v tele blokov definujúcich funkcie a procedúry. Príklad možno vidieť na obrázku 6.3. Je tu zobrazený obsah pracovnej plochy editora VPJ. Blok vľavo bude pri generovaní C++ kódu vyhodnotený, ide o hlavný blok „Program“, ktorého obsah tvorí časť *loop* riadiaceho programu. Blok v strede, definujúci procedúru „do something“, bude tiež vyhodnotený a pripojený v generovanom kóde za časť *loop*, aby bolo možné procedúru z časti *loop* volať. Blok pre cyklus (na obrázku vpravo) bude pri vyhodnocovaní ignorovaný, nakoľko nie je priamym ani nepriamym potomkom bloku „Program“ ani bloku definujúceho procedúru alebo funkciu. Tento stav je používateľovi signalizovaný vizuálne zblednutím bloku.

### 6.2.1 Základné bloky, logika, cykly, aritmetika

Definície blokov kategórií *matematická logika*, *cykly* a *matematika* preberáme od autorov knižnice Blockly. Pri definícii generátora vychádzame z existujúceho, produkujúceho kód v jazyku JavaScript. V aplikácii sú prístupné bloky pre podmienku if, konštanty pravdivostných hodnôt a ich porovnanie, blok pre negáciu, číselné konštanty, unárne mínus, aritmetické operácie a blok pre prístup ku generátoru náhodných čísel.




Obr. 6.4: Abstrakcia bloku testujúceho vlastnosť čísla




Obr. 6.5: Bloky v kategórii *cykly*

Vyššiu abstrakciu pre logické operácie poskytuje napríklad blok „test vlastnosti čísla“ (obrázok 6.4), ktorého návratovou hodnotou je pravdivostná hodnota a používateľovi umožňuje testovať paritu, pozitivitu alebo deliteľnosť čísel na základe slovnej definície tejto vlastnosti. Generátor tento blok do jazyka C++ následne preloží ako príslušnú logickú operáciu, napríklad blok „$x$ je pozitívne“ je prekladaný ako „$x > 0$“, blok „$y$ je párne“ bude interpretovaný ako „$y \,\%\, 2 == 0$)“

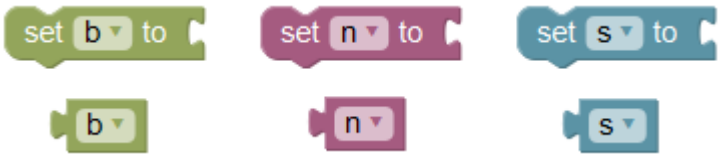
V kategórii cyklov (obrázok 6.5) sú k dispozícii bloky pre štandardný cyklus *for*, *while*, blok pre príkaz *break* a *continue*. Vyššiu abstrakciu umožňuje špeciálny cyklus, pre ktorý používateľ definuje len počet iterácií (na obrázku vpravo dole). Pri generovaní kódu je potom na jeho mieste vytvorená definícia štandardného cyklu *for* s lokálnou premennou, ktorou je iterované v postupných inkrementoch od 0 po užívateľom zadané číslo - 1. Vnútri bloku tohto cyklu môžete vidieť spomínaný blok pre príkazy *break* a *continue*, charakter bloku možno zvoliť výberom z možností.

### 6.2.2 Premenné

Dôležitým komponentom jazyka sú premenné. Blockly je potrebné prispôsobiť pre použitie typových premenných jazyka C, pričom autori podobných aplikácií pristupujú k riešeniu rôzne. V online webovej verzií editora MakeCode (riadenie robota LEGO Mindstorms) sú k dispozícii len číselné premenné, textové reťazce a binárne hodnoty môžu byť len konštantou [23]. Na druhej strane, v produkte Otto Blockly je možné vytvoriť premennú typu znak, reťazec, celé číslo, desatinné číslo i binárna hodnota. K dispozícii sú bloky pre inicializáciu, priradenie, i získanie hodnoty premennej. Typy nie sú však vizuálne výraznejšie odlíšené. V editore kódu možno deklarovať premennú binárneho typu a následne iným blokom túto premennú inkrementovať, na čo validátor bloku oznámi chybu. Otázne však je, či je vhodné povoliť vytvorenie neprípustného spojenia.

V našej aplikácii vynucujeme určenie typu premennej pri jej vytváraní, bez možnosti dodatočnej zmeny. Povolené typy sú binárna hodnota (bool), celé číslo (int16_t) a textový reťazec (String). Bloky pre priradenie do premennej a čítanie premennej majú pre jednoznačnosť v rámci typu rovnakú farbu (obrázok 6.6). Správnosť typu je vynucovaná pri každom pokuse o spojenie blokov. V rámci blokov pre priradenie do premennej (horný rad blokov na obrázku 6.6) a blokov pre čítanie hodnoty z premennej (dolný rad blokov na obrázku 6.6) je k dispozícii ponuka pre výber inej premennej rovnakého typu.

Premenné sú používateľom tvorené kliknutím na tlačidlo v ponuke prvkov VPJ v kategórii /textitpremenné. Pre vytvorenie konkrétneho typu je k dispozícii separátne tlačidlo. V tejto sekcii sú tiež k dispozícii bloky pre čítanie a zápis do každej vytvorenej premennej. Iným spôsobom pre vytvorenie premennej je jej deklarovanie v zozname parametrov procedúry alebo funkcie, či použitie cyklu *for*, kde je automaticky zavedená iteračná premenná typu *číslo*.



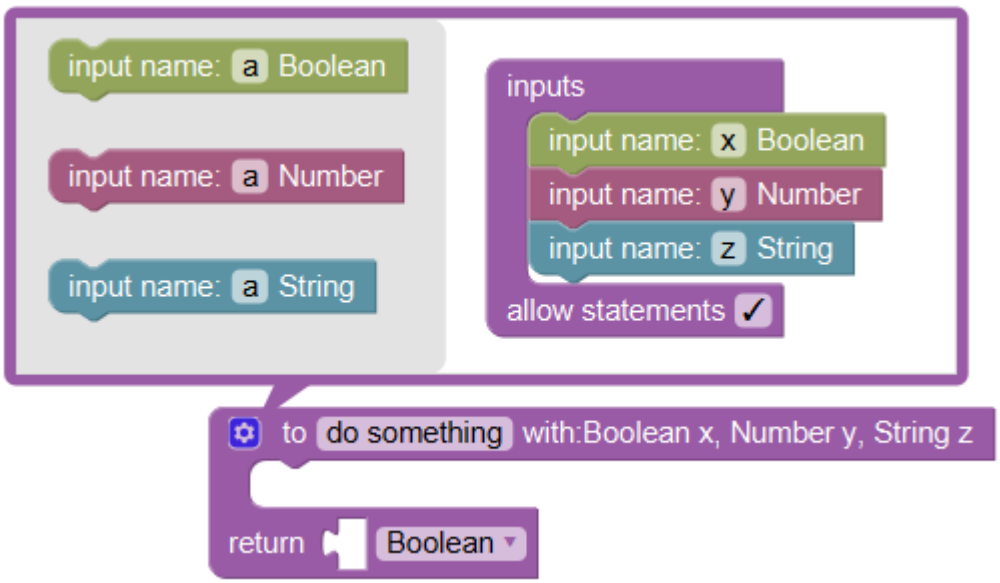

Obr. 6.6: Bloky v kategórii *premenné*

### 6.2.3 Procedúry a funkcie

Bloky pre vytváranie procedúr a funkcií boli modifikované pre podporu typových premenných. V definícii funkcie je nutné deklarovať typ parametrov a pri volaní je následne vynucovaný. Pre tento účel bola rozšírená ponuka dialógového okna pre úpravu počtu parametrov funkcie (obrázok 6.7).

Komplikáciou pri práci s procedúrami a funkciami je určenie rozsahu platnosti premenných, ich argumentov. V Blockly je pre vytváranie blokov umožňujúcich manipuláciu s premennými štandardne určená jedna z kategórií menu ponuky prvkov VPJ. Sú v nej zobrazené všetky vytvorené premenné, lokálne i globálne, všetky, čo boli v editore deklarované. Môže tak ľahko dôjsť k zmätočnej situácii, keď deklarujeme rovnomennú premennú iného typu v rámci dvoch procedúr a v menu sú naraz dostupné obe. Riešením je vynucovanie typu pre raz definovaný názov premennej. Takto možno všetky používateľom vytvorené premenné vyhlásiť za globálne a predísť nežiadaným situáciám.

Blok pre definíciu funkcie bol modifikovaný doplnením pola pre explicitnú špecifikáciu návratového typu (pole pre výber možností, na obrázku 6.7 je zobrazené v pravej dolnej časti bloku, zobrazuje hodnotu „Boolean“).



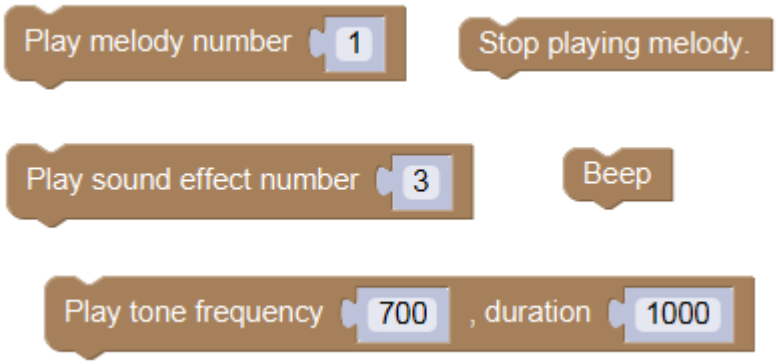

Obr. 6.7: Definícia argumentov funkcie

### 6.2.4 Zvukové efekty a mp3 prehrávač

Bloky v tejto kategórii umožňujú ovládať výstupné zariadenia robota, sirénu a mp3 prehrávač. Poskytujú vysokú abstrakciu nad procesmi v mikropočítači, ktoré zabezpečujú požadovaný výsledný efekt. Pri práci s melódiami (sirénou) je napríklad v mikropočítači implementovaná práca s časovačom, ktorý na základe preddefinovanej postupnosti hodnôt tvorí signály reprezentujúce tóny, následne odosielané do súčiastky tvoriacej zvuk. Implementáciu tejto časti preberáme z pôvodnej verzie riadiaceho programu robota Otto, vyvinutého pre účely denných táborov (kapitola 2.1.4). Vznikli tak bloky pre vydanie konkrétneho tónu ale i komplexnejšie pre zahranie celej preddefinovanej melódie a pre prehranie niekoľkých predpripravených zvukových efektov (obrázok 6.8). Bloky sú parametrizované, číselné hodnoty možno buď to zadať manuálne (konštantou) alebo použitím bloku s výstupom typu *číslo*, napríklad premennej.

Pre ovládanie mp3 prehrávača vznikli bloky pre spustenie konkrétnej skladby, pozastavenie a obnovenie prehrávania, úpravu hlasitosti a blok pre načítanie aktuálne zvolenej hlasitosti (obrázok 6.9). I v tomto prípade zabezpečujú bloky vysokú abstrakciu nad procesom v pozadí. Mp3 prehrávač je riadený odosielaním riadiacich inštrukcií v preddefinovanom formáte.

Pre prehľadnosť kódu tvoreného prekladom použitých blokov je generátorom VPJ vytvorené len volanie funkcie definovanej v podporných súboroch daného logického modulu verzie (kapitola 5.1). Pre ilustráciu uvažujme blok „Prehrať pesničku číslo $x$“. Pre tento úkon je nutné odoslať z mikropočítača do mp3 prehrávača sekvenciu riadiacich inštrukcií [15]. Blok je preložený ako „jednoduché“ volanie funkcie „mp3_play($x$)“, ktorej definícia je umiestnená vo *footer* súbore logického modulu *zvuk*.



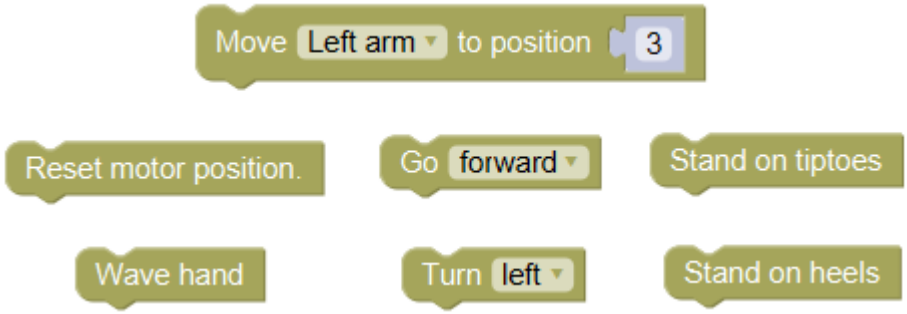

Obr. 6.8: Bloky pre ovládanie zvukových efektov


Obr. 6.9: Bloky pre ovládanie mp3 prehrávača

### 6.2.5 Pohyb

Pre ovládanie servomotorov robota bolo vo VPJ vytvorených niekoľko blokov (obrázok 6.10). Základným je blok pre natočenie konkrétneho motora do konkrétnej polohy (na obrázku 6.10 hore), z ktorého by s použitím ostatných prvkov VPJ bolo možné vyskladať všetky ďalšie, komplexnejšie.

Pre vyššiu úroveň abstrakcie slúžia bloky ako „reset pozícií motorov“, ktorý spôsobí postupné natočenie všetkých motorov do polohy 90 stupňov. Robota tak možno ľahko priviesť do iniciálnej pozície v stoji s upaženými rukami. Bloky „stoj na špičkách“, „stoj na pätách“ a „zamávaj rukou“ sú jednoduchými gestami, spájajúcimi postupný pohyb niekoľkých motorov. Bloky „choď vpred“ a „otoč sa vľavo“ (v poli dostupnom na blokoch možno tiež zvoliť opačný smer, vzad a vpravo) predstavujú komplexný pohyb, ich implementácia je inšpirovaná pôvodným riadiacim programom pre robota Otto a v podporných súboroch logického modulu *pohyb* sú implementované ako prehrávanie krátkej choreografie. Tieto komplexné pohyby sú navrhnuté tak, aby predstavovali jeden ucelený pohyb s pokiaľ možno rovnakou východiskovou a cieľovou polohou všetkých zapojených motorov. Napríklad blok „choď vpred“ by sa dal inak pomenovať ako „krok vpred“. Pre chôdzu robota je tak nevyhnutné tento blok umiestniť do cyklu. Rovnaký princíp platí pre blok otáčania. Nie je tu definovateľná miera natočenia, nakoľko nevieme s istotou povedať, že ju dosiahneme určitou sériou elementárnych pohybov jednotlivých motorov.



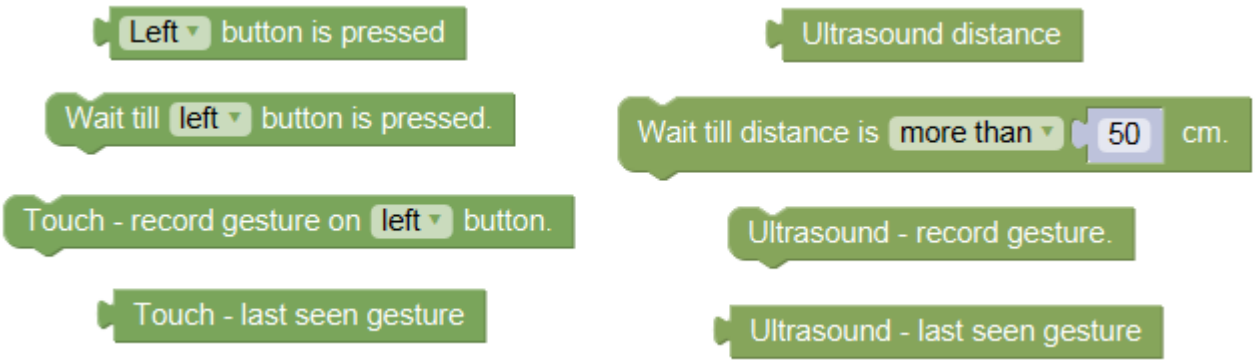

Obr. 6.10: Bloky pre ovládanie servomotorov

### 6.2.6 Senzory

V tejto kategórii je tiež uplatnená vysoká miera abstrakcie. Pre obsluhu dotykových senzorov o čosi menšia, napríklad blok zisťujúci, či je tlačidlo práve stlačené, prekladá generátor priamo na funkciu „digitalRead(TOUCH1)“, kde *digitalRead* je funkcia prístupná v jazyku Arduino, návratovou hodnotou je pravdivostná hodnota, určujúca, či je na danom pine detegované pozitívne napätie. Parameter *TOUCH1* určuje číslo pinu, v ktorom je dane tlačidlo zapojené, táto konštanta je definovaná v pomocnom *header* súbore logického modulu *senzory*. Bloky vyššej abstrakcie tu reprezentujú „čakanie na stlačenie tlačidla“. V tomto prípade je vygenerovaný krátky blok kódu, cyklicky kontrolujúci hodnotu prijímanú na pine prislúchajúcom tlačidlu, kým nie je detegované stlačenie. Vysokú abstrakciu dovoľuje blok pre „načítanie gesta“. V tomto prípade je vykonaná funkcia detegujúca počet stlačení tlačidla. Jej implementáciu môžete vidieť v ukážke 6.2. Pre elimináciu chyby je gesto (jedno zo stlačení) registrované až po niekoľkonásobnej detekcii stlačenia v určitom časovom intervale. Implementácie tiež umožňuje zadanie gesta dlhým stlačením. Po určitom intervale neprestajného „stlačenia“ je zaznamenané ďalšie „stlačenie“. Zadávanie gesta je ukončené po neregistrovaní stlačenia po definovaný čas. Súčasťou blokov pre interakciu s dotykovými senzormi je možnosť voľby senzora (robot Otto má dva).

Pre ultrazvukový senzor boli bloky navrhnuté analogicky. Aktuálne meranú vzdialenosť možno načítať blokom „Ultrazvuk vzdialenosť“ (návratová hodnota je číslo — vzdialenosť v centimetroch), namiesto stlačenia je tu možnosť vyčkávať na detekciu vzdialenosti menšej alebo väčšej ako používateľom zadaná hodnota. Gesto je vyjadrené počtom nameraní vzdialenosti do 50 centimetrov (analógia stlačenia tlačidla).

Posledné zaznamenané gesto (číslo) možno získať použitím bloku „Dotyk — posledné gesto“ (respektíve „Ultrazvuk — posledné gesto“). Bloky pre interakciu so senzormi môžete vidieť na obrázku 6.11.




Obr. 6.11: Bloky pre ovládanie senzorov

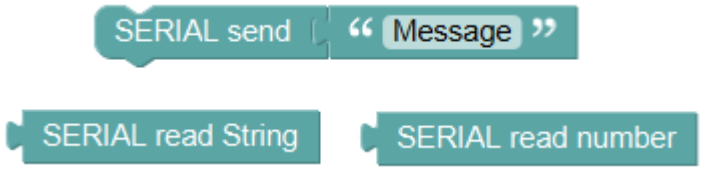
```
void touch_gesture_record(uint8_t sensor, uint16_t time_ignore, uint16_t
    time_max_wait)
{
  uint16_t count = 0;
  uint8_t no_valid_gesture = 0;
  uint32_t time_start = millis();

  while ((millis() - time_start) <= time_max_wait) {
    if (digitalRead(sensor)) {
      no_valid_gesture++;
      delay(10);
      if (no_valid_gesture == 20) {
        count++;

        #ifdef MODULE_SOUND
        beep();
        #endif

        // *** wait time_ignore OR recognise button release
        no_valid_gesture = 0;
        time_start = millis();
        while ((millis() - time_start) <= time_ignore) {
          if (!digitalRead(sensor)) {
            no_valid_gesture++;
            delay(10);
            if (no_valid_gesture == 20)
              break;
          }
          if ((millis() - time_start) >= time_max_wait) {
            TOUCH_last_seen_gesture = count;
            return;
          }
        }
        // ***

        no_valid_gesture = 0;
        time_start = millis();
      }
    }
  }
  TOUCH_last_seen_gesture = count;
}
```

Ukážka kódu 6.2: Pomocná funkcia pre zaznamenanie gesta dotykovým senzorom

### 6.2.7 Sériová komunikácia prostredníctvom USB

Bloky tejto kategórie môžete vidieť na obrázku 6.12. Implementované sú základné operácie pre načítanie prijatej textovej alebo číselnej hodnoty a blok pre odoslanie správy (možno ním odoslať text i číslo). Odosielanie je riešené objektom *Serial*, dostupnom v jazyku Arduino. Blok pre odoslanie správy je prekladaný ako *Serial.println(správa)*. Tento prístup mimo iného zabezpečí, aby všetky správy odoslané v kóde vytvorenom používateľom boli ukončené znakom konca riadku, čo nám v aplikácii umožňuje jednoznačne určiť ich hranice.

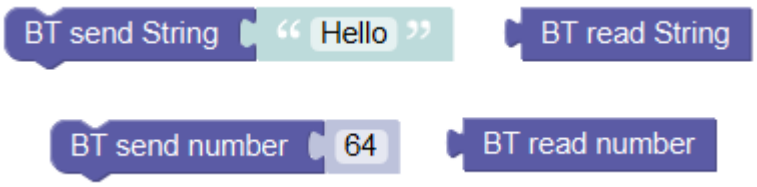
V súvislosti s prijímaním sériovej komunikácie je v Arduino implementovaný buffer. Správy sú doň uložené vždy keď prichádzajú cez sériovú linku a je na používateľovi, kedy ich prečíta. K tomuto účelu je prostredníctvom objektu *Serial* k dispozícii niekoľko funkcií, ktoré ale pre mladšieho používateľa môžu pôsobiť mätúco. Implementujeme preto nad nimi abstrakciu, blok pre načítanie textu a blok pre načítanie čísla. Funkcie pre načítanie hodnoty sú blokujúce, ich volaním sa vykonávanie programu pozastaví, kým nie je prijatá textová (alebo číselná) správa. Buffer je pred volaním funkcie vždy vyprázdnený.




Obr. 6.12: Bloky pre sériovú komunikáciu prostredníctvom USB

### 6.2.8 Sériová komunikácia prostredníctvom Bluetooth

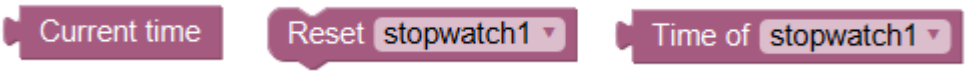
Bloky tejto kategórie môžete vidieť na obrázku 6.13. Analogicky k blokom v ponuke pre obsluhu sériovej komunikácie prostredníctvom USB tu boli definované bloky pre načítane a odoslanie číselnej alebo textovej hodnoty. Proces načítania je rovnako blokujúci, pozastaví vykonávanie programu, kým nie je prijatá správa požadovaného typu a pred týmto úkonom je vždy vyprázdnený súvisiaci buffer.




Obr. 6.13: Bloky pre sériovú komunikáciu prostredníctvom Bluetooth

### 6.2.9 Práca s časom

Táto kategória sprístupňuje používateľovi tri bloky (obrázok 6.14). Blok vľavo umožňuje načítať číselnú hodnotu reprezentujúcu čas v milisekundách od spustenia riadiaceho programu prístupom k Arduino funkcii *millis()*. Abstrakciou nad touto funkciou vznikli bloky pre „stopky“. K dispozícii sú tri nezávisle *stopky*, možno ich resetovať (blok v strede na obrázku 6.14) a načítať čas uplynutý od ich posledného resetovania (blokom vľavo na obrázku 6.14)). Implementácia je tu triviálna — každé stopky sú premennou, „reset stopiek“ priradí do tejto premennej aktuálnu hodnotu *millis()*, čas stopiek je vždy rozdielom aktuálnej hodnoty *millis* a premennej prislúchajúcej daným „stopkám“.



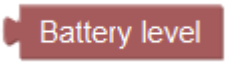

Obr. 6.14: Bloky pre prácu s časom

### 6.2.10 Kontrola batérií

Mechanizmus umožňujúci meranie napätia batérií je súčasťou riadiaceho programu robota Otto vyvinutého pre účely denných táborov. Hodnota je cyklicky meraná a v prípade poklesu pod určitú hodnotu je robot uvedený do núdzového režimu, v ktorom ho nemožno ovládať a vybitie batérií je signalizované vydávaním melódie reprezentujúcej správu SOS. Vo verzii *Otto 2020 Robotická liga* je možné takúto kontrolu ľahko integrovať do riadiaceho programu, nakoľko používateľ nemá prístup k úprave obsahu časti *loop*. Implementácia tejto časti zabezpečuje, aby boli batérie kontrolované počas celého behu programu, v relatívne malých časových intervaloch.

V prípade verzie *Otto 2021 Procedural* však nechávame tvorbu časti *loop* čisto na používateľa. Môže v nej vytvoriť nekonečný cyklus a batérie nikdy skontrolované nebudú. Riešením je pridanie kontroly do každej, potenciálne nekonečne vykonávanej časti. Jedná sa predovšetkým o cykly *while* a funkcie umožňujúce čakanie na nameranie hodnoty zo senzorov.

Používateľovi je v súvislosti s týmto mechanizmom sprístupnený blok pre načítanie hodnoty aktuálneho stavu batérií (obrázok 6.15).




Obr. 6.15: Blok pre načítanie aktuálneho napätia batérií

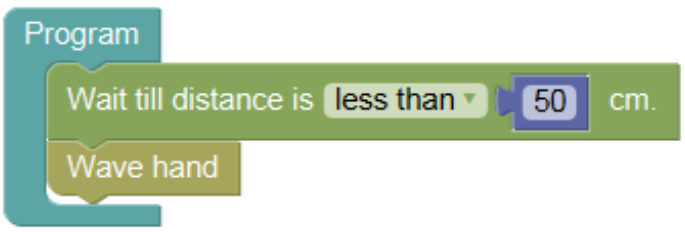
# Kapitola 7

# Ukážky programov

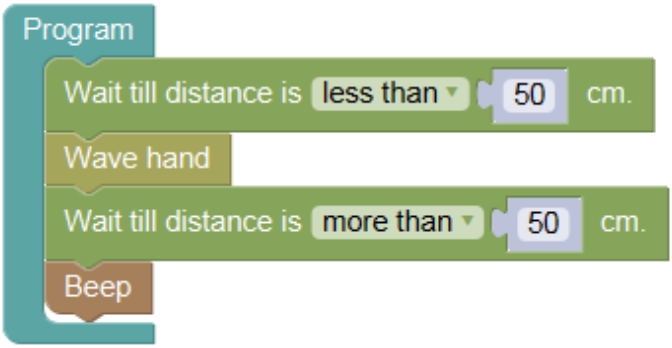
V kapitole uvádzame niekoľko ukážok programov vytvorených v našej aplikácií. Tieto programy sú v nej k dispozícií pre začiatočníkov, ktorým môžu poskytnúť o niečo pohodlnejšiu štartovaciu pozíciu v procese zoznamovania sa s robotom a softvérom.

## 7.1 Ultrazvukový senzor, čakanie

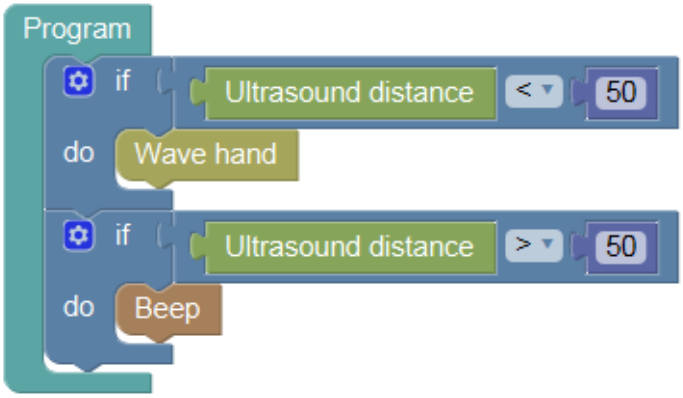
Pre rýchle zoznámenie sa s možnosťami ultrazvukového senzora a významom súvisiaceho bloku pre *čakanie* môže poslúžiť kód na obrázkoch 7.1, 7.2, 7.3 a 7.4. Používateľ tak odhalí význam bloku *čakania* a princíp nekonečného vyhodnocovania časti *loop* riadiaceho programu. Program na obrázku 7.1 predstavuje triviálnu implementáciu, robot máva rukou vždy, keď sa k nemu priblížime na menej ako pol metra. Ukážky 7.2 a 7.3 predstavujú zdanlivo identické programy, no aspekt *čakania* z nich sémanticky robí celkom odlišné algoritmy. Ukážka na obrázku 7.4 predstavuje alternatívny zápis programu v ukážke 7.2, používateľovi má priblížiť logický význam blokov pre *čakanie*, „čakanie je cyklické vykonávanie ničoho“.



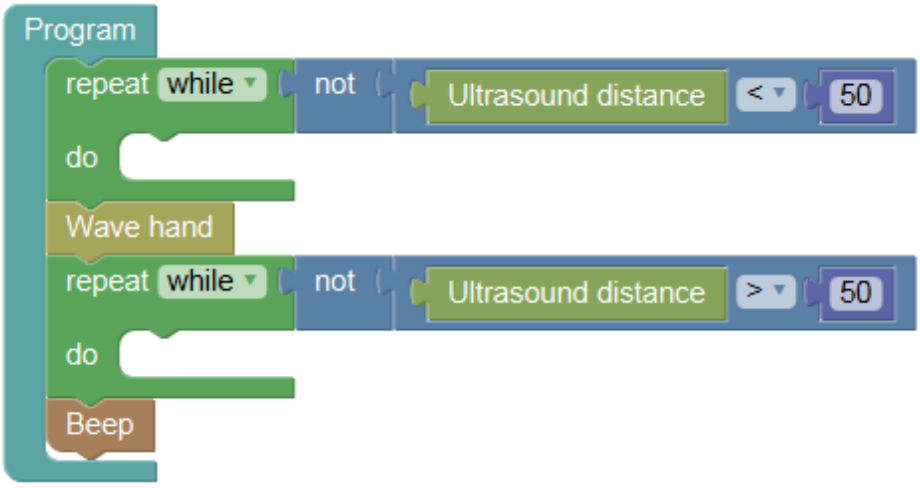

Obr. 7.1: Ukážka programu — zoznámenie s ultrazvukovým senzorom




Obr. 7.2: Ukážka programu — zoznámenie s ultrazvukovým senzorom 2



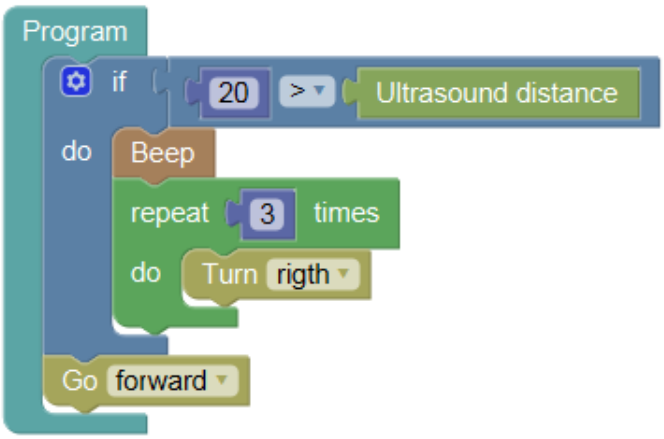

Obr. 7.3: Ukážka programu — zoznámenie s ultrazvukovým senzorom 3




Obr. 7.4: Ukážka programu — zoznámenie s ultrazvukovým senzorom 4

## 7.2 Komplexný pohyb s detekciou prekážky

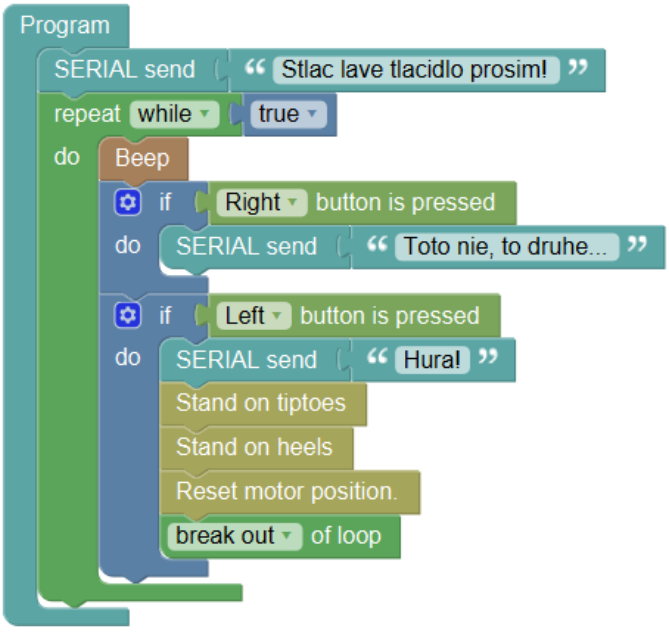
Ukážka 7.5 má za úlohu používateľovi v krátkosti ilustrovať využitie spojenia blokov pre pohyb, načítanie hodnoty zo senzora a cyklov. Praktický príklad ukazuje, ako sa možno na základe údajov o vzdialenosti vyhnúť blízkej prekážke.




Obr. 7.5: Ukážka programu „vyhni sa prekážke“

## 7.3 Výpis cez sériový port USB

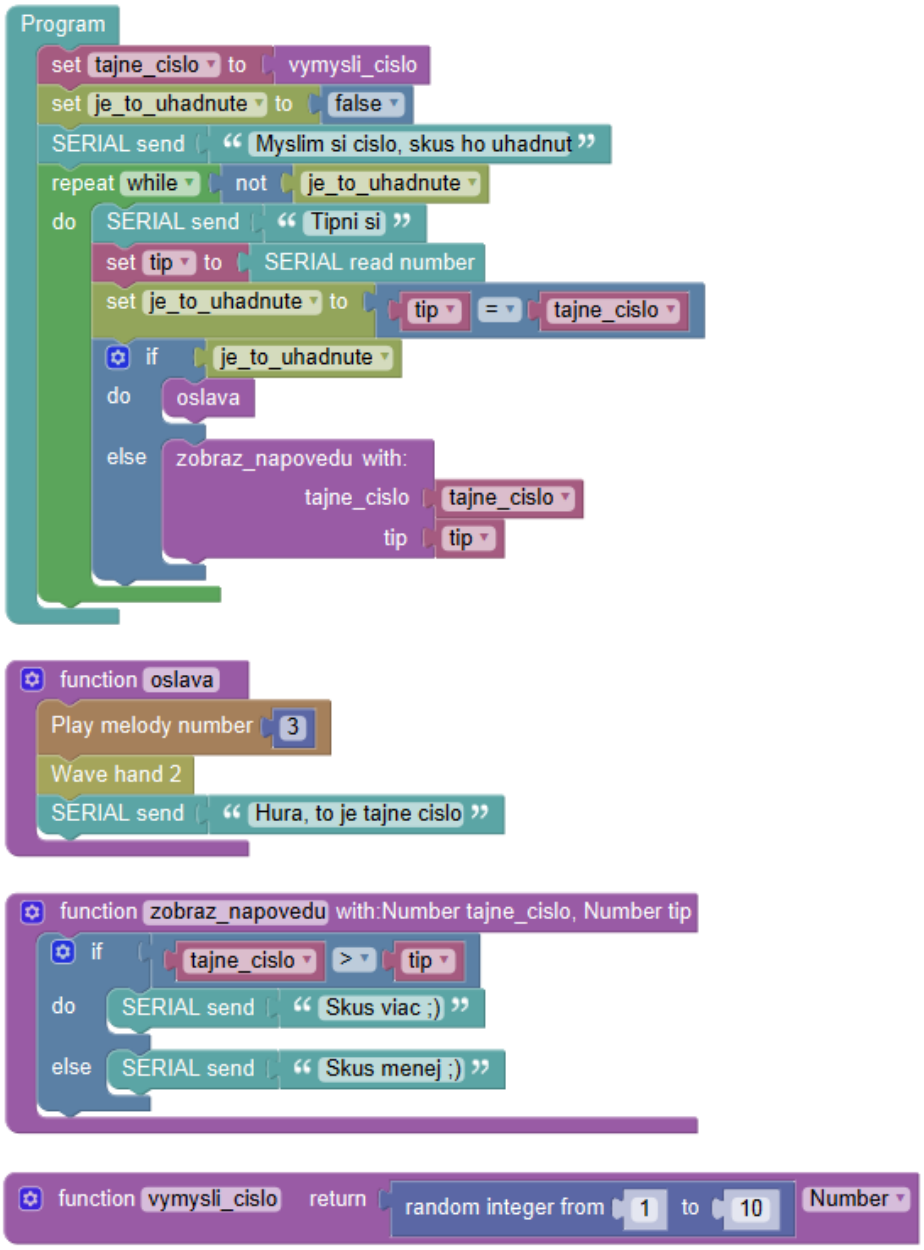
Ukážka 7.6 demonštruje, ako je možné jednoducho doplniť interakciu s robotom použitím sériovej komunikácie.




Obr. 7.6: Ukážka programu „stlač ľavé tlačidlo“

## 7.4 Funkcie a procedúry

V nasledujúcej ukážke 7.7 používateľa oboznamujeme s princípom procedúr a funkcií, ktoré môžu vo vhodnej forme uľahčiť čitateľnosť kódu. Program vždy vygeneruje náhodné číslo a nechá používateľa hádať. Ak uhádne, je vydaný „oslavný“ tón a vykonané jednoduché pohybové gesto. Ak je tip nesprávny, program reaguje odoslaním správy, či bol tip väčší alebo menší ako práve hádané číslo. V nenáročnom cvičení možno tento program prerobiť na tipovanie pomocou gesta rozpoznávaného senzorom alebo pridať implementáciu počítadla skóre.




Obr. 7.7: Ukážka programu „hádaj číslo“

# Záver

V práci sme sa venovali vývoju desktopovej aplikácie s grafickým používateľským rozhraním, umožňujúcej programovať jednoduchého výučbového robota Otto v navrhnutom vizuálnom programovacom jazyku. Najprv sme sa oboznámili s možnosťami spomínaného robota a tvorbou programov pre jednočipový mikropočítač Arduino Nano Strong v jazyku Arduino (C++), ktorým je riadený. Následne vznikal systém, desktopová aplikácia s viacerými modulmi.

Funkcie aplikácie boli z počiatku tvorené v nadväznosti na existujúci riadiaci program robota, umožňujúci ovládanie prostredníctvom znakového terminálu. Pre tento účel vznikol v aplikácii integrovaný terminál sériovej komunikácie. Používateľovi dovoľuje jednoducho vyhľadať dostupné sériové porty a nadviazať spojenie s mikropočítačom.

Odoslaním niekoľkých riadiacich príkazov bolo možné komunikáciou s existujúcim riadiacim programom vytvoriť v pamäti mikropočítača jednoduchú choreografiu. Jedným z našich primárnych cieľov bolo umožniť tvorbu týchto choreografií v prehľadnom vizuálnom jazyku. Pre naplnenie tejto požiadavky sme sa zoznámili s JavaScript knižnicou Blockly, nástrojom pre ich tvorbu. V navrhnutom jazyku vznikli bloky reprezentujúce jednotlivé časti choreografie, programom následne prekladané na ich zavedenú číselnú reprezentáciu, príkazy odosielané sériovou komunikáciou. Bez zmeny riadiaceho programu sme tak umožnili prehľadnú tvorbu jednoduchých sekvencií pohybov.

V ďalšom bol zavedený alternatívny prístup k tvorbe riadiacich programov. Navrhli sme vizuálny programovací jazyk, ktorého prvky reprezentujú rôzne časti jazyka C, ktorým je mikropočítač programovaný. Všetok kód je pritom tvorený výhradne vo vizuálnom jazyku a znalosť jazyka C nie je pre použitie aplikácie nutná. Prístupné sú komponenty pre vetvenie programu, cykly, základné premenné, procedúry a funkcie. V jazyku tiež možno ovládať špecifické funkcie robota. Boli zavedené bloky pre pohyb motorov, prácu so senzormi ale i bloky umožňujúce prijímať a odosielať správy prostredníctvom sériovej komunikácie pripojením rozhrania USB alebo Bluetooth. Takto vytvorený kód je z reprezentácie vo VPJ najprv interpretovaný v jazyku C, potom kompilovaný prepojením s aplikáciou Arduino IDE a následne odoslaný robotovi.

Pri zavádzaní prvkov VPJ bol kladený dôraz na rôznorodosť ich abstrakcie. V kategórii blokov umožňujúcich interakciu so senzormi tak vznikol blok pre jednoduché

načítanie aktuálnej vzdialenosti meranej ultrazvukom no napríklad i blok pre čakanie na nameranie konkrétnej hodnoty. Odhad miery abstrakcie nebol ľahkou úlohou, dúfame však, že poskytnutý rozsah pokryje požiadavky rôznych vekových kategórií.

V aplikácii bol zavedený modulárny systém verzií umožňujúci jej jednoduché rozšírenie o ďalšie funkcie, najmä pridanie ďalších prvkov vizuálneho jazyka. Systém tiež zavádza možnosť logického rozčlenenia funkcií riadiaceho programu. Je tak možné obmedziť dostupné funkcie jazyka, čo môže uľahčiť počiatočné zoznámenie s jeho funkciami a v prípade potreby tiež ušetriť miesto v pamäti mikropočítača. V snahe uľahčiť používateľom fázu prvotného zoznámenia sa s aplikáciou boli tiež vytvorené jednoduché ukážkové programy, na ktoré možno pri práci nadviazať.

Zamýšľaný modul simulácie robota bol vytvorený, no len v obmedzenej miere. Simulovať možno len programy vytvorené pre verziu *Otto 2020 Robotická liga*, konkrétne časti choreografií reprezentujúce pohyb. V tomto smere je možné aplikáciu v budúcnosti značne rozšíriť. Jedným zo zvažovaných riešení pre simuláciu riadiaceho programu prekladaného do jazyka C bolo spustiť vytvorený program ako samostatný proces, s ktorým by bolo následne možné komunikovať prostredníctvom socketov. V ňom použité volania funkcií pre riadenie motorov a senzorov by bolo nutné „presmerovať“ do modulu simulácie našej aplikácie.

K záveru sa nám podarilo previesť i jednoduchý experiment. Prácu s aplikáciou si vyskúšala jedna z účastníčok robotického krúžku. Žiačka ôsmeho ročníka základnej školy v aplikácii vytvorila niekoľko jednoduchých programov, potešila ju najmä možnosť intuitívnej tvorby riadiaceho programu, v ktorom sa veľmi rýchlo zorientovala. Vyjadrila tiež nápady pre prácu s aplikáciou do budúcna, zaujala ju najmä možnosť tvorby tancov. Experimentom boli tiež odhalené niektoré nedostatky, predovšetkým v počiatočnej inštalácii, ktorá nie je pre začiatočníka triviálnou záležitosťou (mnoho komponentov našej aplikácie vyžaduje osobitnú pozornosť, je nutné nainštalovať Java JRE, súbory JavaFX, Arduino IDE a navyše ovládač sériového portu).

Veríme, že naša aplikácia uľahčí prácu začínajúcim programátorom, poskytne im priestor pre rozvoj fantázie a zručností a vzbudí v nich záujem ďalej objavovať možnosti a zákony všetkých zapojených oblastí.

# Literatúra


[1] Alternative programming languages for LEGO Mindstorms, Wayne Burnett, LEGO Engineering Center for Engineering Education and Outreach. Tufts University 2018. http://www.legoengineering.com/alternative-programming-languages/. [Citované 2021-03-25].

[2] Arduino — SoftwareSerial library. https://www.arduino.cc/en/Reference/softwareSerial. [Citované 2021-04-22].

[3] Arduino - serial communication parameters. https://www.arduino.cc/reference/en/language/functions/communication/serial/begin/. [Citované 2021-04-22].

[4] Arduino and Java. https://playground.arduino.cc/Interfacing/Java/. [Citované 2021-04-22].

[5] Arduino bulder. https://github.com/arduino/arduino-builder. [Citované 2021-04-22].

[6] Arduino CLI. https://github.com/arduino/arduino-cli. [Citované 2021-04-22].

[7] Arduino language reference. https://www.arduino.cc/reference/en/. [Citované 2021-01-14].

[8] Arduino Nano Strong - špecifikácia. https://dratek.cz/docs/produkty/0/643/1512125230.pdf. [Citované 2021-01-14].

[9] Basics of UART communication, scott campbell, diy electronics. https://www.circuitbasics.com/basics-uart-communication/. [Citované 2021-04-22].

[10] Blockly - best practices. https://developers.google.com/blockly/guides/app-integration/best-practices. [Citované 2021-04-12].

[11] Centrum vedecko-technických informácií SR. https://www.cvtisr.sk/o-cvti-sr/zakladne-informacie.html?page_id=409. [Citované 2021-02-20].

[12] Centrum vedecko-technických informácií SR - projek IT Akadémia. https://www.cvtisr.sk/cvti-sr-vedecka-kniznica/projekty/narodne-projekty/it-akademia-vzdelavanie-pre-21.-storocie/it-akademia-vzdelavanie-pre-21.-storocie.html?page_id=34856. [Citované 2021-02-20].

[13] CLI of Arduino IDE. https://github.com/arduino/Arduino/blob/master/build/shared/manpage.adoc. [Citované 2021-04-22].

[14] Denný tábor digitálnych technológií - Fablab Bratislava. https://www.fablab.sk/podujatia/denne-tabory/. [Citované 2021-02-22].

[15] DFPlayer Mini MP3 player for Arduino. https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299. [Citované 2021-05-05].

[16] Fablab. https://fabfoundation.org/getting-started/. [Citované 2021-02-22].

[17] The first electronic autonomous robots: the origin of social robotics. https://www.historyofinformation.com/detail.php?id=668. [Citované 2021-04-22].

[18] ISO 8373:2012(en) robots and robotic devices — vocabulary. https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en:term:2.6. [Citované 2021-02-23].

[19] IT Akadémia. http://itakademia.sk/zakladne-informacie. [Citované 2021-02-22].

[20] jMonkeyEngine — documentation. https://wiki.jmonkeyengine.org/docs/3.4/documentation.html. [Citované 2021-04-07].

[21] jMonkeyEngine3 – JavaFX. https://github.com/JavaSaBr/JME3-JFX. [Citované 2021-04-07].

[22] jSerialComm. https://github.com/Fazecast/jSerialComm. [Citované 2021-04-22].

[23] LEGO Makecode web editor. https://makecode.mindstorms.com/#editor. [Citované 2021-04-26].

[24] LEGO Mindstorms education EV3 core set. https://education.lego.com/en-us/products/lego-mindstorms-education-ev3-core-set/5003400. [Citované 2021-03-25].

[25] LEGO Mindstorms ultrasonic sensor. https://www.lego.com/en-sk/product/ev3-ultrasonic-sensor-45504. [Citované 2021-03-25].

[26] LibGDX project setup tool. `https://libgdx.com/dev/project-generation/`. [Citované 2021-04-06].

[27] LWJGL. `https://www.lwjgl.org`. [Citované 2021-04-03].

[28] Otto - Robotická liga 2020 - tance. `https://dtdt.fablab.sk/w/index.php/Otto_-_Robotická_liga_2020_-_tance`. [Citované 2021-05-05].

[29] Otto Blockly. `https://github.com/OttoDIY/blockly`. [Citované 2021-01-14].

[30] Otto DIY. `https://www.ottodiy.com/about`. [Citované 2021-01-13].

[31] Overview the arduino sketch uploading process and ISP, dmjlambert. `https://www.instructables.com/Overview-the-Arduino-sketch-uploading-process-and-/`. [Citované 2021-04-22].

[32] Putty. `https://the.earth.li/~sgtatham/putty/0.74/puttydoc.txt`. [Citované 2021-04-22].

[33] Robot Mokrarosa. `https://mokrarosa.ninja/w/index.php/MoKraRosA`. [Citované 2021-01-14].

[34] Serial communication. `https://learn.sparkfun.com/tutorials/serial-communication/all`. [Citované 2021-04-22].

[35] Ultrazvukový senzor vzdialenosti HC-SR04. `https://www.aliexpress.com/item/1005001621997017.html`. [Citované 2021-04-25].

[36] Vo vedeckom parku oslávil Fablab svoje 2. výročie. `https://cusp.uniba.sk/detail-aktuality/back_to_page/uvp/article/vo-vedeckom-parku-oslavila-kreativna-dielna-fablab-svoje-2-vyrocie`. [Citované 2021-02-22].

[37] Webots. `https://cyberbotics.com/doc/guide/introduction-to-webots`. [Citované 2021-04-03].

[38] What is Arduino? `https://www.arduino.cc/en/Guide/Introduction`. [Citované 2021-01-14].

[39] What is LabVIEW? `https://www.electronics-notes.com/articles/test-methods/labview/what-is-labview.php`. [Citované 2021-03-25].

[40] A. Birk. Behavior-based robotics, its scope and its prospects. In *IECON '98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No.98CH36200)*, volume 4, pages 2180–2185 vol.4, 1998.

[41] Marat Boshernitsan and Michael Sean Downes. *Visual programming languages: A survey*. Citeseer, 2004.

[42] Cynthia Breazeal, Kerstin Dautenhahn, and Takayuki Kanda. Social robotics. *Springer handbook of robotics*, pages 1935–1972, 2016.

[43] Adam Haber, Matthew McGill, and Claude Sammut. Jmesim: An open source, multi platform robotics simulator. In *Proceedings of Australasian ference on Robotics and Automation (December 2012)*, 2012.

[44] Thomas R. Kurfess. *Robotics and Automation Handbook*. CRC Press, 2018.

[45] Petrovic P. and Vasko J. (2020) An Open Solution for a Low-Cost Educational Toy, Robotics in Education, Vienna 2019.

[46] Petrovic P. and Vasko J. MoKraRoSA: A constructionist platform for all ages and talents. Constructionism 2020.

[47] Rachel Parker and Bo Stjerne Thomsen. Learning through play at school. *The LEGO Foundation*, 2019.

[48] Erik Pasternak, Rachel Fenichel, and Andrew N Marshall. Tips for creating a block language with Blockly. In *2017 IEEE Blocks and Beyond Workshop (B&B)*, pages 21–24. IEEE, 2017.