UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

VÝUKOVÉ PROSTREDIE NA PROGRAMOVANIE MOBILNÉHO GUĽOVÉHO ROBOTA

Bakalárska práca

2014 MAREK MELICHERČÍK

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

VÝUKOVÉ PROSTREDIE NA PROGRAMOVANIE MOBILNÉHO GUĽOVÉHO ROBOTA

Bakalárska práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Vedúci: Mgr. Pavel Petrovič, PhD.

Bratislava, 2014 MAREK MELICHERČÍK

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Marek Melicherčík
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Výukové prostredie na programovanie mobilného guľového robota / *Educational Environment for Programming a Mobile Spherical Robot*

Cieľ: Preskúmať spôsoby riadenia a programovania mobilného guľového robota. Navrhnúť a implementovať aplikáciu pre operačný systém Android určenú na riadenie a programovanie mobilného guľového robota pomocou ikonografického detského programovacieho jazyka. Overenie funkčnosti vyvinutého systému prostredníctvom sady príkladov vrátane návodov.

Literatúra: R.Meier: Professional Android 4 Application Development, John Wiley & Sons, 2012.
Orbotix: Sphero Android SDK - API Reference, Orbotix, 2013.

Kľúčové slová: mobilné aplikácie, guľový robot, android, detské programovacie jazyky

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 22.10.2013

Dátum schválenia: 22.10.2013

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

.....
študent

.....
vedúci práce

Čestné vyhlásenie:

Vyhlasujem, že predloženú bakalársku prácu som vypracoval samostatne. Použil som literatúru, ktorú uvádzam v zozname použitej literatúry.

Bratislava 31. 5. 2014 .....

Poďakovanie:

Chcel by som sa poďakovať vedúcemu bakalárskej práce Mgr. Pavlovi Petrovičovi, PhD. za cenné rady, odbornú pomoc a čas strávený na stretnutiach. Zároveň by som chcel poďakovať RNDr. Petrovi Borovanskému PhD., ktorý mi poskytol robota Sphero počas tvorby práce.

Abstrakt

MELICHERČÍK, Marek: Výukové prostredie na programovanie mobilného guľového robota [Bakalárska práca]. Fakulta matematiky, fyziky a informatiky; Univerzita Komenského. Katedra aplikovanej informatiky. Vedúci bakalárskej práce: Mgr. Pavel Petrovič, PhD. Bratislava: FMFI UK, 2014 48 s.

Cieľom práce bolo vytvoriť grafický programovací jazyk na programovanie robota Sphero, ktorý bude jednoduchý a prehľadný pre pohodlnú prácu s ním na mobilnom zariadení s menším displejom. Jazyk by mal čo najviac využívať funkcie a možnosti robota v súlade s bežne zaužívanými programátorskými konceptami. Jazyk bude zasadený do aplikácie bežiacej na platforme Android. Aplikácia musí byť postavená tak, aby bola plne funkčná na čo najväčšom množstve zariadení využívajúcich túto platformu.

Kľúčové slová: mobilné aplikácie, guľový robot, android, detské programovacie jazyky

Abstract

MELICHERČÍK, Marek: Educational Enviroment for Programming a Mobile Spherical Robot [Bachelor work]. Faculty of Mathematics, Physics and Informatics; Comenius University. Department of Applied Informatics. Lead by: Mgr. Pavel Petrovič, PhD. Bratislava: FMFI UK, 2014 48 s.

The major goal of the thesis is to create a graphic programming language for programming of a robot Sphero. The program must be simple and clear for comfortable work with it on a mobile device with smaller display. The language should be able to use as much of the robot`s functions as possible in accordance with common programming conceptions. The language will be placed in an application running on the Android platform. The application must be designed so, that it will be fully functional on as many devices as possible.

Keywords: mobile applications, spherical robot, android, programming language for children

Obsah

Úvod

Žijeme v dobe, v ktorej sú informačné technológie neoddeliteľnou súčasťou nášho života. Či chceme alebo nie, obklopujú nás dennodenne pri čo raz bežnejších činnostiach a asi väčšina z nás nerozumie základným logickým princípom ich fungovania.

Tak ako je potrebná k základnému vzdelaniu znalosť matematiky, s ktorou sa tak isto často stretávame je stále žiaducejšie, aby sme poznali aspoň základné znalosti v oblasti programovania. Jeho učenie nevedie len k samotným znalostiam v tejto oblasti, ale tiež dobre rozvíja logické myslenie. Logiku je dobré trénovať už od útleho veku, aby sa človek naučil rozmýšľať racionálne a zjednodušovať si svoju prácu.

K znalostiam programovania sa najskôr potrebujeme naučiť jeho základné princípy, po zvládnutí ktorých, dokážeme vytvárať postupne aj tie najzložitejšie algoritmy. Reálne programovanie v dnešnej dobe však predstavuje častokrát zložité syntaktické riešenia, ktoré sú len veľmi ťažko alebo až nezvládnuteľné pre deti. Preto boli vyvinuté programovacie jazyky za účelom vzdelávania sa v oblasti programovania.

Tieto jazyky nepoužívajú znakovú (slovnú) syntax, ale grafickú. Ich jednotlivé príkazy sú reprezentované graficky hlavne za pomoci obrázkov, ktoré majú jednoducho dieťaťu ilustrovať to, čo vykonávajú. Kľúčové je teda, aby výsledný grafický kód bol dobre čitateľný, na čom sa podieľa aj dobre spracovaná celková štruktúra stavby grafického kódu. Ak toto nie je dobre zvládnuté, v kóde sa má problém vyznať aj programátor, nakoľko nie je jasné, čo čo vykonáva, čo kde začína a končí a čo sa kam vetví.

Program takéhoto kódu musí ovládať tak isto niečo, čo je deťom blízke. Zväčša to býva virtuálny alebo reálny robot.

Ideálne je teda využiť čo najviac vecí, ktoré sú pre dieťa v dnešnej dobe zaujímavé, sú súčasné a dieťa s nimi prichádza často do styku. Túto myšlienku sa snažím v mojej práci zastúpiť spojením spomínaného grafického programovania a mobilného zariadenia (smartfón alebo tablet).

V mojej bakalárskej práci sa zaoberám vytvorením grafického programovacieho jazyka na ovládanie guľového robota Sphero pre mobilné zariadenia s operačným systémom Android.

1 Analýza problematiky

Vyskúšal som si niekoľko grafických programovacích jazykov a pri ich sledovaní som sa zameral hlavne na ich grafickú stavbu, štruktúru a podporu programovacích konceptov, nakoľko to považujem za jeden z najkľúčovejších faktorov. Pri vytváraní grafického programovacieho jazyka na dotykové zariadenie ako je tablet či smartfóne je potrebné dobre rozvrhnúť priestor prostredia a zvoliť správne tvary, stavbu jednotlivých základných častí (prvkov pre funkcie jazyka), pretože je displej týchto zariadení dosť malý. Základom je časť, ktorá jasne zobrazuje to, akú funkciu reprezentuje a zároveň nezaberá veľa priestoru. Za dôležité pokladám aj to, aby spojené časti programu boli na prvý pohľad prehľadné, aby bolo jasné, kde aká časť začína, kde aká končí a ktorá časť sa kam vetví.

1.1 Grafické programovacie jazyky

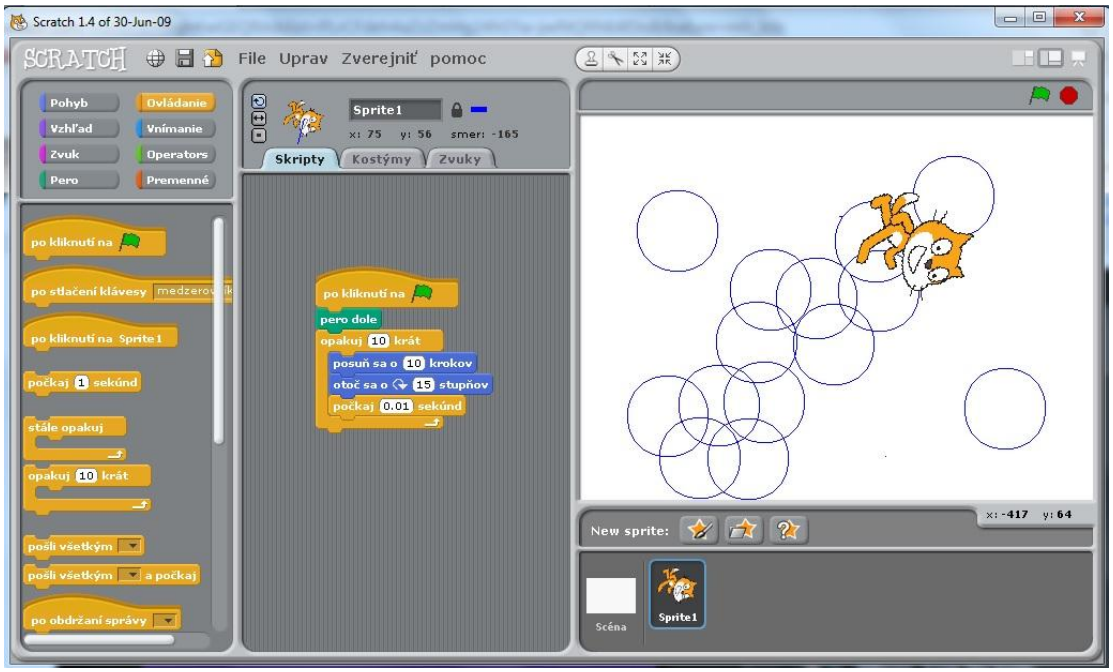
Googleplay obchod na aplikácie pre operačný systém Android neponúka takmer žiadne aplikácie, ktoré poskytujú možnosti grafického programovacieho jazyka, preto som hľadal inšpiráciu hlavne v jazykoch a ich programoch vytvorených pre počítače. Tu sú oveľa väčšie možnosti a v dnešnej dobe už nájdeme aj celkom kvalitne spracovanú formu tohto programovania. Všetky mnou vyskúšané jazyky boli postavené na systéme skladania „drag-and-drop“. Je to systém, kde si vyberiem nejakú programovú časť, zoberiem ju myškou a potiahnem na miesto, kam ju chcem do ostatnej zloženej programovej časti vložiť.

Tieto programovacie jazyky programovali buď správanie nejakého virtuálneho robota (postavičky) alebo správanie reálneho robota.

Možnosti týchto jazykov sú teda vždy obmedzené možnosťami robota, napríklad pohybom alebo senzormi, ktoré ponúka.

1.1.1 Scratch

Scratch je nový programovací jazyk, ktorý vyvinula skupina Lifelong Kindergarten Group z laboratórií MIT Media Lab. Je postavený na „drag-and-drop puzzle“ systéme, kde jednotlivé inštrukcie, funkcie, operácie majú grafický tvar pripomínajúci časť puzzle. [1]



Obrázok 1 Programovacie prostredie jazyka Scratch

Jednotlivé kategórie funkcií sú farebne odlíšené, čo napomáha lepšej orientácií v kóde. Veľmi dobré je tiež, že na každej časti (funkcii) je napísane čo vykonáva.

Nevýhodou však je, že pri zložitejšej funkcii, nejakom dlhšom kóde, sa stráca prehľadnosť a na prvý pohľad nie je celkom jasné, kde sa čo začína a kde končí, kam ide jedna vetva programu a kam druhá.

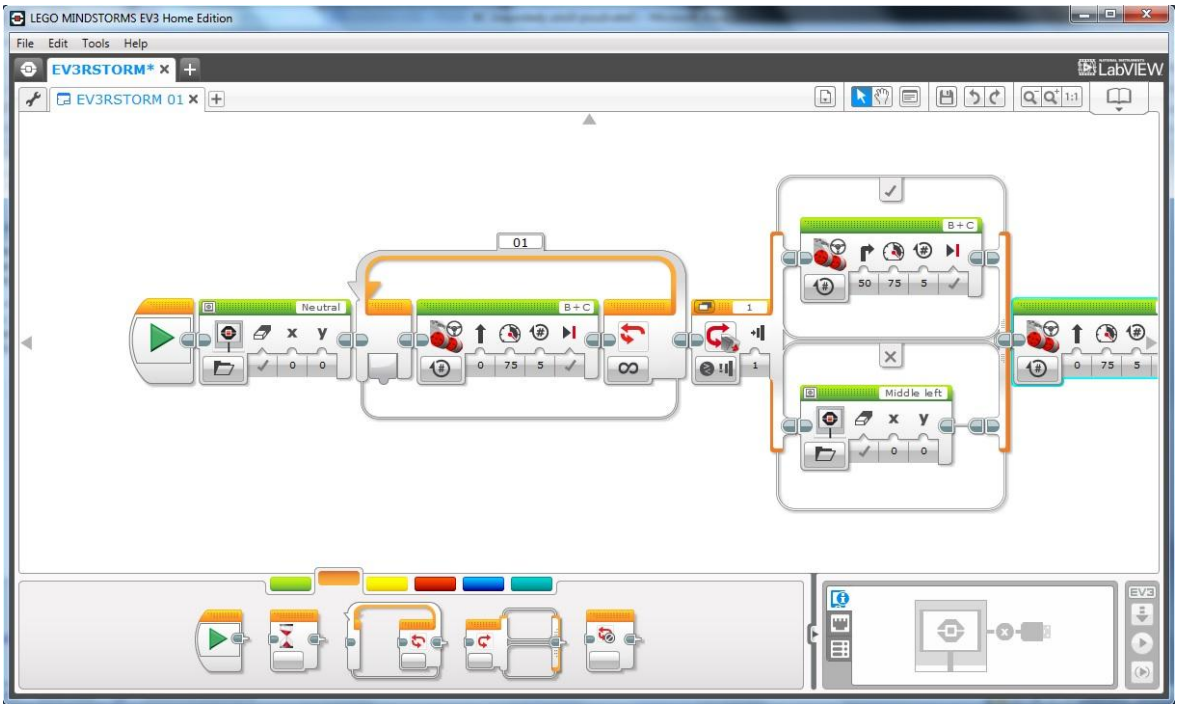
Scratch obsahuje veľké množstvo základných, ale aj pokročilých programátorských konceptov. Zo základných je to podpora sekvencie teda po sebe nasledujúcich príkazov, iterácie a aj nekonečné v reálnom čase bežiace iterácie. Podmienky, pri ktorých je potrebné spomenúť dobre spracované matematické a logické operátory. V palete operátorov nájdeme dokonca aj funkcie ako dĺžka slova, zaokrúhľovanie, náhodný výber či goniometrické funkcie. Ďalej máme možnosť vytvárania vlastných premenných buď globálnych alebo objektových. [2]

Medzi pokročilejšie koncepty tohto jazyka patrí možnosť vytvorenia premennej typu zoznam, podpora paralelného programovania teda možnosť spustenia viacerých blokov programu naraz. [2]

Toto všetko z neho vytvára dosť silný programovací nástroj, ktorý však môže odradiť jeho spomínanou grafickou stavbou, pri ktorej sa stráca prehľadnosť vo väčších funkciách.

1.1.2 EV3

EV3 je grafický programovací „drag-and-drop“ jazyk vytvorený na programovanie robota Lego Mindstorms NXT 2.0. Jednotlivé funkcie sú reprezentované grafickými blokmi, na ktorých je ikona funkcie, ktorá reprezentuje to, čo by mala vykonávať. Pod týmito ikonami si však nie každý predstaví zámer ich tvorcov, názov funkcie sa ale zobrazí aspoň po nadídení myšou. Tieto ikonky preto trochu uberajú čitateľnosti kódu. Na bloku funkcie sú taktiež všetky vstupné premenné, ktoré sú tak isto reprezentované ikonkami. [3]



Obrázok 2 Programovacie prostredie jazyka EV3

Kód je v konečnom dôsledku celkom prehľadný z hľadiska grafického riešenia iterácií a vetvenia v podmienkach. Jednotlivé časti, vetvy kódu sa do seba nezlievajú a nie je problém sledovať, čo kde začína, končí a čo sa kam vetví.

Nevýhodou však je, že kód ide dosť do šírky a jednotlivé bloky sú dosť veľké, čo znamená, že sa ho veľa na pracovnú plochu, obrazovku nezmestí.

EV3 má podporu klasickej sekvencie, iterácie či podmienok. Nechýba ani možnosť definovania vlastných premenných, konštánt či dokonca poľa. Tento jazyk obsahuje aj funkcie ako zaokrúhľovanie či výber náhodného čísla alebo logickej hodnoty. Koncept paralelného programovania mu tiež nechýba. Tieto programové bloky však musia bežať od

začiatku programu. Synchronizovať na nejakú udalosť ich je teda možné buď pomocou čakania alebo iterácie s podmienkou.

1.1.3 WeDo

WeDo je grafický programovací jazyk vytvorený pre programovanie Lego robotov. Rovnako je postavený na systéme „drag-and-drop“. Funkcie tohto jazyka sú reprezentované blokmi v tvare menšieho štvorca. V strede bloku je ikonka zobrazujúca to, čo funkcia vykonáva. Ikonky sú dosť veľké, pekne ilustrované a inštrukcií nie je tak veľa, takže sa ľahko pamätajú a teda čitateľnosť kódu je celkom dobrá.

Jednotlivé bloky sú zafarbené podľa kategórie, do akej ich funkcia patrí, čo tiež zlepšuje prehľadnosť kódu.

Obrázok 3 Programovacie prostredie jazyka WeDo

Ak má funkcia nejaký vstupný parameter, ten sa pripája na spodok bloku. Veľkosť blokov je primeraná, nie veľká ani malá a na pracovnú plochu, obrazovku sa zmestí dostatok kódu.

WeDo podporuje sekvencie a iterácie, no obrovskou nevýhodou je absencia podpory podmienok. Z toho vyplýva nemožnosť vetvenia programových blokov. Nenájdeme tu ani možnosť vytvorenia vlastnej premennej, čo je tak isto veľká nevýhoda.

WeDo je teda jednoduchý jazyk, ktorý nám však veľa toho neponúkne.

1.2 Operačný systém Android

Do úvahy spadajú iba dva operačné systémy, ku ktorým je vytvorené API na programovanie robota Sphero. Je to Android od spoločnosti Google a iOS od spoločnosti Apple.

Pri výbere operačného systému sme prihliadali na jeho obľúbenosť a z nej vyplývajúci podiel na trhu. V tomto rozmere sledovania vychádza ako jasný víťaz Android. Nájdeme ho na mobilných zariadeniach takmer každého výrobcu a takmer vždy aj na všetkých jeho modelových radách. [4]

Pomer užívateľov s iOS operačným systémom oproti užívateľom so systémom Android je dosť nepriaznivý, čo môže byť zapríčinené jeho vysokými nákladmi na hardvér, ktoré sú v porovnaní s konkurentom výrazne vyššie. Mnou vytvorenú aplikáciu by si tak mohlo zaobstarať o veľa menšie množstvo užívateľov. Tiež som bral do úvahy, že nemám takmer žiadne skúsenosti s týmto systémom.

Programovanie v systéme Android je výhodné aj po programátorskej stránke. V súčasnej dobe môžeme nájsť množstvo dobre spracovaných kníh, ktoré vysvetľujú fungovanie a možnosti programovania v tomto systéme. Tak isto je množstvo informácií aj na rôznych internetových portáloch, kde nájdeme kvalitné návody a riešené všetky druhy problémov, ktoré sa pri programovaní môžu vyskytnúť. V neposlednom rade aj spoločnosť Google ponúka dobre spracovanú kompletnú dokumentáciu s návodmi.

1.2.1 Bluetooth komunikácia

Robot Sphero komunikuje s ovládacím zariadením pomocou technológie bluetooth. V porovnaní s wifi má bluetooth pomalšiu komunikáciu, horšiu odozvu a aj v ďalších aspektoch bude asi ťahať za kratší koniec. Je však jedna zásadná výhoda tejto komunikácie a tou je spotreba energie.

Operačný systém Android má plnú podporu pre tento druh komunikácie a v dnešnej dobe má bluetooth asi každý smartfón či tablet. Pre vývojárov na platforme Androidu je v jeho aplikačnom frameworku zahrnuté API pre obsluhovanie tejto technológie, ktoré zahŕňa:

- vyhľadávanie iných zariadení,
- párovanie lokálneho zariadenia s inými zariadeniami,

- vytvorenie RFCOMM kanálov,
- pripojenie k iným zariadeniam,
- prenos dát z alebo do iných zariadení,
- správa viacerých pripojení. [5]

Pre prácu s bluetooth potrebujeme v súbore *manifest.xml* doplniť tag, ktorý nám dáva práva pre prácu s touto technológiou. [5]

```
<manifest ... >
  <uses-permission android:name="android.permission.BLUETOOTH" />
 ...
</manifest>
```

Obrázok 4 Získanie práv pre používanie bluetooth

V logike nejakej aktivity priradíme do premennej typu *BluetoothAdapter* adaptér nášho zariadenia a následne s ním už môžeme pracovať. [5]

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

Obrázok 5 Inicializácie bluetooth adaptéra v kóde

1.3 Robot Sphero

Sphero je guľový robot od spoločnosti Orbotix, ktorý sa ovláda za pomoci smartfónu alebo tabletu. Bol navrhnutý zakladateľmi spomínanej spoločnosti, ktorými je dvojica amerických inžinierov Ian Bernstein a Adam Wilson. Ich prvotným zámerom bolo vytvárať zariadenia ovládané prostredníctvom technológie bluetooth a predávať ich iným firmám ako polotovary. [6]

Robot Sphero bol pôvodne navrhnutý ako reklamné zariadenie ovládané pomocou smartfónu na propagáciu pred obchodnými partnermi. Stala sa však z neho obľúbená hračka s celkom dobrým úspechom na trhu. Jeho cena sa v USA pohybuje okolo 130$ a v Európe je to 115€. Má dobrú podporu aplikácií na Android aj iOS a ich počet stále stúpa, na koľko Orbotix vytvorilo API pre vývojárov na programovanie vlastných aplikácií. [6]

1.3.1 Stavba Sphera

Obrázok 6 Pohľad na Spheera z vonku aj z vnútra [7]

A) Nabíjacia stanica

Sphero počas nabíjania odpočíva v nabíjacej stanici, ktorá využíva indukčný systém na prenos elektrického prúdu do dvoch lítium polymérových batérií robota. Plné nabitie trvá asi 3 až 4 hodiny a poskytuje asi 75 minút jeho aktívneho hracieho času s robotom.

B) Kolesá

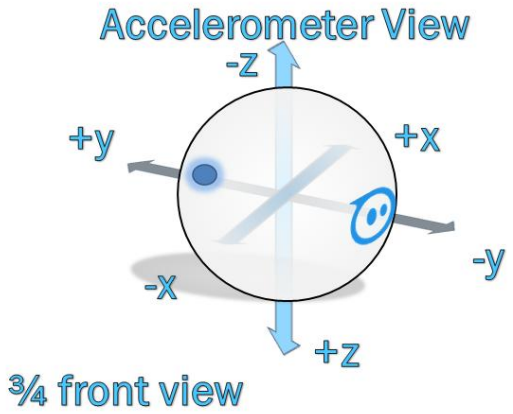
Dve kolesá nezávisle od seba poháňané dvomi motormi umožňujú otáčanie a posun v pred s maximálnou rýchlosťou 1,2 metra za sekundu. Kolesá sú na svojich okrajoch pogumované, aby zabránili prešmykovaniu s obalom. [7]

C) Horné ložisko

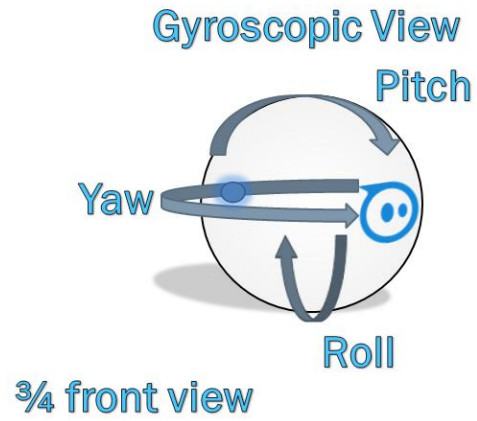
Vzhľadom na to, že vnútro robota sa môže v plastovom obale voľne pohybovať je potrebné vyplniť prázdny priestor týmto ložiskom, aby vnútro robota „nelietalo“ a aby boli kolesá v neustálom kontakte s obalom. [7]

D) Plošné spoje

Procesor zberá dáta z troch ôs akcelerometra a dáta z gyroskopu a z nich následne vypočítava presné otočenie robota. Tieto výpočty sú potrebné pre správne reakcie robota na ovládanie. [7]



Obrázok 7 Pohľad akcelerometra[9]



Obrázok 8 Pohľad gyroskopu[9]

E) Bluetooth vysielač a prijímač

Tento systém zabezpečuje Spherovi bezdrôtovú komunikáciu do vzdialenosti asi 20 až 30 metrov v ideálnych podmienkach. [7]

F) Farebné LED

Robot obsahuje jednu trojicu farebných LED (červená, modrá, zelená) pre namiešanie celého spektra rôznych farieb, ktoré rozžiaria obal robota. Nájdeme tu však ešte jednu modrú LED, ktorá slúži na kalibráciu smeru robota. [7]

1.3.2 Programovanie Sphera

Spoločnosť Orbotix pripravila pre vývojárov dobre spracované SDK, ktoré sú prístupné na stiahnutie na adrese *https://github.com/orbotix/Sphero-Android-SDK*. Obsahom celého balíka je knižnica v jazyku Java pre obsluhu robota, kompletná dokumentácia všetkých jej funkcií a tried a niekoľko jednoduchých aplikácií aj s krátkym popisom a návodom na použitie každej technológie a konceptu.

Ako bolo spomínané Sphero komunikuje prostredníctvom bluetooth, a teda v aplikácií budeme potrebovať povolenia používať ho a zároveň vystupovať aj v pozícií administrátora tohto spojenia.

```
<manifest ... >
  <uses-permission android:name="android.permission.BLUETOOTH" />
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
  ...
</manifest>
```

Obrázok 9 Získanie práv pre používanie bluetooth administrátora

Sphero Android API obsahuje kompletný komponent starajúci sa o pripojenie robota k ovládaciemu zariadeniu. Ukáže nám zoznam zapnutých robotov pripravených k pripojeniu. Kliknutím na názov robota sa vytvorí spojenie. Tento komponent stačí jednoducho nasledovným spôsobom pridať do nejakej aktivity v našej aplikácii a o pripojenie máme postarané.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#ff888888" >

    <orbotix.view.connection.SpheroConnectionView
        android:id="@+id/sphero_connection_view"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="#FFF" />

</LinearLayout>
```

Obrázok 10 Doplnenie pripájacieho elementu do layoutu

V prípade úspešného pripojenia sa nám robot priradí do premennej a môžeme s ním plne pracovať. V prípade neúspechu pripojenia sa nám o tom objaví správa a opätovne zoznam na výber a možnosť pripojenia robota.

```
mSpheroConnectionView = (SpheroConnectionView) findViewById(R.id.sphero_connection_view);
mSpheroConnectionView.addConnectionListener(new ConnectionListener() {
        @Override
        public void onConnected(Robot robot) {
                sphero = (Sphero) robot;
        }
        @Override
        public void onConnectionFailed(Robot sphero) {
                // pripojenie zlyhalo ...
        }
        @Override
        public void onDisconnected(Robot sphero) {
                mSpheroConnectionView.startDiscovery(); //spustime vyhladavanie zariadeni
        }
});
```

Obrázok 11 Inicializácia robota v logike aplikácie

Teraz máme viacero možností ako robotovi budeme servírovať inštrukcie a teda ho ovládať. Môžeme posielať samostatné príkazy, vytvárať makrá alebo robotovi posielať inštrukcie v OrbBasic jazyku. Je to jednoduchý jazyk vytvorený pre interpretáciu príkazov robotovi. Je to najsurovejšia forma ovládania.

1.4 OrbBasic

OrbBasic je vzhľadom na možnosti robota dosť dobre obsiahnutý jazyk a dokážeme v ňom vytvoriť celkom zložité programy. Jeho skladba je jednoduchá skladá sa zo sekvencie programových riadkov, kde je na každom riadku samostatný príkaz. Riadky začínajú ich číselným označením za ním nesleduje medzera, ďalej príkaz a na konci špeciálny znak označujúci koniec riadku. [8]

Obsahuje celkovo 50 premenných, kde do 25 z nich je možnosť aj zapisovať. Ostatných 25 premenných je iba na čítanie a sú v nich uložené hodnoty senzorov robota. Všetky premenné sú celočíselného typu *Integer* siahajúceho od -2,147,483,647 do +2,147,483,647. V nasledujúcej tabuľke je ich prehľad. [8]

Tabuľka 1 Prehľad premenných jazyka OrbBasic

Názov	Prístup	Veľkost	Popis
a ... y	r/w	32-bit signed value	bežné premenné na zapisovanie a čítanie
timerA, timerB, timerC	r/w	16-bit positive value	časovače
ctrl	r/w	1-bit value	stav kontrolného systému
speed	r	8-bit positive value	rýchlosť
yaw	r	16-bit positive value	otočenie robota yaw
pitch	r	16-bit positive value	otočenie robota pitch
roll	r	16-bit positive value	otočenie robota roll
accelX, accelY, accelZ	r	16-bit signed value	akcelerometer v osiach
gyroX, gyroY, gyroZ	r	16-bit signed value	gyroskop v osiach
Vbatt	r	16-bit unsigned value	napätia batérie
Sbatt	r	16-bit unsigned value	stav batérie
cmdroll	r	Boolean	detekcia roll príkazu
spdval	r	8-bit positive value	posledná rýchlosť
hdgval	r	16-bit positive value	posledné otočenie
cmdrgb	r	Boolean	detekcia RGB príkazu
redval, grnval, bluval	r	8-bit positive value	posledné farby
isconn	r	Boolean	stav pripojenia robota
dshake	r	Boolean	detekcia dvojitého zatrasenia
accelone	r	16-bit positive value	akceleračný vektor
xpos, ypos	r	16-bit signed value	pozícia
Qzero, Qone, Qtwo, Qthree	r	16-bit signed value	quaternion

Cykly definujeme tromi hodnotami. Počiatočnú hodnotu počítadla, konečnú hodnotu počítadla a veľkosť kroku počítadla, čo znamená po koľko sa bude k počítadlu pripočítavať. Cyklus je zo spodnej časti ohraničený volaním funkcie *next* na premennú počítadla. Je možné ich aj vnárať do maximálnej hĺbky tretej úrovne. Každá úroveň má svoju vyhradenú premennú počítadla, kde prvá má *X*, druhá *Y* a tretia *Z*. [8]

Ďalej v ňom dokážeme plnohodnotne tvoriť podmienky, kde je však jedno obmedzenie a to, že do každej z vetiev dokážeme dať iba jeden príkaz. Toto sa ale dá v celku pohodlne riešiť skákaním po riadkoch kódu za pomoci funkcie *goto*. Vetvu je teda možné definovať napríklad na konci programu, kde v podmienke skočíme a následne sa po jej vykonaní vrátime za podmienku. [8]

Pri práci s premennými a dosadzovaním hodnôt do funkcií, cyklov a podmienok môžeme využívať aj výrazy. Ich prehľad je v nasledujúcej tabuľke.

Tabuľka 2 Prehľad operátorov jazyka OrbBasic

Operátor	Priorita	Popis
(	vysoká	začína nový podvýraz
)	vysoká	ukončuje podvýraz
*****	medium	celočíselné násobenie
/	medium	celočíselné delenie
%	medium	zvyšok po celočíselnom delení
{	medium	binárny posun doľava
}	medium	binárny posun doprava
+	low	sčítanie
-	low	odčítanie
&	low	bitový AND
|	low	bitový OR

OrbBasic ďalej disponuje funkciami programovými, matematickými a na ovládanie robota. Matematické môžeme opäť používať pri práci s premennými alebo hodnotami. Prehľad všetkých funkcií je v nasledujúcej tabuľke. [8]

Tabuľka 3 Prehľad funkcií jazyka OrbBasic

Funkcia	Prístup	Veľkost	Popis
sqrt *x*	r	32-bit positive value	odmocnina
rnd *x*	r	32-bit positive value	náhodná hodnota
abs *x*	r	32-bit positive value	absolútna hodnota
print	-	-	vypísanie správy
delay *x*	param	16-bit positive value	čakanie programu
end	-	-	ukončenie programu
RGB *r,g,b*	param	8-bit positive values	farba osvietenia
LEDC *x*	param	8-bit positive value	farba osvietenia
backLED *x*	param	8-bit positive value	zadná LED
goroll *h,s,g*	param	16,8,8-bit positive value	pohyb robota
random	-	-	spustenie rand. generátora
heading *h*	param	16-bit positive value	otočenie robota
locate *x,y*	param	32-bit signed value	nastavenie polohy
data *d1{,d2,d3…}*	param	32-bit signed values	množina
rstr	-	-	číta hodnotu z množiny
read *X{,Y…}*		32-bit signed values	číta hodnotu z množiny
tron	-	-	zapnutie sledovania riadkov
troff	-	-	vypnutie sledovania riadkov
reset	-	-	znova spúšťa program
sleep *duration, macro, line_number*	param	16,8,32-bit positive value	uspanie robota
macrun *x*	param	8-bit unsigned value	spúšťa makro
mackill	-		zastavuje makro

macstat	r	32-bit unsigned value	vracia stav vykonávania makra

Nevýhodou OrbBasic-u však je, že v ňom nedokážeme definovať paralelne sa vykonávajúce programové vetvy.

Dôležité funkcie OrbBasic jazyka:

print (print "Hello world") – Pomocou tejto funkcie sa dajú vypisovať z programu správy. Samotný robot správu nemá ako zobraziť, ale vie ju poslať do zariadenia, s ktorým komunikuje. [8]

delay (delay 20) – Funkcia slúžiaca na čakanie. Parameter definuje čas v milisekundách. Po vypršaní času program pokračuje nasledujúcim príkazom. [8]

end – Ukončenie celého programu. [8]

RGB (RGB 255, 255, 255) – Funkcia na zmenu farby osvietenia robota. Na vstupe má tri parametre, číselné hodnoty od 0 – 255, ktoré reprezentujú intenzitu svietenia troch farebných (červená, zelená, modrá) LED diód robota. [8]

goroll (goroll 0, 100, 1) – Táto funkcia prikazuje robotovi, aby sa hýbal. Prvý parameter je smer, číslo v rozmedzí 0 – 359 a určuje smer, akým sa robot hýbe vzhľadom na vykalibrované natočenie. Druhý parameter je rýchlosť, číslo v rozmedzí 0 – 255. Posledný parameter môže mať tri číselné hodnoty. Hodnota 2 prikazuje robotovi rýchly pohyb, 1 normálny pohyb. Ak je definovaná hodnota 0 robot zastavuje. [8]

heading (heading 100) – Otáčanie robota. Parametrom je číslo v rozmedzí 0 – 359 reprezentujúce uhol. [8]

goto (goto 50) – Funkcia, ktorá neovláda robota, ale čítanie kódu. Pomocou nej dokážeme skákať po riadkoch programu. Vstupný parameter je číslo určujúce riadok, kam chcem preskočiť. [8]

gosub (gosub 50) – Rovnaká funkcia ako goto s tým rozdielom, že táto funkcia čaká na funkciu return. Po jej prečítaní sa čítanie vracia na miesto, odkiaľ preskočilo. Pomocou tejto funkcie sa dajú vytvárať časti kódu ako procedúry. [8]

```
10 ctrl = 0
20 LEDC 8
30 X = 255 * ((yaw/45) & 1)
40 backLED X
50 goto 30
```

Obrázok 112 Ukážka OrbBasic jazyka

2 Návrh riešenia

2.1 Aplikácia

Vytvorený programovací jazyk by mal byť vložený do užívateľsky priateľskej aplikácie, ktorá pobeží na mobilnom zariadení. Takisto je potrebná aj jednoduchosť jej ovládania. Programovať v tejto aplikácií by mala byť zábava hlavne vzhľadom na jej cieľovú skupinu, deti na základnej škole.

Rozloženie prvkov aplikácie by malo byť nasledovné. Na vrchu bude aplikačná lišta, na ktorej bude logo a názov aplikácie. Na spodnej časti bude umiestnená lišta, na ktorej budú tlačidlá jednotlivých akcií v poradí: pripojenie na robota, odoslanie kódu do robota, zrušenie aktuálne vykonávaných inštrukcií a tlačidlo rozbaľovacieho menu. V tomto menu sa budú nachádzať akcie vytvorenia nového programu, uloženia aktuálneho programu, uložené programy a pomoc, kde bude popísaná aplikácia a premenné senzorov. V zozname uložených programov sa nám po vybratí jedného z nich ponúkne možnosť ho načítať alebo zmazať. Všetkým zásadným akciám, pri ktorých sa stráca aktuálny program alebo uložený program by mala predchádzať akcia potvrdenia, či si je užívateľ istý svojím konaním. Spodná aj horná lišta by mali byť sivej farby aby nepôsobili rušivo voči farebnému kódu.

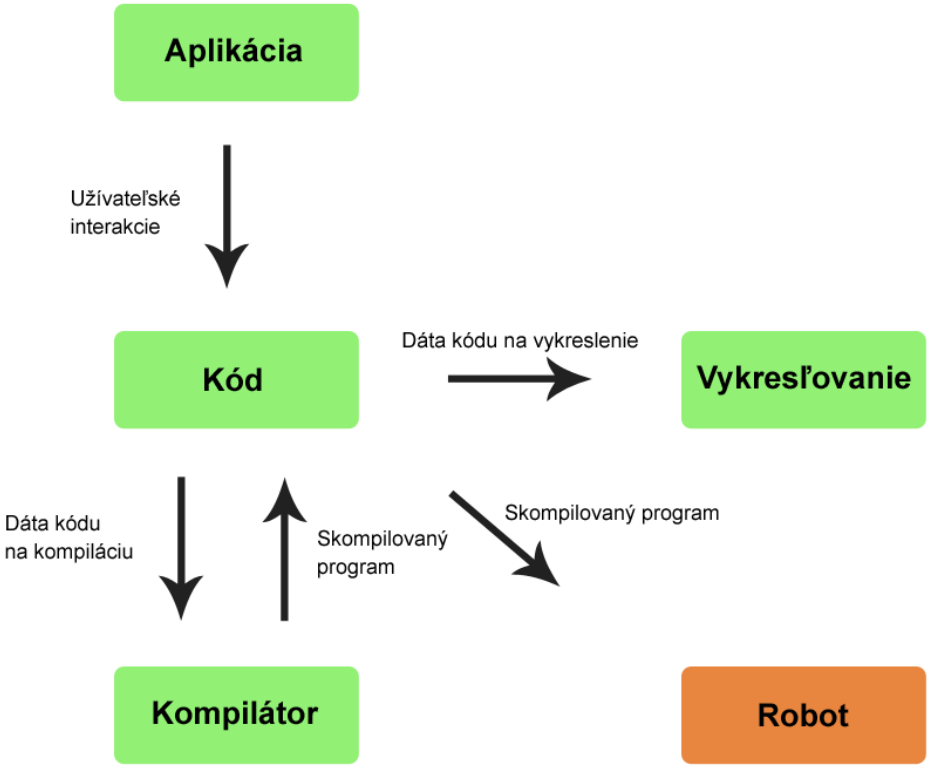
Na celej zvyšnej ploche aplikácie by malo byť biele pracovné plátno. Na ňom sa bude zobrazovať kód a pracovať s ním. Po tomto plátne by sa malo dať posúvať ťahaním jedného dotyku. Približovaním a odďaľovaním dvoch dotykov súčasne sa bude plocha približovať a odďaľovať. Po kliknutí na tehličku kódu sa zobrazí menu, v ktorom bude na výber pridanie novej tehličky a editovanie tehličky, na ktorú sme klikli alebo jej zmazanie. Editovanie a vytváranie sa bude uskutočňovať cez rovnaký dialóg, v ktorom bude mať každý vstupný parameter svoj vstup, do ktorého sa zadá. Po zadaní parametrov sa dialóg potvrdí a potom sám zavrie. Zmeny v kóde sa okamžite zobrazia po ich vykonaní.

Pripájanie robota bude prebiehať v samostatnej obrazovke, kde sa nám zobrazí zoznam nájdených robotov. Po kliknutí na jedného z nich sa vytvorí spojenie a automaticky nás to vráti na obrazovku vytvárania kódu.

O všetkých zásadných akciách by mal byť užívateľ informovaný prostredníctvom krátkych správ na spodnej strane obrazovky.

2.2 Jazyk

Navrhovaný jazyk musí hlavne brať do úvahy zariadenie, na ktorom sa s ním bude pracovať nakoľko jeho možnosti sú obmedzené. Taktiež musí byť vytvorený v rámci možností robota a jeho jazyka. Na nasledujúcom diagrame je zobrazená komunikácia a tok dát jednotlivých častí aplikácie a robota.

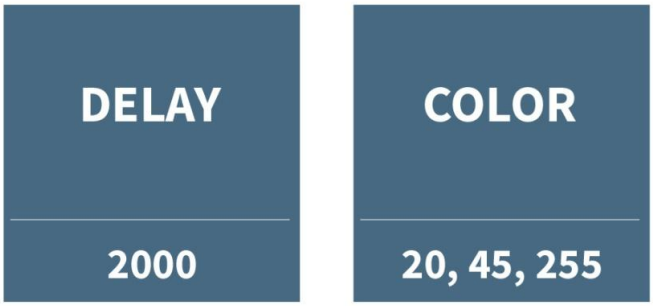


Obrázok 12 Diagram komunikácie a toku dát

2.2.1 Grafický návrh

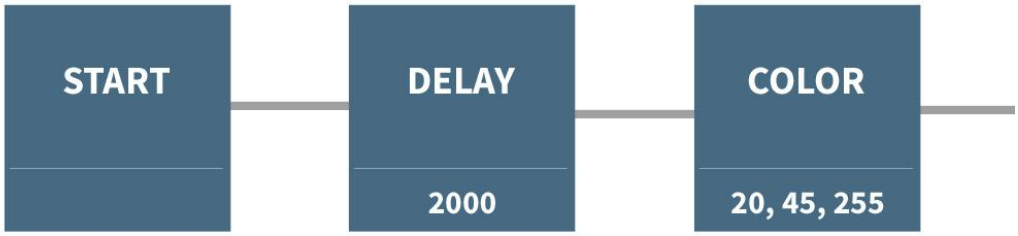
Pri návrhu celkového grafického riešenia musíme prihliadať na to, že aplikácia bude pre mobilné zariadenia, ktoré disponujú v porovnaní s počítačmi podstatne menšími displejmi a teda menšou plochou na realizáciu. Je teda potrebná prehľadnosť, kompaktnosť a spomínaná jednoduchosť.

Najvhodnejším riešením pre reprezentáciu jednotlivých príkazov bude farebný štvorec v strede ktorého bude názov funkcie, ktorú reprezentuje. Viditeľné musia byť aj vstupné premenné k funkciám. Tie by mohli byť vykresľované na spodnej časti štvorca reprezentujúceho príkaz.



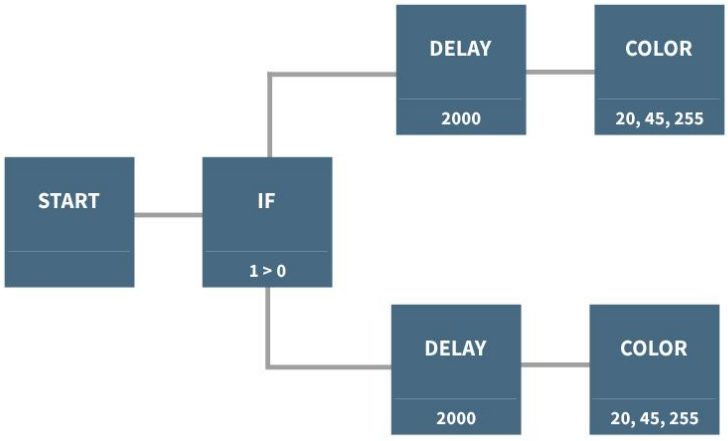
Obrázok 13 Grafický návrh - príkaz

Príkazy musia byť graficky prepojené vláknom, aby bolo jasné ako program nasleduje a kde sa ako vetví.



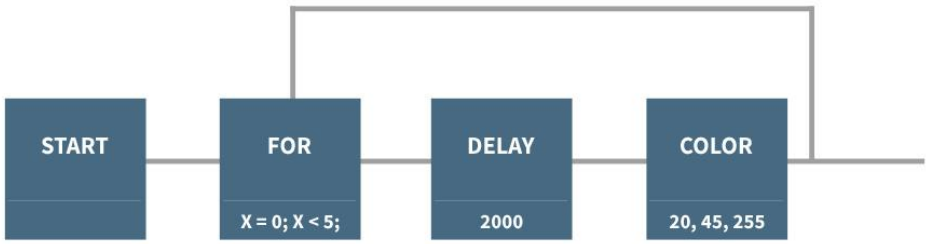
Obrázok 14 grafický návrh – sekvencia

Vetvy by mali byť vykresľované vedľa seba alebo pod seba.



Obrázok 15 Grafický návrh - vetvenie

Cyklus by mal tak isto zobrazovať akú časť kódu ohraničuje.



Obrázok 16 grafický návrh – cyklus

2.2.2 Funkcie

Priblížime si, aké funkcie by náš jazyk mohol mať vzhľadom na možnosti robota a zvolenú formu jeho ovládania.

ROLL() – Funkcia prikazujúca robotovi hýbať sa vpred až pokým nedostane príkaz na zastavenie.

STOP() – Funkcia prikazujúca robotovi zastaviť.

TURN(angle) – Príkaz na otočenie robota. Má jeden vstupný parameter, číslo v rozmedzí 0 – 359, ktorý udáva uhol na aký sa má robot natočiť.

SPEED(speed) – Nastavenie rýchlosti robota. Vstupný parameter je číslo v rozmedzí 0 – 255.

COLOR(r, g, b) – Funkcia pomocou, ktorej robotovi nastavíme farbu osvietenia. Má tri vstupné parametre ktoré môžu mať hodnoty 0 – 255. Každý parameter reprezentuje jednu farebnú(červená, zelená, modrá) diódu a nastavuje jej intenzitu svietenia.

LED(intensity) – Funkcia nastavujúca intenzitu svietenia zadnej LED diódy. Má jeden vstupný parameter nadobúdajúci hodnoty 0 – 255.

DELAY(time) – Uspatie behu programu na zadaný čas vstupným parametrom. Čas sa zadáva v milisekundách.

PRINT(message) – Funkcia na vypísanie správy na zariadení ovládajúcom robota. Vstupný parameter je textový reťazec, ktorý chceme vypísať.

SOUND() – Prehratie notifikačného zvuku na zariadení ovládajúcom robota.

X = value – Priradenie hodnoty do premennej.

IF condition THEN ELSE– Podmienka alebo vetvenie programu.

LOOP X = value to value step value ... next X – Cyklické opakovanie časti programu. Dokážeme nastaviť odkiaľ pokiaľ má cyklus bežať a vieme určiť aj veľkosť jeho kroku.

WHILE condition – Cyklus opakujúci sa pokým platí podmienka zadaná vstupným parametrom.

2.2.3 Dátová reprezentácia

Základnou implementačnou časťou celého riešenia bude dátová reprezentácia jednotlivých funkcií (tehličiek) jazyka. Pri návrhu tejto štruktúry je potrebné myslieť na to, že jednotlivé funkcie jazyka majú rôzny počet vstupných parametrov a potrebujú mať uložené rôzne dáta. Preto treba definovať základnú jednoduchú triedu od ktorej sa budú následne rozširovať ďalšie zložitejšie triedy doplnené o potrebnú štruktúru a funkcionalitu pre reprezentáciu jednotlivých funkcií jazyka.

Už v základnej triede bude musieť byť uložená informácia, o aký typ funkcie alebo tehličky ide. Môžu tu byť aj definované jednotlivé statické konštanty definujúce tieto typy.

Inštancie týchto tehličiek kódu postačí uložiť do utriedeného zoznamu. Jednotlivé časti budeme potom už len pridávať na presné miesto v tomto zozname a bude jednoduché nimi prechádzať v reálnom poradí pri operáciách s nimi.

Časti ako cyklus a podmienka sa budú skladať z viacerých tehličiek. Napríklad cyklus bude mať svoju hlavnú tehličku, kde budú uložené všetky potrebné dáta o cykle a v nej bude uložený aj smerník na jednoduchú tehličku, ktorá uzatvára cyklus. Táto uzatváracia tehlička bude mať tak isto svoje presné miesto v kóde respektíve v spomínanom usporiadanom zozname.

Jednotlivé parametre funkcií jazyka môžu byť typu *String* nakoľko celý prekompilovaný kód odosielaný do robota bude tak isto len textový reťazec tohto typu.

V rozšírených odvodených triedach budú definované aj funkcie starajúce sa o vykresľovanie tehličky daného typu a aj funkcie starajúce sa o vytváranie tehličky. Ich umiestnenie je tu na mieste z dôvodu, že každý rozšírený typ sa bude v týchto jeho častiach mierne odlišovať. Ak funkcia obsahuje nejaké vstupné parametre budú sa musieť zadávať pri jej vytváraní. Celý proces vytvárania takejto funkcie bude implementovaný pomocou *AlertDialog*-u, v ktorom zadáme všetky vstupné parametre a potvrdíme vytvorenie.

2.3 Kompilátor

Robotovi budeme interpretovať naše inštrukcie prostredníctvom OrbBasic jazyka. V porovnaní s makrami je to zložitejšie riešenie, ale na druhej strane s ním dosiahneme viac. V makre vieme dokonca definovať aj cyklus no nedokážeme definovať podmienku čo je dosť podstatná súčasť programovacieho jazyka. V OrbBasic-u to však dokážeme aj preto budeme v našom riešení interpretovať inštrukcie pre robota touto formou.

2.3.1 Kompilácia do OrbBasic

V aplikácií bude potrebné štruktúru grafického kódu prekompilovať do spomínaného OrbBasic jazyka robota. Riešenie tohto problému nie je až tak zložité. Každá funkcia vytváraného grafického jazyka má aj reálnu reprezentáciu v OrbBasic jazyku. Prejdeme teda celý náš grafický kód a jednotlivé funkcie zaradom ako idú, vpíšeme do programu pre robota. Problém však nastáva pri podmienkach. Ako bolo už spomínané je potrebné ich riešiť pomocou skokov *gosub* alebo *goto*. Každá vetva programu bude teda definovaná ako by vo svojej funkcií. Všetky vetvy musíme potom presunúť na koniec programu pre robota kam budeme počas behu programu skákať a následne sa vracať.

2.4 Vykresľovanie

Na vykresľovanie kódu budeme používať grafické plátno, ktoré je v Androide reprezentované triedou *Canvas*. Pre naše riešenie je to najlepšia možnosť ako vykresliť v podstate akúkoľvek grafiku.

Jednotlivé tehličky kódu by mali mať jednoduchý dizajn, aby bol kód dobre čitateľný. Na jednoduchú grafiku nebudeme potrebovať ani žiadne textúry a v podstate zjednodušene povedané budeme iba vykresľovať farebné obdĺžniky. Čím bude grafika jednoduchšie riešená, tým bude mať menšie nároky a výsledná aplikácia bude prístupnejšia pre väčšie spektrum zariadení.

Na vykresľovanie by mala byť implementovaná samostatná trieda, ktorá sa bude starať iba o túto časť programu. V nej musíme myslieť na možnosť presúvania sa po grafickom plátne, možnosť približovania a odďaľovania kódu a možnosť odchytávať udalosti kliknutia na jednotlivé tehličky.

Z hľadiska logickej štruktúry programu by samotná funkcia vykreslenia tehličky mala byť definovaná v jej triede. Pri vykresľovaní budeme potom prechádzať celý kód a volať vykreslenie na samotných tehličkách.

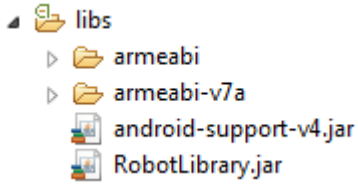
Takéto riešenie je potrebné preto, lebo každý rozšírený typ tehličky bude mať aj o niečo inú grafickú reprezentáciu a tú je dobré si definovať v týchto rozšírených typoch samostatne.

3 Implementácia

3.1 Štruktúra aplikácie

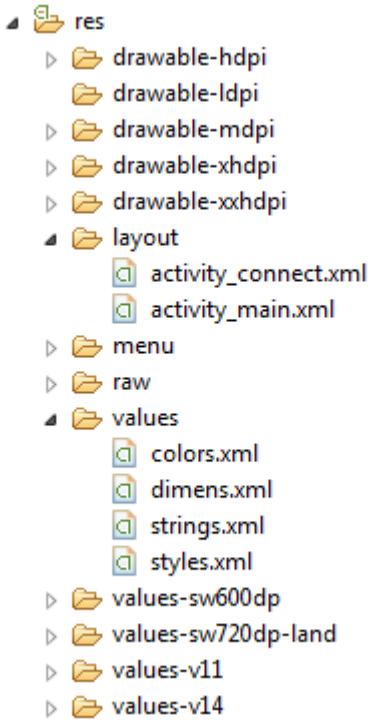
Vytvorená aplikácia má open-source licenciu a je možné si jej zdrojový kód stiahnuť na adrese https://code.google.com/p/baby-orb-basic/. Je postavená na Android aplikačnom framework-u a teda jej základná štruktúra je už jasne daná.

Knižnice pomocou, ktorých komunikujeme s robotom a ovládame ho sú uložené v priečinku */libs*.



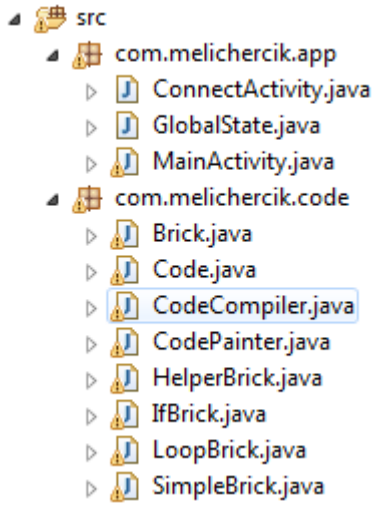
Obrázok 17 štruktúra priečinku /libs

Úložiskom preddefinovaných hodnôt, layout-ov, menu, obrázkov a iných zdrojov je priečinok */res*.



Obrázok 18 štruktúra priečinku /res

Programovaná logika našej aplikácie sa nachádza v priečinku */src* a je rozdelená do dvoch balíkov.



Obrázok 19 Štruktúra priečinku /src

3.1.1 Balík com.melichercik.app

V balíku *com.melichercik.app* sa nachádzajú triedy jednotlivých aktivít a aplikácie rozšírené o potrebnú logiku.

Trieda *GlobalState* rozširuje hlavnú aplikačnú triedu *Aplication* o premennú *robot*. Táto premenná sa tu nachádza preto, aby bola globálne definovaná a viditeľná pre všetky aplikačné aktivity, nakoľko je pripájanie robota a vytváranie kódu oddelené každé vo svojej aktivite. Ich prepínaním by sa dáta o spojení s robotom strácali. V tejto premennej je teda uložené prepojenie na robota a jej prostredníctvom sa vykonáva všetka komunikácia s ním.

O logiku aktivity pripájania na robota sa stará trieda *ConnectActivity*. Súčasťou tejto aktivity je aj *SpheroConnectionView*. Je to element starajúci sa o vyhľadávanie a nadviazanie spojenia s robotom.

Hlavnú aktivitu práce s kódom obstaráva trieda *MainActivity*. V nej sa inicializujú všetky metódy, ktoré pracujú s kódom. Jej súčasťou je aj trieda *RenderView*, ktorá implementuje *Runnable*, aby bolo možné spustiť jej vlákno starajúce sa o neustále vykresľovanie grafickej plochy.

3.1.2 Balík com.melichercik.code

Balík *com.melichercik.code* obsahuje triedy týkajúce sa kódu, jeho samotnej dátovej štruktúry, vytvárania, editovania, vykresľovania a kompilácie.

Hlavnou logickou triedou tohto balíka je trieda *Code*. Spravuje všetky ostatné triedy v tomto balíku a všetky volania týkajúce sa kódu v hlavnej aktivite *MainActivity*, až na editáciu častí sa vykonávajú na inštanciu tejto triedy. V tejto triede sú v uložené všetky tehličky kódu, teda celá jeho dátová štruktúra v usporiadanom zozname. Tento typ ich uloženia bol najvhodnejší, nakoľko dáta sú v ňom uložené v reálnom poradí, ako sa zobrazujú aj v grafickom kóde a pri operáciách je jednoduché nimi v tomto poradí aj listovať.

Hlavnou triedou dátovej štruktúry je trieda *Brick*, ktorá reprezentuje tehličku nášho grafického kódu. Sú v nej definované konštanty všetkých typov tehličiek a ako vytvorená inštancia má v sebe informáciu o aký typ tehličky ide. Ďalej sú v nej definované pomocné premenné x-ovej a y-ovej osi pomáhajúce pri vykresľovaní a detekcii kliknutia na tehličku ako aj samotná metóda zisťovania spomínaného kliknutia. Od tejto základnej triedy sú rozšírené ďalšie štyri, ktorých hlavnou odlišnosťou sú uchovávajúce dáta.

Jednou z nich je trieda *SimpleBrick*. Uchováva v sebe maximálne tri vstupné premenné reprezentujúcej funkcie grafického kódu. Ostatné triedy sa líšia hlavne tým, že majú v sebe uložené aj iné tehličky, ktoré sú im pomocné a teda sú s nimi späté. Napríklad tehlička cyklu potrebuje odkazovať aj na tehličku, ktorá cyklus ohraničuje. Takéto pomocné tehličky má uložené aj podmienka, aby vedela ohraničovať then – vetvu a else – vetvu. Všetky tieto triedy sú rozšírené aj o funkcie vykresľovania a s výnimkou *HelperBrick* aj o funkciu vytvárania so zadávaním vstupných premenných v *AlertDialog*-u.

Vykresľovanie zastrešuje trieda *CodePainter*, ktorej hlavnou podstatou je prelistovanie celého kódu, kde sa na jednotlivé tehličky volajú metódy vykreslenia.

Poslednou triedou z tohto balíku je *CodeCompiler*. V tejto triede sa prechádza celý grafický kód a následne sa z neho vytvára OrbBasic kód, ktorému rozumie robot. Nakoľko OrbBasic v podmienkach nemôže obsahovať viac ako jeden príkaz, podstatnou časťou kompilácie je prehodenie kódu z vetiev podmienok na koniec kompilovaného kódu pre robota. Do podmienok sa následne pridajú príkazy *goto*, ktoré čítanie programu odkazujú na daný riadok, kde je vetva uložená. Na konci týchto vetiev sa pomocou tohto istého príkazu presúva čítanie programu na riadok za podmienkou, z ktorej sme skákali. Potom sa už len prepíšu všetky tehličky na reálne príkazy z jazyka OrbBasic, ktoré vlastne naše grafické tehličky reprezentujú.

3.2 Grafika jazyka

Celkový grafický dizajn jazyka je čistý a veľmi jednoduchý. Pri jeho tvorbe sa dbalo hlavne na tieto dve jeho vlastnosti vzhľadom na to, že je cielený na zariadenia s malým displejom.

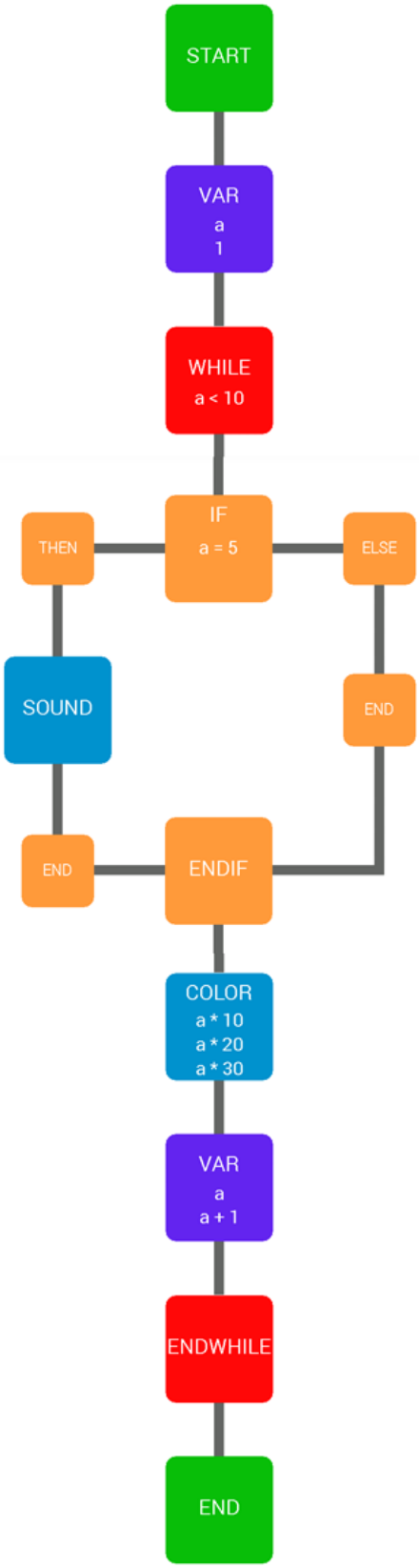
Pracovné plátno je bielej farby aby na ňom dobre vynikali všetky farebné kombinácie funkcií.

Funkcie jazyka sú reprezentované farebnými "tehličkami", štvorčekmi so zaoblenými rohmi. Logicky sú funkcie farebne odlíšené, aby sa v kóde dalo už na prvý pohľad lepšie orientovať. Na každej tehličke je kapitálkami napísaný názov funkcie, ktorú reprezentuje. Ak má funkcia aj nejaké vstupné premenné, sú napísané na tehličke pod názvom, pričom ak je premenných viac, vždy je každá ďalšia napísaná na novom riadku. Ak je názov vstupnej premennej dlhší a teda by sa nezmestil na šírku tehličky, text je odrezaný na potrebnú dĺžku a doplnený tromi bodkami, aby bolo jasné, že nie je zobrazená kompletná informácia. Pre jej zobrazenie stačí otvoriť editáciu (detail) danej funkcie.

Aby bol jasný sled programu, každá tehlička je prepojená vláknom sivej farby na nasledujúcu.

V prípade podmienok sa program aj graficky rozvetvuje, kde je na ľavej strane then vetva, teda vetva kam program postupuje pri splnení zadanej podmienky. Pravá vetva reprezentuje else, a teda program sem putuje v prípade, kedy podmienka splnená nebola. Začiatok a koniec vetiev je tak isto reprezentovaný tehličkami, ktoré sú však menšie, aby zbytočne neprehusťovali kód a aby nepôsobili tak podstatne ako ostatné tehličky, nakoľko tieto sú len pomocné k funkcií podmienky.

V prípade cyklov by nemalo zmysel pridávať ďalšie vlákno, ktoré by spájalo jeho začiatok a koniec po strane. Pretože ak by bolo v cykle vnorených viac podmienok, prepájajúce vlákno by sa muselo veľmi natiahnuť aby obišlo celý rozšírený kód a strácalo by tým svoj ilustračný zmysel. Cykly teda nie sú nijak špeciálne prepojené ale ich tehličky majú výraznú červenú farbu aby boli na prvý pohľad viditeľnejšie.



Obrázok 20 Ukážka programu vytvoreného grafického jazyka

Na obrázku je jednoduchý program kde sa mení farba osvietenia robota podľa hodnoty premennej *a*. Ak premenná a nadobúda hodnotu 5 zaznie notifikačný zvuk.

3.3 Funkcie jazyka

3.3.1 Start

Táto funkcia v našom grafickom jazyku reprezentuje štartovací bod a je na začiatku každého vytváraného programu. Nie je možné ju mazať ani nijak upravovať. Slúži len ako stavebný kameň, na ktorý je možné napájať iné funkcie. Je reprezentovaná tehličkou zelenej farby s nápisom START.

V dátovej štruktúre je typu *SimpleBric* aj keď neuchováva žiadnu premennú ale je potrebné ju vykresľovať a preto ju ukladáme pod týmto rozšíreným typom, ktorý túto možnosť podporuje. Pri kompilácií sa táto funkcia do OrbBasic jazyka vôbec neprekladá.

3.3.2 End

Funkcia ukončujúca beh celého programu. Nie je možné na nej vykonávať žiadnu akciu. Reprezentuje ju zelená tehlička s nápisom END.

Svoj význam nadobúda hlavne pri kompilácií a v preloženom jazyku pre robota. Keď pri kompilácií vytvárame vetvy programu umiestňované na jeho koniec, je potrebné, aby sa do nich prechádzalo iba skokmi funkcie g*osub*. Umiestnením funkcie End pred ich začiatok, a teda na koniec normálneho behu programu, zamedzíme tomu, aby sem dostalo čítanie kódu normálnym sledom programu.

3.3.3 Roll

Funkcia, ktorá dáva robota do pohybu až pokým nedostane príkaz Stop na zastavenie. Má dva parametre - smer a rýchlosť. Smer je hodnota v rozmedzí 0 – 359 a reprezentuje otočenie robota vzhľadom na jeho vykalibrovanú polohu. Rýchlosť je hodnota v rozmedzí 0 – 255. Funkcia je reprezentovaná modrou tehličkou a nápisom ROLL.

V dátovej štruktúre ju reprezentuje trieda *SimpleBrick*. Pri kompilácií sa prekladá do funkcie *goroll h,s,g*. Za parameter g sa dosadzuje hodnota 1, čo znamená normálny chod robota.

3.3.4 Stop

Funkcia slúžiaca na zastavenie pohybu robota. Nemá žiadny vstupný parameter. Reprezentuje ju tehlička modrej farby s nápisom STOP.

V dátovej štruktúre je typu *SimpleBrick*, aby ju bolo možné pomocou funkcií tejto rozšírenej triedy vykresľovať. V kompilátore sa prekladá do rovnakej funkcie ako Roll ale za parameter *s* reprezentujúci rýchlosť sa dosadzuje 0 a parameter *g* 2 čo znamená, že robot brzdí.

3.3.5 Color

Funkcia pomocou ktorej ovládame svetelné zafarbenie robota. Má tri vstupné parametre - red, green a blue, ktoré môžu nadobúdať hodnoty v rozmedzí 0 – 255. Každý z nich reprezentuje jednu z troch farebných led diód robota (červená, zelená, modrá) a ovláda tým intenzitu jej svietenia. V grafickom jazyku ju reprezentuje modrá tehlička s nápisom COLOR.

Typ *SimpleBrick* ju reprezentuje v dátovej štruktúre. Pri kompilácií do jazyka robota sa prekladá do funkcie *RGB r,g,b*.

3.3.6 Led

Funkcia, ktorou ovládame intenzitu modrej zadnej LED diódy robota. Má jeden vstupný parameter a to číslo v rozmedzí 0 – 255 znamenajúci intenzitu svietenia. Reprezentuje ju modrá tehlička s nápisom LED.

V dátovej štruktúre je reprezentovaná triedou *SimpleBrick*. Kompilátor ju prekladá do funkcie *backLED x*.

3.3.7 Delay

Funkcia pozastavujúca vykonávanie kódu, uspí program na zadaný čas. Na vstupe dostáva jeden parameter - dĺžku trvania tohto uspatia zadanú v milisekundách. V grafikom kóde ju nájdeme ako modrú tehličku s nápisom DELAY.

Je definovaná typom *SimpleBrick* v dátovej štruktúre. Kompilátorom je prekladaná do funkcie *delay x.*

3.3.8 Print

Funkcia na vypisovanie správ pre užívateľa. Na vstupe dostáva jeden parameter a to správu, ktorú má vypísať. Nie je možné vypisovať žiadnu premennú iba zadaný text. Pri inicializácií robota sa nastavuje aj *EventListener*, ktorý čaká na správy od robota v reálnom čase počas behu programu. Správa sa po prijatí zobrazuje v aplikácií ovládajúcej robota ako "Toast". Funkciu reprezentuje modrá tehlička s nápisom PRINT.

V dátovej štruktúre je reprezentovaná typom *SimpleBrick*. V kompilátore sa do jazyka robota prekladá ako funkcia *print "message"*.

3.3.9 Sound

Funkcia slúžiaca na prehratie zvuku na zariadení ovládajúcom robota. Táto funkcia funguje na rovnakom princípe ako Print, ale reaguje iba na vyhradený text správy. Po jeho prijatí sa spustí notifikačný zvuk. Táto funkcia je reprezentovaná modrou tehličkou s nápisom SOUND.

Do dátovej štruktúry sa ukladá pod typom *SimpleBrick*. Pri kompilácií sa prekladá rovnako ako funkcia Print, pomocou funkcie *print "playSoundMessage"*. ale správa ma presne zadaný text vyhradený pre túto akciu.

3.3.10 Variable

Funkcia na priraďovanie hodnoty do premennej. Má na vstupe dva parametre a to názov premennej a priraďovanú hodnotu. Názvom premennej môže byť len znak základnej abecedy okrem znakov *x, y, z*, ktoré sú vyhradené pre cykly. Ako hodnotu môžeme priraďovať len celočíselné hodnoty typu *Integer*. V kóde je reprezentovaná oranžovou tehličkou s nápisom VAR.

V dátovej reprezentácií je typu *SimpleBrick*. Kompilátor ju prekladá do funkcie priradenia $x = y$.

3.3.11 Loop

Funkcia na cyklické opakovanie časti kódu. Na vstupe má tri celočíselné parametre a to počiatočnú hodnotu počítadla, konečnú hodnotu počítadla a krok akým počítadlo ide. Vzhľadom na možnosti OrbBasic jazyka je možné robiť cykly do maximálnej hĺbky tretej

úrovne. Je reprezentovaná počiatočnou tehličkou červenej farby s nápisom LOOP a koncovou tehličkou červenej farby s nápisom ENDLOOP.

V dátovej štruktúre je počiatočná tehlička typu *IfBrick* a koncová typu *SimpleBrick*. V počiatočnej časti cyklu je uložená aj referencia na jej koncovú časť. Kompilátor začiatok reprezentuje funkciou *for X = i to e step s*. Názov premennej počítadla sa mení v závislosti od hĺbky cyklu. Pri hĺbke prvej úrovne je *X*, pri druhej je *Y* a pri tretej je *Z*. Koniec cyklu je kompilovaný do funkcie *next X*, kde sa názov počítadla mení tým istým pravidlom ako na začiatku.

3.3.12 While

Funkcia na cyklické opakovanie kódu ako Loop s tým rozdielom, že tu sa opakuje pokiaľ platí zadaná podmienka. Podmienka sa zadáva na vstupe ako parameter. Pri tejto forme cyklu nie sme obmedzovaný žiadnou maximálnou hĺbkou vnárania. Túto funkciu reprezentuje červená tehlička s nápisom WHILE a pomocná, tiež červená tehlička s nápisom ENDWHILE ohraničujúca kód opakovania zdola.

Pri dátovej reprezentácií je to totožné ako pri funkcií Loop. Kompilátor začiatok tejto funkcie úplne vyhadzuje a koniec prepisuje do podmienky *if condition then goto*, ktorá ak je splnená, skáče na miesto, kde bol začiatok.

3.3.13 If

Funkcia na vetvenie programu podľa podmienky. Má na vstupe jednu premennú a to podmienku. Podmienka sa tvorí klasicky, ako to poznáme z iných programovacích jazykov. Ak chceme zmazať túto funkciu, obsah jej vetiev sa nezmaže, ale ostane v kóde v poradí najskôr then vetva a potom else vetva Táto funkcia sa skladá až zo šiestich tehličiek. Je to potrebné, aby sme v grafickom kóde ale aj v dátovej štruktúre vedeli ohraničovať jednotlivé vetvy. Hlavná tehlička má na sebe nápis IF, pomocné majú nápisy THEN, ELSE, END, ENDIF. Všetky sú rovnakej farby.

Hlavná časť je typu *IfBrick* a vedľajšie typu *HelperBrick*. Hlavná časť si v sebe uchováva referencie na všetky jej pomocné časti a naopak, tie si zas uchovávajú referenciu na ňu. Pri kompilácií sa kód usporadúva spomínanou formou v kapitole Balík com.melichercik.code v časti venujúcej sa triede *CodeCompiler*. Funkcia sa prekompiluje do formy *if condition then goto a else goto b*.

3.3.14 Return

Funkcia vracajúca čítanie programu na daný riadok. Táto funkcia sa v grafickom jazyku nachádza iba dočasne počas kompilácie. Pridáva sa na koniec každej vetvy a vracia čítanie programu za podmienku. Do jazyka robota sa prekladá ako funkcia *goto l.*

3.4 Možnosti aplikácie

Aplikácia disponuje niekoľkými možnosťami na nadviazanie spojenia s robotom, odoslanie programu pre robota, prerušenie aktuálneho vykonávania činností robota, vytváranie nového programu, uloženie existujúceho programu, načítanie alebo zmazanie uloženého programu a ovládanie grafickej plochy kódu.

3.4.1 Pripojenie robota

Na pripojenie robota je vytvorená samostatná aktivita *ConnectActivity*, pretože hlavná aktivita má upravený pohľad pre funkciu vykresľovania grafickej plochy kódu, na ktorý už nie je možné zobraziť element na vyhľadávanie a pripojenie robota. Tento element je súčasťou knižnice pre robota a teda je postačujúce ho iba vložiť do pohľadu novej aktivity.

Knižnica má plne obsiahnutú celú problematiku nadviazania spojenia s robotom a teda je potrebné iba inicializovať vyhľadávanie a nastaviť potrebný *Listener*, ktorý odchytí akciu nadviazania spojenia s robotom a potom toto spojenie uložiť do globálnej premennej celej aplikácie aby sa spojenie nestratilo pri prechode aktivitami.

3.4.2 Odoslanie programu robotovi

Odoslanie sa vykonáva vo funkcií *executeCode(),* ktorú nájdeme v hlavnej aktivite. V nej najskôr za pomoci funkcie *checkSpheroConnection()* zistíme či máme nadviazané spojenie s robotom. Ak máme, nasleduje kompilácia nášho grafického kódu uloženého v dátovej štruktúre do kódu OrbBasic čitateľného pre robota, ktorú sme si už priblížili. Potom zastavíme prípadné vykonávanie programu a pošleme robotovi aktuálny program. Tieto aktivity sa už vykonávajú jednoducho volaniami funkcií z knižnice pre robota.

3.4.3 Prerušenie vykonávania činností robota

Prerušovanie sa volá z funkcie *abortExecuting()* umiestnenej v hlavnej aktivite. Tak isto ako pri odoslaní programu najskôr kontrolujeme spojenie s robotom. Následne mu pošleme príkazy z knižnice pre robota, ktoré zastavujú všetku jeho aktivitu.

3.4.4 Nový program

Akcia vytvorenia nového programu prepíše celý aktuálny program novým prázdnym programom. Pri tejto akcií sa vytvára *AlertDialog*, po potvrdení ktorého sa spustí vytvorenie. Pri procese vytvárania sa v hlavnej aktivite *MainActivity* do premennej *code*, v ktorej je uložená aktuálna inštancia celého kódu, vytvorí inštancia kódu nanovo ale iba s jej inicializačnými dátami. Stará inštancia sa tým pádom stráca.

3.4.5 Uloženie programu

Program sa ukladá prostredníctvom serializácie ako celý objekt triedy *Code* zapísaný do súboru. Táto trieda teda musí implementovať *Serializable* a tak isto aj všetky objekty tried uložené v jej premenných. Uchováva si iba utriedený zoznam dátovej štruktúry tehličiek a teda jej základná trieda *Brick* tiež implementuje *Serializable*.

O celú túto akciu ukladania sa stará funkcia hlavnej aktivity *rawSaveProgram(final String fileName)*, ktorá dostáva na vstupe názov súboru. Vytvorený súbor sa ukladá na internom úložisku aplikácie priamo v pamäti zariadenia. Ak nejde o automatické uloženie programu, ktoré predchádza zanikaniu hlavnej aktivity, ale ide o uloženie vynútené užívateľom funkcii ukladania samotného súboru, predchádza funkcia *saveProgram()*, kde sa vytvára *AlertDialog*. V tomto dialógu sa zadáva to textového vstupu typu *EditText* názov ukladaného programu a teda názov samotného súboru.

3.4.6 Načítanie programu

Uložený objekt triedy *Code* v súbore sa z neho načíta a spätne sa z neho vytvorí objekt, ktorý sa uloží do premennej *code* v hlavnej aktivite. Tento proces samozrejme opäť prebieha za pomoci metódy serializácie.

V aplikácií máme opäť dva druhy načítavania a to automatické, vždy pri spúšťaní hlavnej aktivity alebo po vynútení užívateľom. Na konci obidvoch stojí funkcia

rawLoadProgram(final String fileName), ktorá tento proces spúšťa. Pri načítaní vynútenom od užívateľa jej predchádza funkcia *loadProgram()*, kde vyberá spomedzi všetkých uložených programov zobrazených v *ListView* a funkcia *loadProgramAction(final String file)*, kde sa vyberie akcia načítania.

3.4.7 Vymazanie programu

Akcia vymazania programu je postavená len na úplnom vymazaní súboru z úložiska zvoleného uloženého programu.

Vymazaniu predchádza funkcia *loadProgram()* spomínaná pri načítavaní, kde sa tak isto iba zvolí súbor a tiež po nej nasleduje funkcia *loadProgramAction(final String file)*. V poslednej spomínanej funkcii sa po zvolení možnosti vymazania program vymaže.

3.4.8 Ovládanie grafickej plochy

Grafická plocha ponúka možnosti posúvania celého kódu a aj jeho škálovanie.

Posúvanie začína vo funkcií *onTouch(View v, MotionEvent event)* v hlavnej aktivite, kde sa najskôr zisťuje či nemáme na displeji viac ako jeden dotyk a následne sa skontroluje aj to, či na displeji neostal ešte jeden dotyk po škálovaní. Ak sme prešli touto kontrolou do premenných *offsetX* a *offsetY* uložených v hlavnej aktivite, pripočítame rozdiel v daných osiach medzi aktuálnou a počiatočnou pozíciou dotyku.

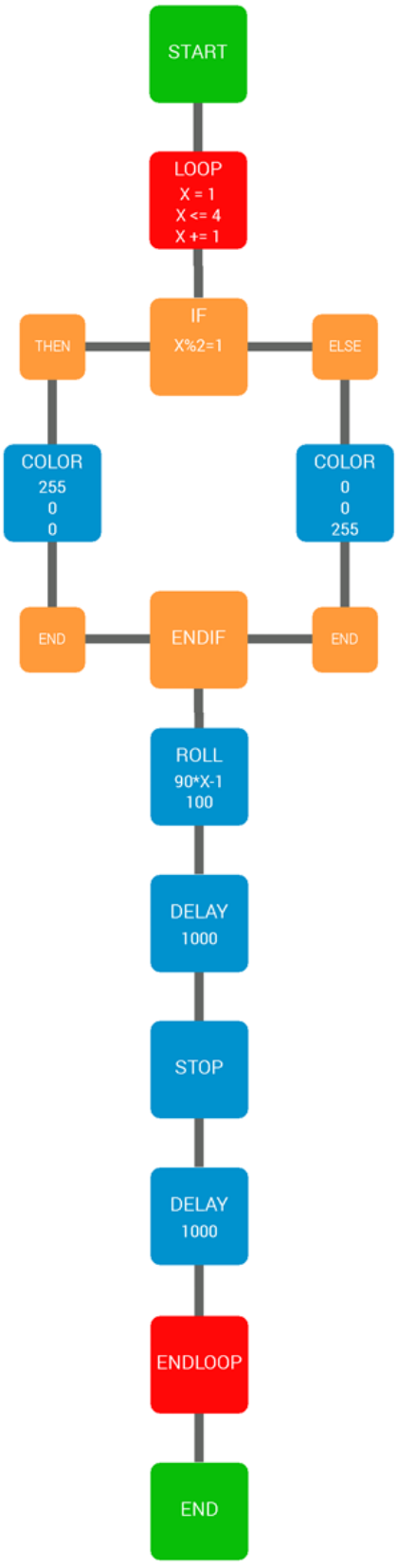
Vykresľovanie tehličiek je založené na tom, že sa im vypočíta poloha vzhľadom na poradie a vetvu v kóde. K týmto vypočítaným polohám sa pripočítavajú hodnoty posunutia z dotykovej udalosti. Výsledná hodnota sa odzrkadľuje v posunutí miesta vykresľovania tehličiek kódu.

Škálovanie začína v rovnakej funkcií ako posúvanie. Pre túto akciu v nej ale najskôr kontrolujeme či máme na displeji práve dva dotyky. Ak sme prešli touto kontrolou, zväčšenie alebo zmenšenie vypočítavame ako rozdiel vzdialeností aktuálnych a počiatočných dotykov. Tento rozdiel sa pre zjemnenie citlivosti delí číslom sto a následne sa pripočítava k premennej *scale* uloženej v hlavnej aktivite. Táto premenná reprezentuje šírku a výšku tehličky kódu v pixeloch. Jej počiatočná hodnota je 200, maximálna hodnota 300 a minimálna 50.

4 Ukážka programu

V nasledujúcej časti bude popísaný program, pomocou ktorého robot opíše štvorec a prikaždom zatočení striedavo mení farbu svojho osvietenia na modrú a červenú.

Na vrchu *Obrázku 21* na nasledujúcej strane môžeme vidieť začiatok celého programu funkciou START, za ňou nasleduje začiatok cyklu idúceho od 1 po 4 krokom 1. Štvorec má štyri strany takže sa štyri krát budeme otáčať. Ďalej je zachytená podmienka, ktorá rozvetvuje na základe toho či premenná cyklu nadobúda párnu alebo nepárnu hodnotu. Ak je hodnota nepárna robot sa rozsvieti na červeno a naopak ak je párna rozsvieti sa na modro. Za podmienkou prvá funkcia ROLL mu prikazuje aby sa začal hýbať vypočítaným smerom o rýchlosti 100. Program ďalej čaká jednu sekundu a potom zastavuje robota funkciou STOP. Opäť nasleduje sekundové čakanie aby sme robotovi dali čas zastaviť, nakoľko to nedokáže okamžite. V poslednej časti programu je už iba ukončenie (ohraničenie) cyklu a ukončenie celého programu funkciou END.



Obrázok 21 Ukážka programu štvorec

Záver

Cieľom práce bolo vytvoriť grafický programovací jazyk, ktorý bude dostatočne jednoduchý a prehľadný a aplikáciu pre mobilné zariadenia na platforme Android, do ktorej bude tento jazyk zasadený. Naplneniu tohto cieľa predchádzalo niekoľko krokov.

Najskôr bola potrebná analýza problematiky grafických programovacích jazykov. Tu som vychádzal z niekoľkých jazykov z tejto kategórie a pozeral som sa hlavne na ich grafickú stavbu a využitie programovacích konceptov v nich. Väčšina z nich mala príliš zložitú celkovú grafickú koncepciu pre prácu s nimi na mobilnom zariadení, ktoré disponuje menším displejom. Niekoľko z týchto jazykov som aj detailnejšie popísal v práci, kde som poukazoval na ich výhody a nevýhody vzhľadom na potreby môjho riešenia.

Informácie získané z analýzy som aplikoval v ďalšom kroku, návrhu grafickej stavby jazyka. Tu bolo najťažšie zvoliť správnu reprezentáciu funkcie, aby bolo jasné čo vykonáva a aké sú jej vstupné parametre. Zvolená reprezentácia mala hlavne jednoduchosť. Na tehličkách zastávajúcich funkcie nie sú zobrazované zbytočné informácie a ich dizajn je čistý, aby vytvorený kód pôsobil prehľadne.

Po návrhu jazyka nasledoval návrh jeho dátovej reprezentácie v aplikácií ako aj návrh jej samotnej. Pri dátovej štruktúre som prihliadal na to, aby bola logicky správne zostrojená a aby bola programová práca s ňou čo najjednoduchšia. Dizajn aplikácie bol navrhovaný tak, aby nepôsobil rušivo voči grafickému plátnu na vytváranie kódu.

Výsledná aplikácia má dostatočnú funkčnosť na to, aby sa mohla aj reálne používať za účelom výučby programovania na základných školách. Okrem tohto účelu aplikácia spolu s robotom poskytuje zábavu aj pre skúsenejších programátorov.

Aj po odovzdaní práce a aplikácie stále ostáva niekoľko smerov v ktorých je ešte stále priestor na vylepšovanie. K vylepšeniu grafiky by pomohla možnosť reprezentovať názov funkcie nie len slovom ale aj ikonkou. Ďalej by pomohla vlastná klávesnica aplikácie, ktorá by mala len znaky použiteľné na vstupoch. Pri ich zadávaní by mohlo ponúkať návrhy inicializovaných premenných v programe.

Použité zdroje

[1] RESNICK, M., MALONEY, J., MONROY-HERNANDEZ ´ , A., RUSK, N., STMOND,E., BRENNAN,K., MILLNER, A., ROSENBAUM, E., SILVER, J., SILVERMAN, B., AND KAFAI, 2009.
Scratch: Programming for all. Comm. ACM 52, 11, 60–67.

[2] Lifelong Kindergarten Group, MIT Media Lab
Programming Concepts and Skills Supported in Scratch, dátum prístupu: 16.1.2014
http://computerkiddoswiki.pbworks.com/f/Programming+Concepts+and+Skills+Supported+in+Scratch.doc

[3] The LEGO Group.
LEGO.com Mindstorms, dátum prístupu: 18.1.2014
mindstorms.lego.com

[4] iCharts, 2013.
Worldwide Smartphone Platforms Market Share, 2013 Q2 (Unit Share), dátum prístupu: 18.1.2014
http://www.icharts.net/chartchannel/worldwide-smartphone-platforms-market-share-2013-q2-unit-share_m33qyshgc

[5] Google Inc.
Bluetooth | Android Developers, dátum prístupu: 25.1.2014
http://developer.android.com/guide/topics/connectivity/bluetooth.html

[6] The New Toy, 2 Red House Square, Northampton, NN3 6WL, Anglie.
Sphero 2.0 | O nás | inteligentní robotická koule pro iOS, Android a Windows Phone, dátum prístupu: 30.1.2014
http://www.sphero.cz/o-nas

[7] Stephen Cass, 21.2.2012.
Rolling, Rolling, Rolling ... | MIT Technology Review, dátum prístupu: 30.1.2014
http://www.technologyreview.com/hack/426981/rolling-rolling-rolling/

[8] Orbotix, 29.5.2013.
orbBasic 1.07, dátum prístupu: 30.1.2014
https://github.com/orbotix/DeveloperResources/blob/master/docs/orbBasic_1.07.pdf

[9] Orbotix, 22.11.2013.
Sphero Android SDK, dátum prístupu: 30.1.2014
https://github.com/orbotix/Sphero-Android-SDK/

Prílohy

Súčasťou práce je aj CD. Popis jeho obsahu je nasledovný:

- zdrojový kód aplikácie
- bakalárske práca v PDF formáte