UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

PROGRAMOVACIE PROSTREDIE PRÍSTUPNÉ PRE NEVIDIACICH ŽIAKOV NIŽŠIEHO SEKUNDÁRNEHO VZDELÁVANIA

DIPLOMOVÁ PRÁCA

Bratislava 2018 **Matúš Kováč**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

PROGRAMOVACIE PROSTREDIE PRÍSTUPNÉ PRE NEVIDIACICH ŽIAKOV NIŽŠIEHO SEKUNDÁRNEHO VZDELÁVANIA

Diplomová práca

Študijný program: Aplikovaná informatika

Študijný odbor: 1114 aplikovaná informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Školiteľ: doc. RNDr. Ľudmila Jašková, PhD.

Čestné prehlásenie

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením školiteľky doc. RNDr. Ľudmily Jaškovej, PhD.s použitím literatúry a zdrojov uvedených v závere práce.

Bratislava, ...

Matúš Kováč

Poďakovanie

Abstrakt

Kľúčové slová:

Abstract

Key words:

Obsah

Zoznam obrázkov

Úvod

Cieľom práce je navrhnúť a implementovať programovacie prostredie pre nevidiacich žiakov nižšieho sekundárneho vzdelávania. Táto diplomová práca nadväzuje na bakalársku prácu Programovacie prostredie pre zrakovo postihnutích žiakov ZŠ (1) z roku 2016.

Pre túto prácu som sa rozhodol, pretože som chcel ďalej využiť vedomosti a skúsenosti, ktoré som nadobudol pri programovaní bakalárskej práce v oblasti tvorby softvéru pre vzdelávanie. Ďalej som chcel využiť skúsenosti z oblasti programovania kompilátorov a interpretrov ktoré som získal počas štúdia. Zároveň som chcel vytvoriť programovacie prostredie, ktoré sa bude reálne používať pri vyučovaní informatiky na základnej škole pre nevidiacich a slabozrakých.

Práca je rozdelená na 5 kapitol, každá kapitola sa viaže k jednej etape vytvárania aplikácie.

Prvá kapitola **Východiská** sa viaže k prípravnej etape, ktorá prebehla pred samotnou tvorbou aplikácie. Popisujem tu technológie a programy, ktoré som pri vývoji použil, uvádzam prehľad podobných aplikácií, v ktorých som našiel inšpiráciu, na záver sa venujem téme kompilátorov.

V 2. kapitole **Špecifikácia požiadaviek** sa venujem špecifikácii programovacieho jayzka, popisu funkcionality a požiadaviek na vzhľad používateľského prostredia.

V 3. kapitole **Návrh** detailne popisujem programovací jazyk a samotnú aplikáciu.

V 4. kapitole **Implementácia** sa venujem implementácii. Píšem o architektúre kompilátora, algoritmoch a dátových štruktúrach, ktoré som použil pri programovaní aplikácie.

Piata kapitola **Testovanie** obsahuje popis priebehu a závery z testovania, ktorého hlavná časť prebehla priamo so žiakmi základnej školy pre slabozrakých a nevidiacich v Bratislave.

1 Východiská

Táto kapitola je rozdelená na 4 časti. V nich postupne popisujem podobné programy a relevantnú odbornú literatúru. Ďalej poskytujem prehľad technológií, ktoré som buď použil pri implementácii, alebo spolupracujú s mojím programom. Na konci tejto časti píšem o kompilátoroch a interpretroch a materiáloch z ktorých som čerpal pri návrhu a implementácii môjho riešenia.

1.1 Žiaci so zrakovým postihnutím

Používateľmi aplikácie, ktorú som vytvoril sú predovšetkým nevidiaci žiaci. Venujme sa teraz podrobnejšie ich popisu. Ide o špeciálnu kategóriu zrakovo postihnutých žiakov.

Žiaci so zrakovým postihnutím (2) sa delia do 4 kategórií podľa druhu a stupňa zrakových porúch.

- **Žiaci s poruchami binokulárneho videnia** majú zrakové vnímanie narušené na základe funkčnej poruchy.
- **Slabozrakí žiaci** majú zníženú schopnosť zrakového vnímania v oblasti rýchlosti a presnosti, prípadne majú narušené zorné pole alebo farbocit.
- **Čiastočne vidiaci žiaci** tvoria hraničnú skupinu medzi nevidiacimi a slabozrakými. Pri niektorých činnostiach sú schopní využívať zvyšky zraku.
- **Nevidiaci žiaci** nedokážu príjmať informácie z okolitého sveta vizuálnou cestou. Pri práci s počítačom sú odkázaní na asistenčný softvér.

V práci sa zameriame na čiastočne vidiacich a nevidiacich žiakov a ich spôsob práce s počítačom, ktorý popisujeme nižšie. Pre obe kategórie budeme kvôli stručnosti a prehľadnosti používať názov **nevidiaci žiaci**.

1.1.1 Zásady tvorby softvéru pre zrakovo postihnutých

Nakoľko aplikácia je vyvíjaná pre potreby nevidiacih žiakov, ktorí ovládajú počítač len pomocou klávesnice s asistenciou čítača obrazovky, je potrebné, aby bola prispôsobená špecifickým potrebám žiakov. Nakoľko čítač obrazovky si s obrazovým výstupom neporadí, výstupom musí byť text, ktorý dokáže žiakovi sprostredkovať čítač obrazovk alebo zvuk, ktorý sa prehrá.

1.2 Prehľad použitých technológií

1.2.1 Čítač obrazovky NVDA a hlasový syntetizér Laura

NVDA (3) (NonVisual Desktop Access) je bezplatný čítač obrazovky. Čítač obrazovky umožňuje zrakovo postihnutým ľuďom používať počítač. Číta text na obrazovke syntetizovaným hlasom. Používateľ si vie vybrať čítaný text buď presunom kurzora myši na oblasť s textom, alebo pomocou klávesových príkazov. Je lokalizovaný do slovenčiny a dá sa dokúpiť aj prirodzene znejúci slovenský hlas Laura. Pri programovaní mojej aplikácie som dbal na to, aby všetky ovládacie prvky boli podporované týmto čítačom.

1.2.2 Syntéza reči pomocou eSpeak

eSpeak (4) je kompaktný hlasový syntetizér s otvoreným zdrojovým kódom pre viaceré jazyky vrátane Slovenčiny. V aplikácii využívam jeho rozhranie príkazového riadku na syntetizovanie textov na zvuky.

1.2.3 Jazyk C#

C# (5) je jednoduchý , moderný, objektovo-orientovnaý programovací jazyk pre všeobecné využitie. Je založený na základoch programovacích jazykov C++ a Java. Pri voľbe programovacieho jazyka som sa rozhodoval medzi viacerími jazykmi. Pre C# som sa rozhodol predovšetkým preto, že sa používa na vývoj aplikácií pre operačný systém Windows, ktorý je na slovenských školách najčastejšie používaným operačným systémom. Má veľmi dobrú podporu na tvorbu obsahu určeného pre zrakovo postihnutých používateľov.

1.2.4 Vývojové prostredie Visual Studio IDE

Visual Studio IDE (6) je integrované vývojové prostredie. Podporuje 36 programovacích jazykov vrátane jazyka C#. Pre vývoj aplikácie v prostredí Visual Studio som sa rozhodol predovšetkým preto, pretože ide o profesionálne vývojové prostredie, ktoré mi umožnilo naplno využívať potenciál programovacieho jazyka. Prostredie obsahuje editor zdrojového kódu podporujúci zvýrazňovanie syntaxe, automatické dopĺňanie a mnoho ďalších funkcií, ktoré uľahčujú a zrýchľujú prácu. Visual Studio má zabudovaný

debugger a návrhár formulárových aplikácií. Tento návrhár urýchľuje vytváranie používateľského rozhranie a jeho prvky sú podporované čítačom obrazovky NVDA.

1.2.5 Knižnica Autocomplete Menu

Autocomplete Menu (6) je knižnica s otvoreným kódom (opensource), ktorá slúži na automatické dopĺňanie textu z ponuky. V programe ju využívam aby som používateľovi zobrazil ponuku príkazov na základe znakov ktoré zadal. Po zvolení príkazu ho je možné stlační klávesy pridať do programu. [obrazok]

1.3 Prehľad podobných programov

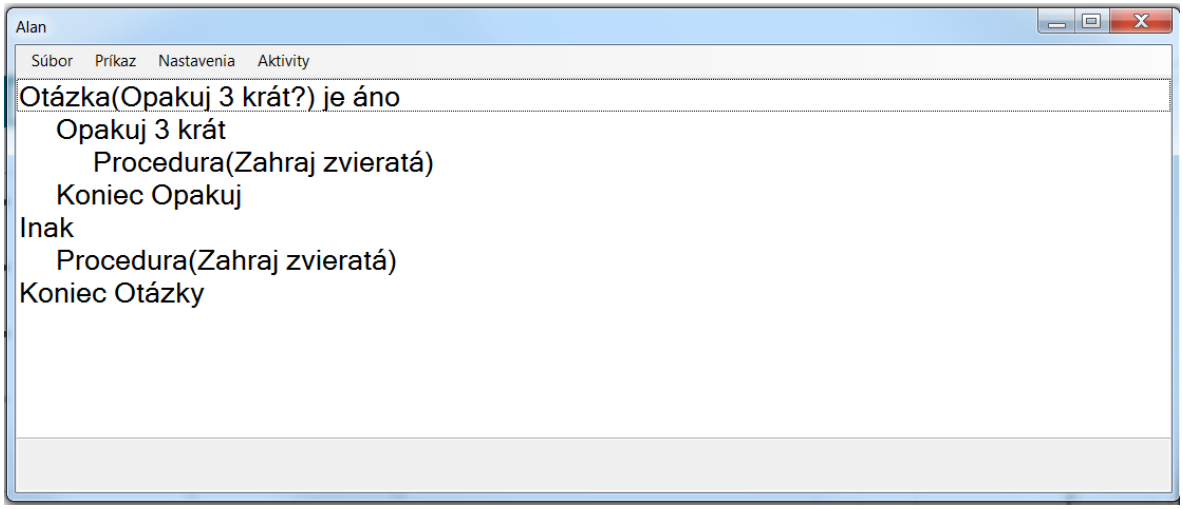
1.3.1 Zvukové programovacie prostredie Alan

Alan je program, ktorý vznikol vrámci bakalárskej práce **Programovacie prostredie pre zrakovo postihnutých žiakov ZŠ** (1). Je určený na výučbu programovania pre zrakovo postihnutých žiakov ZŠ. Pri vyučovaní základov programovania využíva zvukový programovací jazyk (8). Výstupom je sekvencia zvukov, ktorá sa pri spustení programu prehrá, čím poskytne žiakovi spätnú väzbu, aby dokázal vyhodnotiť správnosť riešenia. Má jednoduché grafické rozhranie tvorené hlavným oknom s aplikačnou ponukou. V aplikačnej ponuke sa nachádzajú všetky ovládacie prvky, pomocou ktorých si žiaci vytvárajú program. Program je možné uložiť do súboru, prípadne ho z neho načítať. Zvukový programovací jazyk obsahuje niekoľko druhov príkazov:

- Základné príkazy:
  - Zahraj - Zahrá zvolený zvuk z ponuky.
  - Povedz - Syntetizuje text na reč.
- Príkaz cyklu - umožňuje vytvárať jednoduché cykly so zadaným počtom opakovaní.
- Premenná a priradenie - umožňuje jednoduché priradenie celočíselnej premennej.
- Príkaz vetvenia - základný príkaz vetvenia, ktorý využíva premennú alebo vyhodnocuje pravdivosť tvrdenia.
- Procedúra - umožňuje vytvoriť a pomenovať blok príkazov.

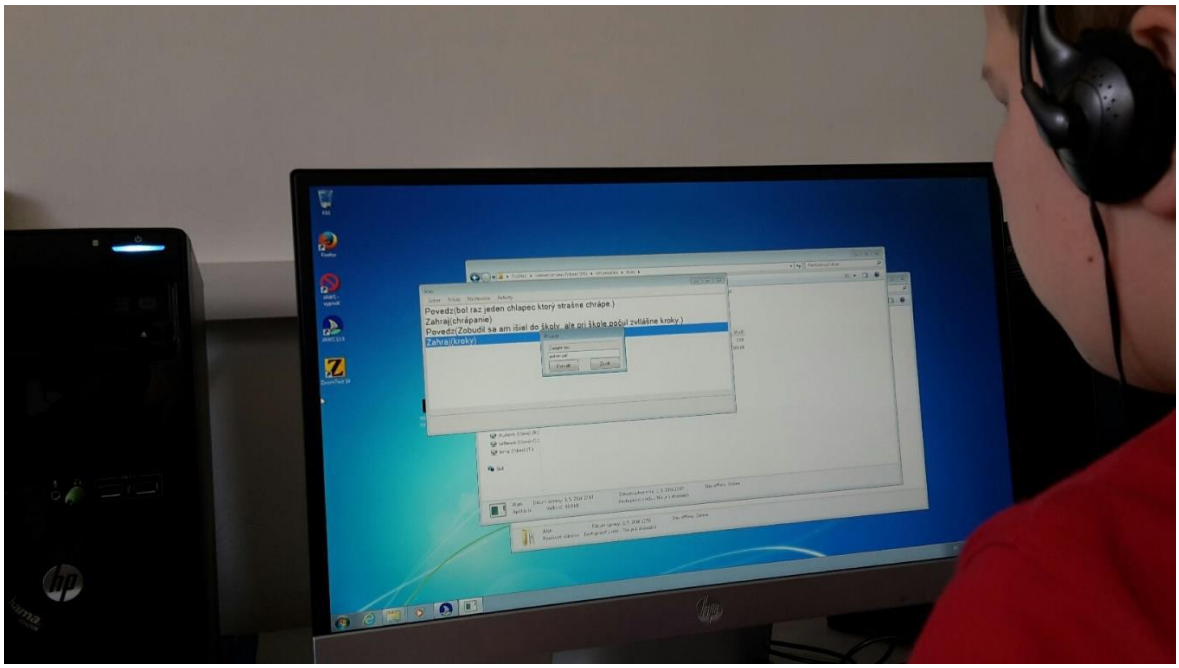
Tieto príkazy sa do programu zadávajú cez aplikačnú ponuku, a doplňujúce informácie sa zadávajú do dialógových okien. Pri často používaných príkazoch sa dajú

použiť klávesové skratky. Vďaka jednoduchému návrhu je pridávanie príkazov relatívne rýchle a intuitívne.



Obrázok 1 Program Alan

Program sa od školského roku 2016/2017 aktívne využíva na hodinách informatiky na základnej škole pre slabozrakých a nevidiacich v Bratislave. Zároveň za jeho pomoci prebieha výskum metodiky vyučovania informatiky pre nevidiacich žiakov na Katedre základov vyučovania informatiky Fakulty matematiky, fyziky a informatiky Univerzity Komenského v Bratislave (…). Moja práca nadväzuje na túto bakalársku prácu, bude určená pre starších žiakov a bude mať bohatší programováci jazyk a výrazne prepracovanejší kompilátor.



Obrázok 2 Žiak pracujúci s programom Alan

1.3.2 Animované programovacie prostredie pre deti ToonTalk

ToonTalk (8) je programovací systém, v ktorom je zdrojový kód animovaný a programovacie prostredie je počítačová hra. Programátor ovláda postavu programátora vo svete hry, pomocou ktorej vytvára, spúšta, debuguje a upravuje programy. Každý abstraktný výpočtový aspekt je zmapovaný do konkrétnej metafory. Napríklad výpočet je mesto, aktívny objekt je dom, vtáci ktorí nosia správy medzi domami sú metódami, a podobne. Tvorcovia sú presvedčení, že ich program dáva deťom príležitosť vytvárať reálne programy spôsobom, ktorý je jednoduchý a zábavný.

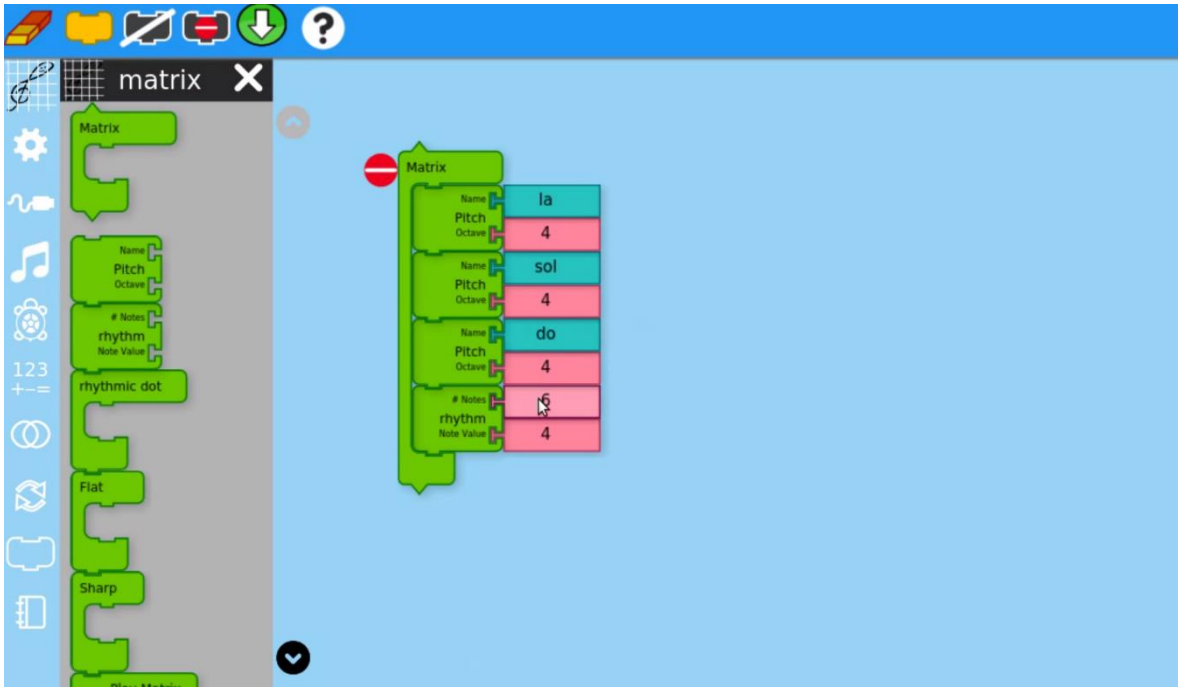
Obrázok 3 Snímok z programu Toontalk

ToonTalk je skvelým príkladom toho, na akú úroveň sa detské programovacie jazyky vyvinuli. Problémom je, že je použiteľný len pre vidiacich žiakov. Je zrejmé, že zvukový programovací jazyk nie je v porovnaní s jazykom ToonTalk z pohľadu vidiaceho žiaka dostatočne atraktívny. Na základe skúseností získaných z testovania programu Alan ale môžem povedať, že sa zvuky pri programovaní deťom veľmi páčili. Bude však zaujímavé zistiť ako budú reagovať starší žiaci.

1.3.3 Zvukové programovacie prostredie Music Blocks

Music Blocks (zistit a vlozit zdroj) je vzdelávací softvér s otvoreným zdrojovým kódom. Je navrhnutý tak, aby učiteľom a žiakom umožnil objavovať základné hudobné koncepty vo vizuálnom programovacom prostredí. Program je novým prostriedkom porozumenia základných hudobných konceptov, a zároveň nástrojom na výučbu

programovania a logiky. Vzhľad programu mi pripomína známy vzdelávací softvér Scratch. Využíva kartičky, ktoré používateľ pomocou myši ťahá na miesto určené pre tvorbu programu. Samotný jazyk je pomerne jednoduchý, je zložený z blokov, do ktorých používateľ pridáva tónycharakterizované pomocou doplňujúcich kartičiek (určujú vlastnosti tónu).



Obrázok 4 Okno programu MusicBlocks (...)

Základný princíp tvorby programu - ťahanie kartičiek - nieje použiteľný pre zrakovo postihnutých žiakov. Program je však pre mňa zaujímavý po stránke využitia základných hudobných konceptov, ktoré nevidiaci žiaci ovládajú, pre ľahšie pochopenie programovacích konceptov.

1.3.4 Fyzický programovací jazyk Torino

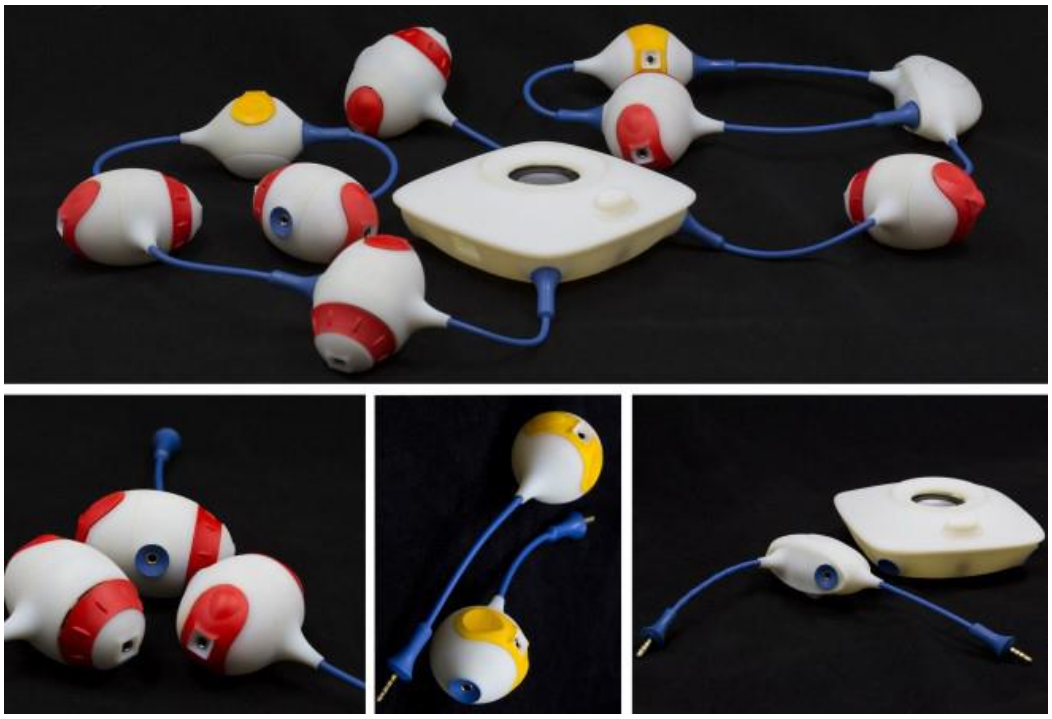
Torino (9) je fyzický programovací jazyk prístupný nevidiacim žiakom. Jeho základom sú inštrukčné prvky, ktoré sú fyzicky prepájateľné a manipulovateľné. Ich výstupom je príbeh alebo hudba. Žiaci môžu tieto prvky prepájať a vytvárať z nich program. Jazyk Torino 9 druhou inštrukčných prvkov:

- Generovanie tónu- je možné nastaviť druh a dĺžku tónu.
- Generovanie pauzy- je možné nastaviť dĺžku pauzy.

- Cyklus- je možné nastaviť počet opakovaní, je možné vytvoriť aj nekonečný cyklus.
- Podmienka- na základe vstupu žiaka je možné rozvetviť program.

Všetky inštrukčné prvky musia byť prepojené s hlavným prvkom, ten obsahuje 5 zásuviek ktoré sú spúštané paralelne.

Veľkým problémom nevidiacich žiakov je, že okrem zvuku nemajú pri programovaní žiadnu vizuálnu spätnú väzbu. Vďaka jazyku Torino si môžu program ohmatať ešte pred samotným spustením, a to im poskytne lepšiu predstavu o programe, ktorý vytvárajú.



Obrázok 5 Torino- fyzický programovací jazyk

1.3.5 Programovacie prostredie Sonic Pi

Sonic Pi (10) je programovacie prostredie s otvoreným zdrojovým kódom, určené na objavovanie a vyučovanie programovacích konceptov v školách, pomocou procesu vytvárania zvukov. Okrem toho, že Sonic Pi je vhodný výučbový nástroj, vyvinul sa na výkonné živé programovacie prostredie vhodné pre profesionálnych hudobníkov.

Jazyk tohto prostredia bol veľkou inšpiráciou pri písaní tejto práce. Spôsob, akým autor umožnil programovanie hudby jeho používateľom, je intuitívny, jednoduchý a zároveň profesionálny. Jeho základom sú cykly, ktoré v kombinácii s príkazmi syntetizujúcimi zvuky, a príkazom spánku umožňujú vytvárať melódie. Jazyk podporuje aj

podmienky, premenné, funkcie a vlákna. Tento jazyk obsahuje aj takzvaný živý cyklus, ktorý umožnuje meniť príkazy v cykle počas jeho vykonávania. Aj keď ide o zaujímavý koncept, v mojej práci sa živý cyklus neobjaví, nakoľko ho nepovažujem za vhodný pre nevidiaceho žiaka nižšieho primárneho vzdelávania.

Obrázok 6 Snímok prostredia Sonic Pi (...)

1.3.6 Programovací jazyk Quorum

Quorum (11) je univerzálny programovací jazyk. Program bol pôvodne určený pre zrakovo postihnutých žiakov, ale potom čo si získal popularitu, bola jeho funkcionalita rozšírená pre potreby všetkých žiakov. Momentálne má tento jazyk širokú škálu aplikácií. Dá sa v ňom spracovávať zvuk, vytvárať hry a iné. Základná funkcionalita jazyka umožňuje prácu s premennými (celočíselnými, reálnymi číslami, logickými a textom), základnými matematickými operátormi (+, -, *, /, mod), cyklami a podmienkami. V prípade programovania pokročilejších aplikácií je treba využiť knižnice. Quarum obsahuje knižnice pre prístupnosť, kontainery, spracovanie digitálneho signálu, grafiku, fyziku, hudbu a mnoho ďalších. Taktiež podporuje objektovo orientované programovanie, výnimky a pluginy. Vďaka jednoduchej syntaxi jazyka je použiteľný aj pre začiatočníkov.

Pokročilá funkcionalita spolu so schopnosťou kompilovať programy pre Windows, web, alebo iPhone z neho robia zaujímavú voľbu aj pre pokročilejšieho programátora.

1.4 Kompilátory

Pri navrhovaní a programovaní kompilátoru, som najviac čerpal z prednášok Ľubomíra Salanciho (12). V týchto prednáškach ukazuje neštandardný prístup k programovaniu kompilátorov, ktorý nieje založený na teoretických dôkazoch a postupoch, ale na praktickom prístupe a prehľadnom kóde.

1.4.1 Kompilátor

Kompilátor je softvér, ktorý prekladá zdrojový kód jedného programovacieho jazyka, do iného- cieľového jazyka. Kompilátory majú najdôležitejšiu úlohu pri prekladaní vyšších programovacích jazykov (C++, Java, C#, …) do nižších programovacích jazykov (assembler, strojový kód), čím uľahčujú prácu programátorom. Počas procesu prekladania zvykne kompilátor programátora upozorniť na chyby v zdrojovom kóde. V mojej práci využívam kompilátor na preloženie programovacieho jazyka pre žiakov do jazyka C#. Programovací jazyk pre žiakov má jednoduhší syntax. Pre žiakov je ľahšie pochopiteľný. Zdrojový kód je oproti ekvivalentnému kódu v jazyku C# kratší.

1.4.2 Časti kompilátora

Nakoľko programovanie kompilátora je komplexná práca, štandardne sa člení do niekoľkých častí. Nie každý kompilátor nutne musí obsahovať všetky časti, záleží to od náročnosti kompilátora a rozhodnutia autorov. Niektoré časti tiež môže riešiť programovací jazyk v ktorom je kompilátor programovaný (13).

- **Lexikálna analýza** je úvodná časť kompilátora. Jej vstupom je zdrojový kód. Analyzátor ho rozdelí na slová, ktoré vzniknú odstránením medzier a komentárov zo zdrojového kódu. V prípade že slovo sa nezhoduje, so žiadnym príkazom jazyka alebo názvom premennej, analyzátor vyhlási chybu. Ak slovo korešponduje s príkazom jazyka, Lexikálny analyzátor ho posunie syntaxovému analyzátoru.
- **Syntaxová analýza** dostane ako vstup slovo, ktoré patrí do jazyka. Toto slovo zaradí do syntaxového stromu, ktorý tvorí štruktúru programu a obsahuje v sebe všetky inštrukcie.

- **Kontrola typov** (type checking) analyzuje syntaxový strom, a kontroluje či program neporušuje určené pravidlá jazyka. Napríklad kontroluje či sú správne deklarované premenné.
- **Generátor prechodného kódu** (Intermediate code generation) preloží program do jednoduchého strojovo nezávislého prechodného jazyka. Prechodný kód eliminuje potrebu nového kompilátora pre každé jedinečné zariadenie.
- **Alokácia pamäte** Symbolické mená premenných použité pri generovaní prechodného kódu sú preložené na čísla, ktoré sa zhodujú s miestom v pamäti v strojovom kóde.
- **Generátor strojového kódu** preloží prechodný kód do textovej podoby strojového kódu (assembler) pre špecifické zariadanie.
- **Skladanie** (assembly) Pri skladaní je assemblerovský kód preložený do binárnej reprezentácie.

1.4.3 Vývojové prostrede Z7CLL IDE a Pascal-like jazyk Z7CLL na programovanie robota

Z7CLL (15) je programovacie prostredie a zároveň programovací jazyk, určený k programovaniu virtuálneho robota. Práca slúži ako nástroj na vyučovanie programovania. Autor sa rozhodol zvoliť kompilovaný jazyk, miesto interpretovaného prevažne z dôvodu eliminácie nutnosti interpretovania pri opätovnom spustení. V práci popisuje návrh jazyka a spôsob akým kompilátor implementoval. Centrom jazyka ja virtuálny robot ktorý je reprezentovaný ako dvojrozmerný obrázok. Robot sa pohybuje dvojrozmernej ploche a vykonáva úlohy. Programovací jazyk Z7CLL je silný jazyk a umožnuje programovať programátorsky pokročilé algoritmi (viď Obrázok 7 Fibonacciho postupnosť naprogramovaná v jazyku Z7CLL). obsahuje dva typy premenných, a to celočíselné a reálne. Ďalej obsahuje sadu príkazov na ovládanie robota, podmienené príkazy, nekonečné a konečné cykli a rozhranie na vytváranie funkcií s parametrom.

```
importuj("base");              // importovanie zdrojového kódu

definuj main() typu cele cislo premenne // definovanie funkcie
    nech i je cele cislo;      // definovanie lokálnych premenných
    nech x je cele cislo;
    nech y je cele cislo;
telo                           // začiatok tela funkcie
    nastav x na 1;             // nastavenie počiatočných hodnôt
    nastav y na 1;

    pre i rovne 1 po 10 urob // for cyklus
        sys_print(x);          // systémová funkcia na výpis čísla
        nastav y na x + y;
        nastav x na y - x;
    koniec;                    // ukončenie bloku cyklu

    vrat 0;                    // návratová hodnota funkcie
koniec;                        // ukončenie tela funkcie
```

Obrázok 7 Fibonacciho postupnosť naprogramovaná v jazyku Z7CLL

1.4.4 Návrh jazyka

Programovací jazyk použitý v mojej práci, bol navrhnutý školiteľkou v spolupráci s pani učiteľkou na základnej škole pre slabozrakých a nevidiacich v Bratislave. Pri navrhovaní jazyka bol kladený dôraz na to, aby bolo použité minimálne množstvo znamienok ako úvodzovky, bodkočiarky, dvojbodky a podobne, z dôvodu ľahšej zapamätateľnosti a zjednoduhšienia práce žiakom.

Jazyk je postavený na týchto základných príkazoch:

- **Zahraj-** príkaz pri vykonaní prehrá zvuk z množiny zvukou vybraných učiteľom. Má 1 parameter, a to názov zvuku. Syntax príkazu je `zahraj(zvuk)`. Napríklad `zahraj(pes)`.
- **Povedz-** program syntetizuje text zadaný používateľom. Má 1 parameter, a to ľubovoľne dlhý reťazec . Syntax príkazu je `povedz(text)`. Napríklad `povedz(ahoj, ako sa voláš?)`.
- **Tón-** program vygeneruje tón zadanej dĺžky. Má 2 parametre. Prvý parameter je názov tónu (c1,d1,c2,d2,…). Druhý parameter je dĺžka trvania. Môže byť zadaná v milisekundách, prípadne slovom (osmina, stvrt, cela,…). Syntax

jazyka je `ton(názov tónu, dĺžka)`. Napríklad `zahraj(c1,cela)`, alebo `zahraj(c1,500)`.

- **Ticho-** program spraví pauzu, nebude pokračovať vo vykonávaní ďalších príkazov zadanú dobu. Má jeden parameter, a to dĺžku pauzy zadanú v milisekundách alebo slovom. Syntax príkazu je `ticho(dĺžka)`. Napríklad `ticho(osmina)`.

V jazyku sa vyskytujú 4 druhy premenných, a to:

- **Cislo-** je celočíselná premenná. Pri prvom výskyte každej celočíselnej premennej v programe, je nutné ju inicializovať príkazom `cislo názov = hodnota`. Napríklad `cislo počet_opakovaní = 10`. Rovnako je možné premennú inicializovať pomocou príkazu `zadajCislo(premenná, výzva)` ktorý pri vykonávaní programu otvorí dialógové okno do ktorého používateľ zadá hodnotu premennej. Tento príkaz má 2 parametre. Prvý parameter je názov premennej, a druhý výzva ktorá sa zobrazí v dialógovom okne. S celočíselnou premennou je možné vykonávať základné aritmetické operácie (sčítavanie, odčítavanie, násobenie a delenie). Je možné používať aritmetické operátory (+,-,*,/) alebo príkazy (`pripocitaj,odpocitaj, nasob, vydel`). Príkazy majú 2 parametre, prvý je názov premennej a drúhy je hodnota. Syntax týchto príkazov je rovnaký, a to `pripocitaj(názov_premennej, hodnota)`. Napríklad príkaz `pripocitaj(pocet,5)` pripočíta k premennej pocet hodnotu 5. Rovnaké pripočítanie je možné vykonať aj príkazom `pocet = pocet + 5`.
- **Text-** je textový reťazec. Pri prvom výskyte každej textovej premennej v programe je nutné ju inicializovať príkazom `text meno = eva`. Nakoľko sme sa snažili vyhnúť používaniu úvodzoviek, do premennej sa priraďí reťazec až do konca riadku s inicializáciou prípadne priradením. Alternatívou k tejto inicializácii je príkaz `zadajText(názov_premennej, výzva)` ktorý otvorí dialógové okno s výzvou do ktorého používateľ zadá hodnotu premennej. Príkaz má dva parametre, prvý je názov premennej a druhý výzva ktorá sa zobrazí v dialógovom okne.

- **Zvuk-** je premenná ktorá obsahuje názov zvuku. Pri prvom výskyte každej zvukovej premennej v programe je nutné ju inicializovať príkazom `zvuk zviera = pes`, prípadne príkazom `zadajZvuk(názov_premennej, výzva)`. Tento príkaz zobrazí dialógové okno s výzvou z parametra, v ktorom si používateľ z ponuky zvukou vyberie jeden ktorý sa priradí do premennej.
- **Postupnost-** je premenná ktorá obsahuje postupnosť zvukov. Inicializuje sa príkazom `Postupnost názov_premennej = prvok1; prvok2; …; prvokN;`. Napríklad `Postupnost NaseZvierata = pes; macka; papagaj;`.

  S postupnosťami je možné vykonávať nasledujúce operácie:
  - `Prvy(postupnost)`- zahrá prvý zvuk postupnosti.
  - `Posledny(postupnost)`- zahrá posledný zvuk postupnosti.
  - `Nasledujuci(postupnost)`- zahrá nasledujúci zvuk postupnosti.
  - `Predchadzajuci(postupnost)`- zahrá predchádzajúci zvuk postupnosti.
  - `Prvok(postupnost, číslo)`- zahrá ten zvuk postupnosti, ktorý sa nachádza na poradí ktoré udáva parameter číslo. Poradie je indexované od nuly.
  - `Dlzka(postupnost)`-Vráti číslo ktoré údáva počet prvkov postupnosti.

  Napríklad, nasledujúca postupnosť príkazov zahrá celú postupnosť:

```
Pocet = Dlzka(skladba)
Odpocitaj(Pocet,1)
Prvy(Skladba)
Opakuj Pocet krat
    Nasledujuci(Skladba)
Koniec Opakuj
```

V jazyku sa nachádzajú dva druhy cyklou:

- **Cyklus s pevným počtom opakovaní-** počet opakovaní je možné zadať číslom, alebo celočíselnou premennou. Musí to byť kladné číslo väčšie ako 0 a menšie ako 51. Syntax príkazu je:

```
Opakuj pocet krat
   blok príkazov
koniec opakuj
```

Napríklad:

```
Opakuj 10 krat
   Povedz(Nebudem podvádzať na teste)
Koniec Opakuj
```

- **Cyklus s podmienkou-** ide o obdobu while cyklu. Telo s podmienkou sa vykonáva pokiaľ je splnená podmienka. Syntax príkazu je:

```
Kym podmienka rob
   blok príkazov
Koniec Kym
```

Napríklad:

```
Cislo pocet = 10
Kym pocet > 0
   Zahraj(pes)
   pocet = pocet - 1
Koniec kym
```

V jazyku sa nachádzajú dva druhy vetviacich príkazov:

- **Príkaz vetvenia- otázka-** je príkaz vetvenia, ktorý má 1 parameter text otázky. Pri vykonávaní programu sa používateľovi otvorí dialógové okno s textom otázky, na ktorú môže odpovedať kliknutím na tlačidlo áno alebo nie. V závislosti od odpovede používateľa je následne vykonaný blok príkazov. Syntax príkazu je nasledovná:

```
Otazka(text otazky) je ano
     blok príkazov
Inak
     blok príkazov
Koniec Otazka
```

Napríklad:

```
Otázka(Chceš ísť na farmu?) je ano
     Povedz(Vitaj na farme. Máme tu kravičky)
     Zahraj(Krava)
Inak
     Povedz(Tak ideme pozrieť vlaky)
     Zahraj(Lokomotiva)
Koniec Otazka
```

- **Príkaz vetvenia s premennou-** je klasické vetvenie pri ktorom je možné použiť celočíselnú alebo textovú premennú. Ako operátor je možné jeden zo znakov >, <, <=, >=, =<, =>, <>, ><. V závislosti od pravdivosti tvrdenia je vykonaný blok príkazov. Syntax:

```
Ak(Premenna1 = Premenna2)
     blok príkazov
Inak
     blok príkazov
Koniec Ak
```

Napríklad:

```
ZadajCislo(Vek, Koľko máš rokov?)
Ak (Vek < 15)
     Povedz(Pôjdeme do zoo)
     Zahraj(Opica)
Inak
     Povedz(Pôjdeme do strašidelného zámku)
     Zahraj(Výkrik)
Koniec Ak
```

V jazyku je možné vytvárať bloky príkazov, ktoré sa nazývajú procedúry. Syntax príkazu na vytvorenie procedúry:

```
Urob názov_procedúry
     blok príkazov
Koniec Urob
```

Syntax príkazu na volanie procedúry:

```
názov_procedúry
```

Napríklad procedúru:

```
Urob Zoo
```

```
    Zahraj(Opice)
    Zahraj(Slon)
    Zahraj(Tiger)
Koniec Urob
```

Zavoláme príkazom:

`Zoo`

V jazyku sa nachádzajú príkazy určené na generovanie náhodných premenných:

- **Generovanie náhodného zvuku** sa dá dosiahnuť pomocou troch príkazov:
  - Generovanie náhodného zvuku zo všetkých, ktoré sú k dispozícii:

    `Nahodne`
  - Generovanie náhodného zvuku zo všetkých zvukou jednej kategórie:

    `Nahodne(Kategoria)`
  - Generovanie náhodného zvuku z postupnosti:
    `Nahodne(Postupnost)`
- **Generovanie náhodného čísla** sa robí pomocou príkazu `Nahodne(cislo1, cislo2)` ktorý vráti náhodné číslo z intervalu <`cislo1`,`cislo2`>.

2 Špecifikácia požiadaviek

2.1 Špecifikácia jazyka

.

3 Návrh

3.1 Návrh prostredia

4 Implementácia

4.1 Vývojové prostredie

5 Testovanie

Záver

Príloha

Krátky manuál

Použitá literatúra

1. **Kováč, Matúš.** *Programovacie prostredie pre zrakovo postihnutých žiakov ZŠ, bakalárska práca.* Bratislava : FMFI UK, 2016.

2. **Šimko, Jozef, Šimko, Marian a Dredan, Štefan .** *Vzdelávanie žiakov so zrakovým postihnutím.* [Online] [Dátum: 4. 1 2018.] http://www.nucem.sk/documents//48/skoly/skolenie_pedagogov_a_pedagogickych_zamestnancov/142606_Vzdel%C3%A1vanie_%C5%BEiakov_so_zrakov%C3%BDm_postihnut%C3%ADm.pdf.

3. ***NVDA*. [Online] [Dátum: 4. 1 2018.] https://www.nvaccess.org/.**

4. eSpeak. [Online] [Dátum: 8. 1 2018.] http://espeak.sourceforge.net/.

5. C#. [Online] [Dátum: 1. 8 2018.] https://docs.microsoft.com/en-us/dotnet/csharp/.

6. Microsoft. Visual Studio IDE. [Online] [Dátum: 8. 1 2018.] https://www.visualstudio.com/vs/.

7. Torgashov, Pavel. *Autocomplete Menu.* [Online] [Dátum: 4. 1 2018.] https://www.codeproject.com/Articles/365974/Autocomplete-Menu.

8. Jaime Sánchez, Fernando Aguayo. APL: Audio Programming Language for Blind Learners. [Online] [Dátum: 1. 26 2018.] http://repositorio.uchile.cl/bitstream/handle/2250/124982/Sanchez_Jaime_APL.pdf.

9. Kahn, Ken. ToonTalk. [Online] [Dátum: 8. 1 2018.] http://www.toontalk.com/Papers/jvlc96.pdf.

10. Anja Thieme, Cecily Morrison, Nicolas Villar, Martin Grayson a Siân Lindley. *Torino.* [Online] [Dátum: 19. 1 2018.] https://www.microsoft.com/en-us/research/wp-content/uploads/2017/03/Torino_Camera_ready_06_04_17-1.pdf.

11. Aaron, Sam. *Sonic Pi.* [Online] [Dátum: 21. 1 2018.] http://sonic-pi.net/.

12. *Quorum.* [Online] [Dátum: 21. 1 2018.] https://www.washington.edu/doit/introduction-quorum-0.

13. Salanci, Ľubomír. *Kompilátory a interpretre.* [Online] [Dátum: 2017. 5 17.] https://www.edi.fmph.uniba.sk/~salanci/Kompilatory/index.html.

14. Mogensen, Torben Ægidius. *Basics of Compiler Design*. 2010. ISBN 978-87-993154-0-6.

15. Gerencséri, Kristián. Vývojové prostredie Z7CLL IDE a Pascal-like jazyk Z7CLL na programovanie robota. *Česko-Slovenská študentská vedecká konferencia v didaktike informatiky*. 2017.