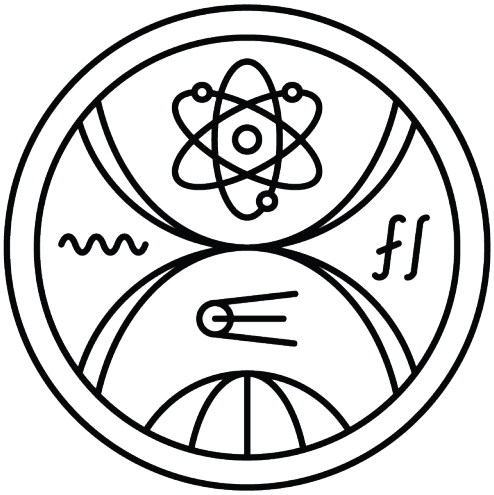
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



EDUKAČNÉ PROSTREDIE NA PROGRAMOVANIE HUDBY PRÍSTUPNÉ PRE NEVIDIACICH ŽIAKOV SEKUNDÁRNEHO VZDELÁVANIA

DIPLOMOVÁ PRÁCA

2023
BC. JAKUB ŠVORC

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

EDUKAČNÉ PROSTREDIE NA PROGRAMOVANIE HUDBY PRÍSTUPNÉ PRE NEVIDIACICH ŽIAKOV SEKUNDÁRNEHO VZDELÁVANIA

DIPLOMOVÁ PRÁCA

Študijný program:	Informatika
Študijný odbor:	Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	doc. RNDr. Ľudmila Jašková, PhD.

Bratislava, 2023
Bc. Jakub Švorc

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Jakub Švorc
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Edukačné prostredie na programovanie hudby prístupné pre nevidiacich žiakov sekundárneho vzdelávania
Educational environment for music programming accessible to secondary blind pupils

Anotácia: Autor vytvorí programovacie prostredie s vlastným kompilátorom alebo interpreterom. Základné príkazy zabudovaného programovacieho jazyka budú slúžiť na prehratie tónov zvoleným hudobným nástrojom. Okrem toho bude možné použiť aj komplikovanejšie štruktúry, ako je cyklus, príkaz vetvenia, podprogram, vlákno.
Editor kódu bude mať zabudovanú kontrolu syntaxe a funkciu prediktívnej ponuky príkazov.
Prostredie bude prístupné pre čítač obrazovky a bude plne ovládateľné pomocou klávesnice. Nevidiacim používateľom umožní okrem bežnej práce s textom aj jednoduchým spôsobom získať prehľad o štruktúre vytvoreného kódu.
Použiteľnosť výslednej aplikácie pre cieľového používateľa bude zabezpečená vďaka výskumu vývojom (design based research), t.j. iteratívnym vývojom a overovaním s rôznymi typmi používateľov (nevidiaci programátor, učiteľ nevidiacich žiakov, nevidiaci žiak).

Cieľ: Vytvoriť programovacie prostredie umožňujúce programovať hudbu pozostávajúcu z viacerých paralelne znejúcich melódií. Dôraz bude kladený na zabezpečenie plnej prístupnosti a efektívnej práce s editorom kódu pre žiakov so zrakovým postihnutím.

Literatúra: S. Aaron, Code music with Sonic Pi, Retrieved from https://www.raspberrypi.org/magpi-issues/Essentials_Sonic_Pi-v1.pdf
C. C. De Oliveira, Designing educational programming tools for the blind: mitigating the inequality of coding in schools, 2017.
HADWEN-BENNETT, A. et al. Making Programming Accessible to Learners with Visual Impairments: A Literature Review, International Journal of Computer Science Education in Schools, April 2018, Vol. 2, No. 2, ISSN 2513-8359.

Vedúci: doc. RNDr. Ľudmila Jašková, PhD.
Katedra: FMFI.KDMFI - Katedra didaktiky matematiky, fyziky a informatiky
Vedúci katedry: prof. RNDr. Ivan Kalaš, PhD.

Poďakovanie: Rád by som poďakoval mojej školiteľke doc. RNDr. Ľudmile Jaškovej, PhD., za vedenie práce, cenné rady, trpezlivosť a motiváciu pri tvorbe Diplomovej práce.

Abstrakt

Táto diplomováho práca opisuje vývoj programovacieho prostredia s vlastným kompilátorom a jazykom. Aplikácia je určená pre žiakov 2. stupňa základnej školy so zrakovým znevýhodnením. Základné príkazy zabudovaného programovacieho jazyka slúžia na prehratie tónov zvoleným hudobným nástrojom. Je možné použiť aj komplikovanejšie štruktúry, ako je cyklus, príkaz vetvenia, podprogram, vlákno. Editor kódu bude mať zabudovanú kontrolu syntaxe a funkciu prediktívne ponuky príkazov. Prostredie bude prístupné pre čítač obrazovky a bude plne ovládateľné pomocou klávesnice. Nevidiacim používateľom umožní okrem bežnej práce s textom aj jednoduchým spôsobom získať prehľad o štruktúre vytvoreného kódu. Použiteľnosť výslednej aplikácie pre cieľového používateľa bude zabezpečená vďaka výskumu vývojom (design based research), t.j. iteratívnym vývojom a overovaním s rôznymi typmi používateľov (nevidiaci programátor, učiteľ nevidiacich žiakov, nevidiaci žiak).

Kľúčové slová: jedno, druhé, tretie

Abstract

This master's thesis describes the development of a programming environment with its own compiler and language. The application is intended for students of the 2nd level of primary school with visual impairments.

The basic commands of the built-in programming language are used to play tones with a chosen musical instrument. It is also possible to use more complex structures, such as loops, conditional statements, subprograms, and threads.

The code editor will have built-in syntax checking and a predictive command suggestion feature. The environment will be accessible for screen readers and fully controllable via the keyboard.

For visually impaired users, it will allow not only regular text work but also provide a simple way to understand the structure of the created code. The usability of the resulting application for the target user will be ensured through design-based research, i.e., iterative development and testing with various types of users (blind programmers, teachers of blind students, blind students).

Keywords:

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Citácie:
[6] [14] [15]

Motivácia

Kapitola 1

Prehľad problematiky

V tejto kapitole sa zoznámime s technológiami určenými pre zrakovo postihnutých používateľov. Porovnáme ich technické prevedenia, zhodnotíme ich použiteľnosť pre nevidiacich žiakov a popíšeme si znalosti a zručnosti, ktoré chceme rozvíjať vrámci vzdelávacieho procesu. Preskúmame možnosti týchto softvérov, ich širšej aplikácie a náväznosti na ďalšie učivo v oblasti informatiky. Zadefinujeme, čo potrebuje vedieť žiak základnej školy, aby čo najjednoduchšie a najlepšie pochopil základné programovacie konštrukcie a pojmy. Následne si predstavíme podobné existujúce implementácie.

1.1 Typy zrakového postihnutia a ich kategorizácia

Zrakové postihnutie sa prejavuje rôzne na základe čoho, ho vieme kategorizovať do niekoľkých typov, podla [11]. V tejto podkapitole si ich stručne popíšeme, aby sme vedeli navrhnúť prístupný softvér pre každého. Pokúsime sa lepšie porozumieť potrebám jednotlivcov so zrakovým postihnutím a implementovať výsledky do nášho prostredia.

1.1.1 Kategórie zrakového postihnutia

V tejto kapitole si uvedieme prehľad kategórií zrakových postihnutí a ich jednotlivé charakteristiky [11].

Nevidiaci

Nevidiaci s úplnou stratou zraku sú závislí na alternatívnych spôsoboch získavania informácií, ako sú hlasové technológie, Braillovo písmo a hmatové pomôcky.

Čiastočne vidiaci a slabozrakí

Čiastočne vidiaci neprišli úplne o zrak, majú problémy najmä v priestorovej orientácii. Slabozrakí, podobne ako čiastočne vidiaci, majú ťažkosti s ostrosťou videnia, periférneho

videnia, rýchlosti a presnosti zraku. Obe skupiny používateľov môžu do istej miery reagovať na vizuálne podnety. Dôležité je dbať na možnosti prispôsobenia ako sú farebné kontrasty, nastavenia veľkosti znakov, objektov a klávesové skratky.

Osoby s poruchami binokulárneho videnia

Poruchy binokulárneho videnia sú problémy so schopnosťou očí spolupracovať pri vnímaní priestoru a jeho hĺbky. Ide o široké spektrum zrakového postihnutia, teda tieto poruchy môžu mať rôzne príčiny a môžu sa prejaviť rôznymi spôsobmi. Môžu mať aj vplyv na spoluprácu očí pri sledovaní objektov.

Cieľová skupina Naša aplikácia je určená pre skupinu nevidiacich, pre druhý stupeň základných škôl.

1.2 Čítače obrazovky

V tejto časti predstavíme programy, ktoré boli vyvíjané s ohľadom na špecifické potreby programátorov so zrakovým postihnutím. Taktiež spomenieme pomocné nástroje, ktoré súvisia s týmito aplikáciami alebo sú v nejakej forme ich súčasťou.

Čítač obrazovky je softvér, ktorý umožňuje prečítať text ktorý je zobrazený na obrazovke, prostredníctvom syntetizátora reči alebo braillovho displeja. [3] Preto sú kľúčovým nástrojom v digitálnom svete. Ich vývoj sa v priebehu času výrazne zdokonalil vzhľadom na narastajúce potreby používateľov a nové technológie.

Používatelia sa nevyhnutne stretnú aj s nedostatkami čítačov, najmä pri komplexných webových stránkach či nie optimalizovaných aplikáciách. Niektoré grafické prvky môže byť obtiažne interpretovať, čím používateľ prichádza o informácie. Navyše sú limitovaní skutočnosťou, že nie všetky aplikácie (či už webové alebo desktopové) dodržiavajú normy pre spoluprácu s čítačmi obrazovky.

1.2.1 NVDA - NonVisual Desktop Access

NVDA je open-source čítač obrazovky, určený primárne pre operačný systém Windows. Má za sebou živú komunitu a je pravidelne aktualizovaný. NVDA podporuje až 55 jazykov vrátane slovenského.

Vývojármi tohto softvéru sú dvaja nevidiaci programátori, ktorí z vlastných skúseností dokázali navrhnúť softvér, ktorý sa následne začal používať vo viac ako 175 krajinách. Nie je adresovaný len programátorom, ale širokej verejnosti a pomáha používateľom s každodennými úlohami ako je pohyb na internete, písanie dokumentov, financie, online komunikácia a mnoho ďalších.

Je plne kompatibilný s aplikáciami Word, Excel, PowerPoint, Google Chrome, Mozilla, Microsoft Edge, WordPad, NotePad, a keďže ho preferujú aj žiaci, tak aj s naším prostredím.

1.2.2 JAWS (Job Access With Speech)

Patrí medzi najpopulárnejšie čítače obrazovky na svete. Poskytuje hlasový a Braillový výstup pre počítačové aplikácie [3]. Obsahuje :

- ovládače pre populárne Braillové displeje
- funkciu OCR (optické rozoznávanie znakov) pre obrazové súbory alebo neprístupné PDF dokumenty
- službu Picture Smart na rozpoznávanie obrazov
- hlasového asistenta pre často používané príkazy

Pracuje s Microsoft Office, Google Docs, Chrome, Edge, Firefox a mnohými ďalšími. Podporuje Windows® 11, Windows 10, Windows Server® 2019 a Windows Server 2016.

Braillov displej

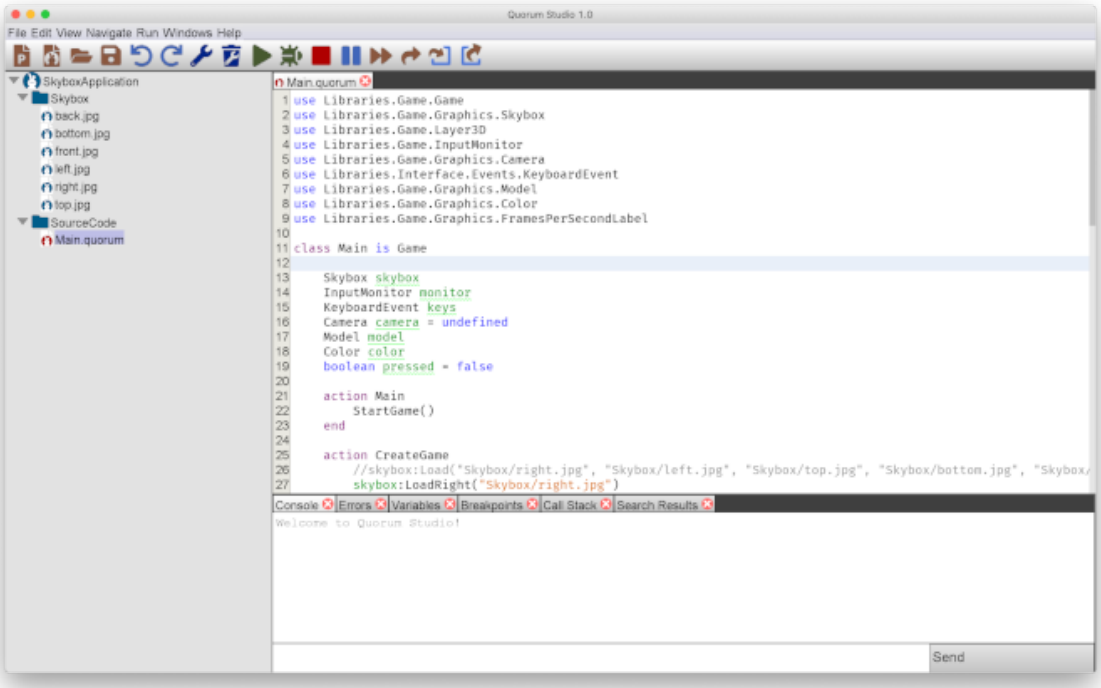
Braillov displej je zariadenie na obrázku 1.1, ktoré môže užívateľ s čítačom obrazovky synchronizovať s počítačom, tabletom (napríklad iPadom) alebo telefónom. Displej zobrazuje hmatateľné braillové znaky textu z obrazovky. Vďaka tomu ho môžu používať aj nepočujúci, nie len nevidiaci ľudia. Tieto zariadenia, narozdiel od čítačov obrazovky, nie sú tak finančne dostupné. [8]

Obr. 1.1: Braillov displej

1.3 Analýza programov pre nevidiacich študentov

1.3.1 Quorum

Interpretovaný programovací jazyk Quorum prišiel spolu s integrovaným vývojovým prostredím (Integrated Development Environment - IDE) Quorum Studio v roku 2016 [5].



Obr. 1.2: Quorum programovacie prostredie

Jedná sa o programovací jazyk na tvorbu softvéru aj pre používateľov s rôznym fyzickým postihnutím. Na jeho tvorbe, finančnej podpore a údržbe sa podieľa niekoľko škôl a univerzít v USA.

Použiteľnosť jazyka Quorum

Quorum je navrhnutý tak, aby čo najviac zjednodušil syntax a zároveň podporil plnú funkcionalitu štandardných programovacích jazykov. Konkrétne poskytuje plnú funkcionalitu a silu jazyka Java, nad ktorou bol postavený interpreter jazyka. Kompatibilita s JVM poskytuje rýchlosť, platformovú nezávislosť, pričom zjednodušuje syntax tak, aby dodržiavala štandardy , ktoré sú dôležité pre zrakovo postihnutých používateľov a spoluprácu s čitačom obrazovky - vynecháva zložité znaky, nie je potrebné dodržovanie odsadenia, podpora debuggovania, ovládateľnosť klávesovými skratkami.

Porovanie s Javou

S Javou sa spájajú mnohé nepríjemnosti, ktoré sa týkajú jej syntaxe - dlhé názvy tried, ťažkopádna práca s textovými reťazcami (chýbajúca podpora formátovania stringov alebo iná manipulácoa). Ďalej zložité názvy výnimiek, zložitá práca s objektovými premennými - na ich prácu je potrebné používať gettery a settery namiesto properties, množstvo vygenerovaného kódu (neobsahuje top-level statements ako C#, C++, C) a iné.

Použiteľnosť nástroja Quorum je široká, nie je špecificky určený pre používateľov so

zrakovým postihnutím. Kvôli tomu obchádza niektoré odporúčania a štandardy pre vývoj softvéru pre zrakovo postihnutých. Pri použití starších verzií je potrebné mať už nainštalovanú Javu, ak ju inštalačný balíček nemá v sebe obsiahnutú.

1.3.2 Sonic Pi

Open-source aplikácia Sonic Pi [1] je interaktívne programovacie prostredie s cieľom využitia na hudobnú tvorbu. Jeho jednoduchý jazyk a zameranie na vytváranie skladieb prostredníctvom programovania ho robia prístupným a pútavým pre študentov, ktorí majú záujem o hudobné aplikácie. Avšak práve jeho špecifickosť využitia môže byť obmedzením z hľadiska všeobecných vzdelávacích cieľov.

V článku [13] o testovaní SonicPi autori sledovali, ako študenti budú hodnotiť vlastnosti softvéru ako sú zábavnosť, relevantnosť a frustrácia v rôznych fázach tohto testovania (obr. 1.4). Testovacia skupina mala zo začiatku problém zorientovať sa a niektorí študenti nemali toľko skúseností z oblasti informatiky. Napriek tomu po tomto testovaní softvér a samotný proces programovania hodnotili pozitívne (obr. 1.3) aj napriek ťažkému začiatku (niekedy frustrujúcemu).

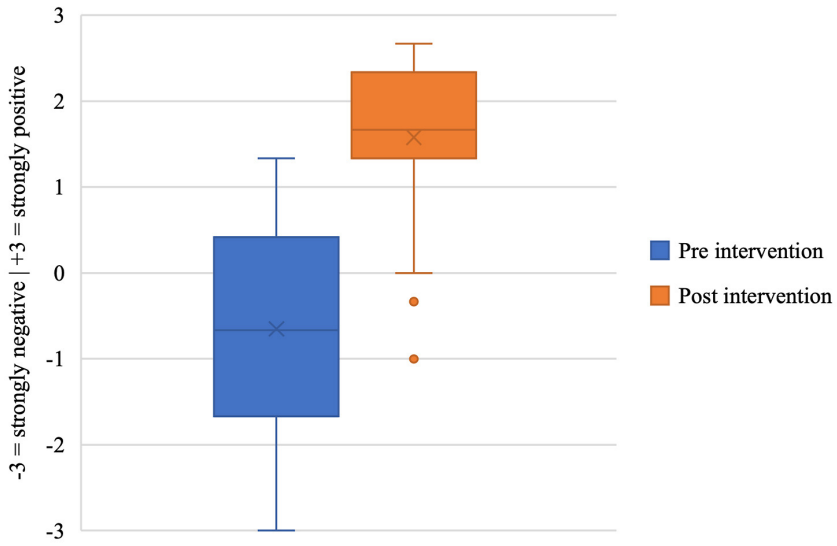
	T1	T2
Jacob	"boring"	"coding is really fun and interesting, I liked working with my friend on cool sounds"
Charlotte	"not sure sorry"	"I feel quite good, it was quite fun thanks!"
William	"OK"	"coding can be really cool, I feel really good"

Obr. 1.3: SonicPi testovanie (pred/po): programovanie dojmy

Na obrázku 1.4 môžeme vidieť výsledky ankety z testovania SonicPi. Študenti odpovedali na otázku, akú dôležitosť pripisujú schopnosti programovať. Modrou farbou sú odpovede pred vyskúšaním SonicPi a oranžovou po istom čase testovania.

Využitie Sonic Pi

Okrem širokej preddefinovanej sady zvukov umožňuje aplikácia vytvárať svoje vlastné zvuky a rytmy. Používa sa pri vyučovaní ale aj profesionálne na tvorbu hudby rôznych zvukových ukážok do filmov, videohier alebo iných digitálnych diel.



Obr. 1.4: SonicPi testovanie (pred/po): výsledky dotázníku

Výhody Sonic Pi

Z hľadiska náročnosti je jazyk prispôsobený širokej verejnosti a zvládli by ho používať aj žiaci základnej školy. Celkovo ide o pútavé interaktívne prostredie, kde výsledkom programovania je práve hudba.

Nevýhody Sonic Pi

Programovacie prostredie nie je kompatibilné s čitačmi obrazovky, nie je plne ovládateľné klávesnicou s neprimeraným množstom ovládacích prvkov. Jeho špecializovaný charakter môže byť obmedzením pre cieľovú skupinu používateľov. Z týchto dôvodov ho môžeme označiť za menej vhodný na vyučovacie účely.

Napriek tomu, že syntax jazyka je intuitívna a čitateľná pre bežných používateľov, obsahuje špeciálne znaky a bloky kódu sú v ňom automaticky formátované odsadením (ako v Pythone), čo by aj s použitím čitača obrazovky bolo náročné na orientáciu v kóde.

1.3.3 MusicBlocks

Music Blocks [2] je navrhnutý pre učiteľov a študentov, aby mohli preskúmať základné koncepty hudby v prostredí vizuálneho programovania.

Music Blocks bol vyvíjaný na základe Turtle Blocks, ktoré bolo inšpirované Logom na kreslenie umelecky efektných obrázkov prostredníctvom programovania. Obe aplikáie sú určené pre internetový prehliadač. Music Blocks rozširuje možnosti Turtle Blocks tým, že obsahuje sadu audio funkcií súvisiacich s výškou tónu a rytmom.

Obr. 1.5: Music blocks webové prostredie

Výhody MusicBlocks

Music Blocks je inovatívny a prospešný pre hudobné vzdelávanie z niekoľkých dôvodov: na jednej strane predstavuje novú metódu porozumenia základným konceptom hudby, na druhej strane je nástrojom na učenie programovania a logických zručností. Integruje hudobné a STEM (veda, technika, inžinierstvo, matematika) základy spôsobom, ktorý je zábavný, škálovateľný a autentický. [6].

Pretože je aplikácia webového charakteru, nie je potrebné inštalovať dodatočné nástroje.

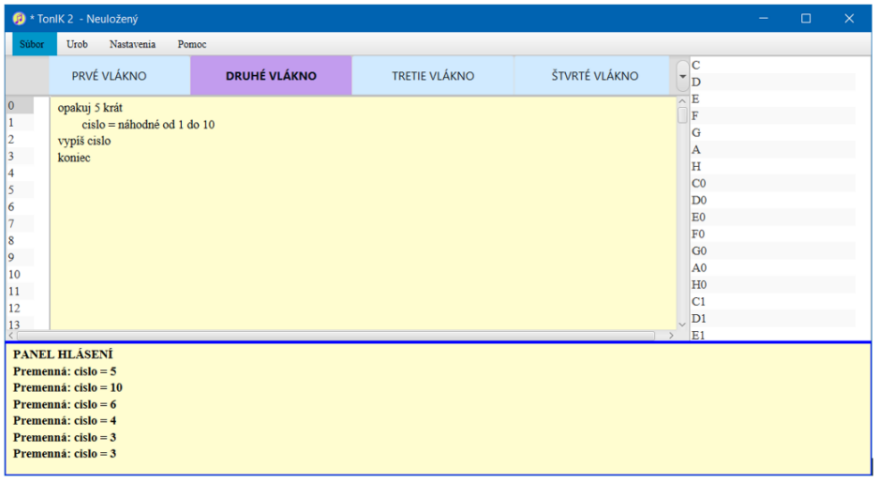
Taktiež to, že aplikácia je vyvíjaná nad Logom a jej určenie je podobné ako Logo, je vhodná pre mladšie vekové kategórie. Vedia sa v ňom prezentovať aj iné ako hudobné koncepty a princípy a priblížiť ich tam žiakom oveľa viac a skôr.

Nevýhody MusicBlocks

Ako bolo spomenuté, aplikácia je určená pre vizuálne programovanie. Obsahuje časti, ktoré sa síce dajú ovládať klávesnicou, no na prístup k nim je potrebné používanie myši. Ďalej, jej obsah nie je možné prezentovať pomocou čítača obrazovky. Aplikácia preto nie je vhodná pre používateľov so zrakovým postihnutím.

1.3.4 TonIK2

TonIK2 je aplikácia, ktorá vznikla ako diplomová práca [12]. Aplikácia umožňuje programovanie hudby a zvukov, ktoré je možné modifikovať a ovládať pomocou príkazov (na obrázku 1.6).



Obr. 1.6: TonIK2

Použitie TonIK2

Aplikácia umožňuje hudobné programovanie pre špecifickú skupinu používateľov, pre zrakovo postihnutých študentov základných škôl. Pomocou zvukov a hudobných prvkov sa snaží priblížiť a vysvetliť niektoré základné programovacie koncepty - cykly, vetvenia, používanie premenných a podprogramov, použitie vlákien, pre paralelný beh častí programu.

Tým, že je špecificky zameraná na skupinu zrakovo postihnutých žiakov, je dôležité, že dodržuje odporúčania a štandardy pre vývoj softvéru pre nevidiacich.

V nástroji sa programuje pomocou vlastného interpretovaného jazyka. Interpreter je napísaný v Jave a jazyk sa skladá z nasledovných typov príkazov:

- zvukové príkazy Tón, Postupnosť, Odzadu, Stupnica, Akord, Náhodný - tieto príkazy priamo prehrávajú zvuk,
- príkaz Zmeň nástroj - nastaví nástroj, ktorý ďalej prehrá zvukové príkazy,
- cyklus s pevným počtom opakovaní – umožňuje vytvárať jednoduché cykly so zadaným počtom opakovaní,
- príkazy vetvenia - na základe vyhodnotenia logickej podmienky vykonajú jednu alebo druhú vetvu programu.

1.3.5 Porovnanie riešení

Cieľom našej práce je výskumnými metódami za pomoci testovania implementovať čo najlepšie riešenia pre náš edukačný softvér. Jeho prvý testovací návrh vytvoríme podľa preštudovaných vedeckých článkov a v tejto podkapitole spomínaných existujúcich riešení 1.1.

Z nasledovných výsledkov pozorovania sa pokúsime vyvodiť, aké znaky by malo mať naše prostredie.

Porovnanie riešení

Aplikácia	Jazyk	Platforma	Tematika	Komplexnosť
Quorum	Založený na jazyku Java	Multiplatformový	Všeobecná	Mierne pokročilí
Sonic Pi	Ruby	Multiplatformový	Hudobná	Začiatočníci, pokročilí
MusicBlocks	Vizuálny programovací jazyk	Webová aplikácia	Hudobná	Začiatočníci
TonIK2	Inšpirovaný jazykom Logo	Multiplatformový	Hudobná	Začiatočníci

Tabuľka 1.1: Aplikácie - zhrnutie

Všetky aplikácie sú navrhnuté pre edukačné účely. Rozvíjajú predstavivosť a oboznamujú žiakov a študentov s princípmi programovania zaujímavou formou. Quorum je najbližší k rozšíreným programovacím jazykom ako sú Java, C#, Python, ale je obohatený o audio funkcie a prvky.

Všetky jazyky sú navrhnuté tak, aby obsahovali čo najmenej nealfanumerický znakov, ako sú zátvorky, bodkočiarky a podobne. Pracujú perfektne s čítačmi obrazovky. Prostredia sú jendoduché a intuitívne. Naplno využívajú variabilitu svojich nástrojov, a tak aj s menšou ponukou môže používateľ riešiť komplexné úlohy. Ak aplikácie obsahujú grafické prvky, tak sú veľké, kontrastné. Celé menu je obsluhovateľné klávesovými skratkami a do jednotlivých okien sa môže používateľ dostať príslušnou klávesou alebo kombináciou kláves.

Žiaci majú k dispozícii:

- premenné
- cykly
- vetvenia - podmienky
- jednoduché podprogramy
- spätnú väzbu v podobe audio výstupu (napr. prečítajú sa kontrolné výpisy)
- terminál - kontrola kódu a hlásenie chýb

V súčasnosti už existuje širšia ponuka programovacích prostredí, jazykov a nástrojov, ktoré sú prispôsobené potrebám nevidiacich programátorov. V tejto podkapitole uvedieme súčasný stavu riešenej problematiky doma aj v zahraničí. Niekoľko z nižšie menovaných bolo vyvíjaných priamo na našej fakulte.

Aplikácia TonIK2 je najviac blízka našej cieľovej skupine a aj jej určením.

Behové prostredie a jazyk

Rozdiel medzi našou aplikáciou a TonIK2 je v prvom rade jazyk. Náš jazyk je kompilovaný a beží na nami navrhnutej virtuálnej mašine, zatiaľ čo TonIK2 je interpretovaný.

Štruktúra kódu

Z toho vyplýva aj samotná syntax jazyka. V prostredí TonIK2 je potrebné jednotlivé kľúčové slová a príkazy písať vždy na nový riadok. V prípade, že sa na jeden riadok pokúsime napísať 2 príkazy, program zahlási chybu. Pretože interpreter vykonáva riadok po riadku, je potrebné mať takto štrukturovaný kód.

V našej aplikácií nie je dôležité ako je program napísaný. Dôležité je, aby bol syntakticky korektný. Môžeme tak zapísať viacero príkazov alebo kľúčových slov na jeden riadok, čo umožňuje žiakom si kód usporiadať tak, ako je im najlepšie.

Výstup aplikácie

TonIK2 umožňuje ukladať kód súboru do textového súboru, spätne ho načítať a prepisovať ho. Tento súbor je následne uložený na disk a je možné s ním manipulovať aj mimo prostredia.

Výstupom našej aplikácie je tiež textový súbor v ktorom je zapísaný kód programu. Je možné ho taktiež spätne načítať a manipulovať s ním a to tiež aj mimo aplikácie. Okrem toho, naša aplikácia umožňuje vytvoriť aj zvukový súbor vo formáte MIDI, ktorý je možné v prostredí Windows prehrať so štandardným zabudovaným multimediálnym prehrávačom.

Dodatočné potrebné inštalácie

TonIK2 je aplikácia vytvorená nad Javou a pre jej beh je potrebné mať program Java Runtime Environment (JRE) nainštalovaný, od verzie 1.8 vyššie. Spustenie tejto aplikácie tiež vyžaduje dodatočné nastavovanie alebo otvárať kontextovú ponuku (napríklad pravý klik na aplikáiciu myškou) a odtiaľ vybrať, ako sa má aplikácia spustiť, čo je menej bežný spôsob spúšťania aplikácií, menej intuitívny a pre niektorých zložitejší.

Naša aplikácia je vytvorená nad prostredím .Net. Toto prostredie je štandardnou súčasťou operačného systému Windows. Výstupom .NET aplikácií je spúšťateľný súbor (.EXE), takže aplikáciu môžeme jednoducho spustiť dvojklikom.

Môžeme ale naraziť na problém, kedy na počítači je iná alebo vôbec žiadna verzia .Net prostredia a je potrebné stiahnuť a nainštalovať novú. Vtedy operačný systém Windows používateľa upozorní na chýbajúce prostredie a vyzve používateľa aby si ho stiahol a nainštaloval. Používateľ nemusí hľadať na internete potrebné webové stránky a verzie, operačný systém mu tento link poskytne vo vyskakovacom okne spolu s upozornením.

Okrem toho, je možné aplikáciu doručiť ako natívnu binárnu aplikáciu, ktorá nepotrebuje prostredie .Net. Takáto aplikácie ale nemusí už byť kompatibilná medzi rôznymi počítačmi.

1.4 Popis použitých technológií

Pri vývoji programovacieho prostredia sme sa rozhodli využiť technológie, ktoré nám zabezpečia čo najlepšie riešenia pre prístupný softvér pre nevidiacich programátorov.

1.4.1 Jazyk C#

C# (C Sharp) je objektovo orientovaný programovací jazyk od spoločnosti Microsoft. Jazyk je vhodný pre rôznorodé typy aplikácií a je a kompatibilný s .NET. Vďaka tomu môžeme pristupovať ku množstvu knižníc. Medzi jeho výhody patrí bezpečnosť s automatickým čistením pamäti, objektová orientácia a čitateľnosť. Je obľúbený pre vývoj desktopových, webových, mobilných aplikácií a hier, vďaka čomu je široko využívaným jazykom.

1.4.2 .NET

.NET je open-source platforma od Microsoftu. Ide o univerzálny nástroj na vývoj aplikácií, ktorý je kombinovateľný s viacerými programovacími jazykmi ako sú C# alebo C++/CLI. Okrem Windows, Linux a macOS ním môžeme vyvíjať aplikácie aj pre Android a iOS.

1.5 Vývoj softvéru pre nevidiacich študentov

Ako sme už spomenuli v sekcii 1.1.1, rôzne postihnutia majú rôzne požiadavky na funkčnosť.

Spoločným problémom je väčšiny programovacích jazykov je, že sa snažia byť na všeobecné použitie a ich syntax je zameraná na čísla a symboly. Na riešenie tohto problému je potrebné dodržovať praktiky a odporúčania.

1.5.1 Požiadavky na softvér pre nevidiacich

V tejto časti si popíšeme odporúčania a požiadavky [14, 9] na tvorbu jazyka, prostredia a softvéru, ktorý je určený pre používateľov so zrakovým postihnutím.

- Nevidiaci

  Pretože nevidiaci sú závislí od čítačov obrazovky, nie je potrebné aby bol softvér vizuálne príťažlivý. Musíme však dbať na to, aby bol softvér ovládateľný klávesnicou a klávesovými skratkami.

Spolu s tým je dôležité aby boli prístupné ovládacie prvky (tlačidlá, menu, ponuky, okná a pod.), ktoré sú pre používateľa dôležité, aby sme tak zabránili neočakávanej manipulácií s elementami.

Okrem dostupnosti ovládacích prvkov je potrebné aj ich správne, výstižné a konkrétne pomenovanie. Napríklad popis prvku ”my textbox” používateľovi nepovie čo sa do daného textového poľa píše. Ak ho popíšeme ”textové pole s kódom” je jasnejšie načo je určené.

Taktiež treba brať do úvahy škálu nastavení a prispôsobení, ktoré pre používateľov sprístupníme. Napríklad nemá zmysel pre nevidiaceho používateľa poskytovať nastavenie farebnej schémy aplikácie alebo veľkosť písma.

Pretože syntax jazyka bude čítaná čítačom obrazovky, je potrebné minimalizovať špeciálne symboly (bodky, zložené zátvorky a pod.), nestavať bloky kódu pythonovou syntaxou (blok kódu je tvorený odsadením). Možnosť používať skrátené slová namiesto celých kľúčových slov jazyka (napr. ”repeat” skrátiť na ”rpt”), ušetrí používateľom čas najmä v prípade, kedy sa opakovane číta celý text kódu.

Je dôležité tiež brať do úvahy, že čítač obrazovky štandardne nerozlišuje veľké a malé písmená. Toto nastavenia je možné prepnúť, môže však spôsobiť náročnejšie spracovanie hovoreného slova a porozumeniu. Preto by jazyk nemal rozlišovať malé a veľké písmená a stavať syntax a konvencie na nich.

- Čiastočne vidiaci a slabozrakí

  Slabozraký oproti nevidiacemu používateľovi má čiastočný pohľad na našu aplikáciu. S tým súvisí aj jej prispôsobiteľnosť.

  Slabozrakí sú schopní čítať sami pomocou zväčšovacích nástrojov. Je preto potrebné v softvéroch dodržiavať dostatočný kontrast písma a pozadia.

  Pri niektorých úrovniach postihnutia s čiastočným videním nemusí byť ani potrebné používať zväčšovacie nástroje, stačí keď je možné zväčšovať písmo textu.

  Okrem veľkosti, je dôležitý aj samotný font. Je dôležité myslieť na to, že v niektorých fontoch sú niektoré znaky ľahko zameniteľné. Najčastejším príkladom je písmeno malé `l` a číslo `1`. Odporúča sa používať bezpätkový font, ktorý neobsahuje vizuálne dekorácie a je čitateľnejší.

  Je dobré zvážiť aj rôzne farebné schémy aplikácie, ktoré prispôsobujú kontrast písma voči pozadiu a zameriavajú sa na problém s vnímaním farieb.

  Odporúča sa vyhýbať dekoráciám v aplikácií. Vyskakovacie polo-priesvitné okná nie sú vhodné, nakoľko je ťažké ich vidieť keď je obrazovka zväčšená alebo zameraná na časť, kde sa toto okno nenachádza.

Animované alebo blikajúce prvky môžu pôsobiť príjemne a pútavo, avšak pri ich nadmernom použití môžu byť rušivé, zvlášť pre slabozrakého používateľa, ktorému môžu odvádzať pozornosť.

1.6 Kompilátor a interpreter

Aby sme objasnili princíp a fungovanie nášej aplikácie, najprv si ujasníme pojmy kompilátor a interpreter a ich vzťah s programovacími jazyky.

Programovací jazyk je vo svojej podstate iba súbor pravidiel, ako sa zapisuje program. V samotnej textovej (alebo inej) podobe však nie je spustiteľný. Podľa spôsobu spracovania, vyhodnotenia a vykonávania zaraďujeme programovacie jazyky do 2 kategórií - kompilované a interpretované.

1.6.1 Kompilátor

Kompilátor [7] je program, ktorý spracuje text iného programu a preloží ho. Podľa toho, aký je cieľ preloženého kódu, môže byť kód preložený buď do strojového kódu, ktorý je možné vykonávať priamo na procesore, alebo do medzikódu (bajtkódu), ktorý nie je vykonávaný priamo procesorom, ale virtuálnou mašinou (strojom).

Bajtkód, medzikód a virtuálna mašina

Kód kompilovaný priamo do strojových inštrukcií je náročný na prenos medzi rôznymi systémami. Na riešenie tohto problému boli zavedené virtuálne mašiny, ktoré slúžia na preklad bajtkódu do strojových inštrukcií.

Kód programu sa neprepisuje do strojového kódu, ale je preložený do inštrukcií a do postupnosti bajtov pre virtuálnu mašinu a nie pre procesor.

Virtuálna mašina pozná presnú architektúru počítača a teda vie ako má bajtkód preložiť tak, aby bolo možné ho priamo vykonávať na procesore.

Tak je možné už skompilovaný program prenášať medzi rôznymi systémami a teda programátor nemusí riešiť problém s kompatibilitou, stačí ak cieľový počítač má nainštalovaný virtuálny stroj, ktorý sa stará o preklad pre konkrétny systém.

Najznámejšie firmy využívajúce takýto spôsob kompilácie sú Oracle a jazyk Java alebo Microsoft a jazyk C#.

Strojový kód

Kompilovaný kód môže byť však preložený aj do jazyka assembly (asemblér), ktorý sa dá priamočiaro prepísať do postupnosti bajtov. Tieto inštrukcie sú však zviazané s nastavením počítača a jeho architektúrou a je možné ich priamo vykonávať na procesore.

Ak má koncový používateľ iný operačný systém, iné nastavenia alebo iný procesor, je pravdepodobné, že takto skompilovaný kód nebude možné u neho spustiť, alebo počas behu môže nastať nechcené správanie programu.

1.6.2 Proces kompilácie

Spracovanie a preklad kódu prebieha v niekoľkých krokoch, bez ohľadu na to, či bude spúšťaný priamo na procesore alebo bude preložený do bajtkódu.

Lexikálny analyzátor

Prvým krokom kompilácie, je pretavenie textu kódu na postupnosť tokenov.

Text kódu sa po jednotlivých znakoch prečíta a z jednotlivých slov (lexém) sa vytvoria tokeny.

Tokeny si môžeme predstaviť ako kľúčové slová, špeciálne zápisy alebo znamienka matematických operácií.

V tomto kroku sú z kódu odstranené vodiace (white-space) znaky, komentáre a je vyhodnocovaná správnosť syntaxe jazyka. V prípade chybnej syntaxe (syntax error) môžeme upozorniť používateľa a ďalšie spracovanie ukončiť.

O takéto spracovanie textu sa stará mechanizmus lexikálny analyzátor.

Syntaktický analyzátor

Jednotlivé tokeny, alebo lexémy, sa vyhodnotia syntaktickým analyzátorom a spoja sa do väčších programových konštrukcií, ako je cyklus, vetvenie, skoky, matematické operácie a iné.

Výsledkom syntaktickej analýzy je syntaktický strom. Jedná sa o dátovú štruktúru, ktorá zachytáva postupnosť programových konštrukcií a ich poradie volania, parametre, optimalizovať program a iné.

Generovanie kódu

Potom, ako máme vytvorený syntaktický strom, musíme z neho vygenerovať spustiteľný kód.

Tým, že syntaktický strom udržuje postupnosť programových konštrukcií a ich hierarchiu, vieme z neho jednoducho rekurzívne vygenerovať bajtovú reprezentáciu tohto stromu a teda celého programu, či už pre procesor alebo virtuálnu mašinu.

Vykonávanie programu

Ak je vygenerovaný kód určený priamo pre procesor, v tomto kroku sa pre program pripravia a priradia registre, vyhradí operačná pamäť a inštrukcie sa postupne posúvajú do

procesora, ktorý ich vykonáva.

V prípade, že ich vykonáva virtuálna mašina, nastaví sa jediný pseudoregister (v skutočnosti sa jedná o jednoduchú premennú), ktorý zastupuje úlohu registra PC (program counter). Ten sa posúva po vyhradenej pamäti a vykonáva inštrukcie, ktoré daný bajt na danom mieste predstavuje, čím simuluje chod reálneho počítača.

1.6.3 Interpreter

Narozdiel od kompilácie, sa kód spracovaný interpreterom neprekladá do strojového kódu, ale jednotlivé riadky (prípadne bloky) kódu sú postupne po jednom spracované a vykonané [7].

Takto je možné manipulovať s programom za jeho behu. Jeho vykonávanie je však výrazne pomalšie v porovaní s kompilovaným jazykom.

Okrem toho, ak sa v programe vyskytuje nejaká chyba, lexikálny analyzátor nás na to neupozorní až kým sa nepokúsi daný kus kódu vykonať.

1.6.4 Proces interpretácie

Lexikálny analyzátor

Podobne ako v prípade kompilácie sa v tomto kroku text prechádza po jednotlivých znakoch a budujú sa tokeny.

Rozdiel je však v tom, že interpreter nečíta celý kód naraz. Spracuje jeden token naraz a podľa neho okmažite zavolá vykonávanie príkazu podľa hodnoty tokenu a po jeho vykonaní sa posunie ďalej.

Syntaktický analyzátor

V prípade interpretera, syntaktický analyzátor na základe tokenu vytvorí objekt, ktorý reprezentuje daný príkaz. Namiesto vytvorenia celého syntaktického stormu tak analyzátor vytvorí jeden objekt príkazu a vykoná ho. Po jeho vykonaní sa presunie na ďalší token a cyklus sa opakuje.

Naša aplikácia bude bežať na virtuálnej mašine, ktorú si zhotovíme. Máme tak možnosť upozorniť používateľa na chyby ešte pred spustením kódu, v prípade budúcej potreby vieme celý program veľmi uložiť do binárneho súboru (serializovať ho) a prípadne ho znova rýchlo načítať, čím vieme zabezpečiť prenos a kompatibilitu medzi rôznymi počítačmi.

Vzhľadom na cieľovú skupinu a náročnosť ich programov a aplikácií však takúto implementáciu nevyžadujeme a program budeme ukladať ako text a ako MIDI výstup.

Kapitola 2

Návrh

V tejto kapitole opíšeme návrh a implementáciu nami navrhnutého programovacieho prostredia MusIK. Budeme sa pri tom opierať o zdrojové články. Tie popisujú ako vizuálnu stránku prostredia, tak aj syntaktický návrh jazyka.

2.1 Návrh používateľského rozhrania

Používateľské rozhranie (user interface - UI) musí brať ohľad na cieľového používateľa, prípadne cieľovú skupinu, a podľa toho sa prispôsobiť. Vzhľadom na našu cieľovú skupinu musíme zabezpečiť dostupnosť potrebných funkcií, jednoduchú a rýchlu orientáciu a ovládanie. Na to využijeme ovládanie pomocou klávesových skratiek.

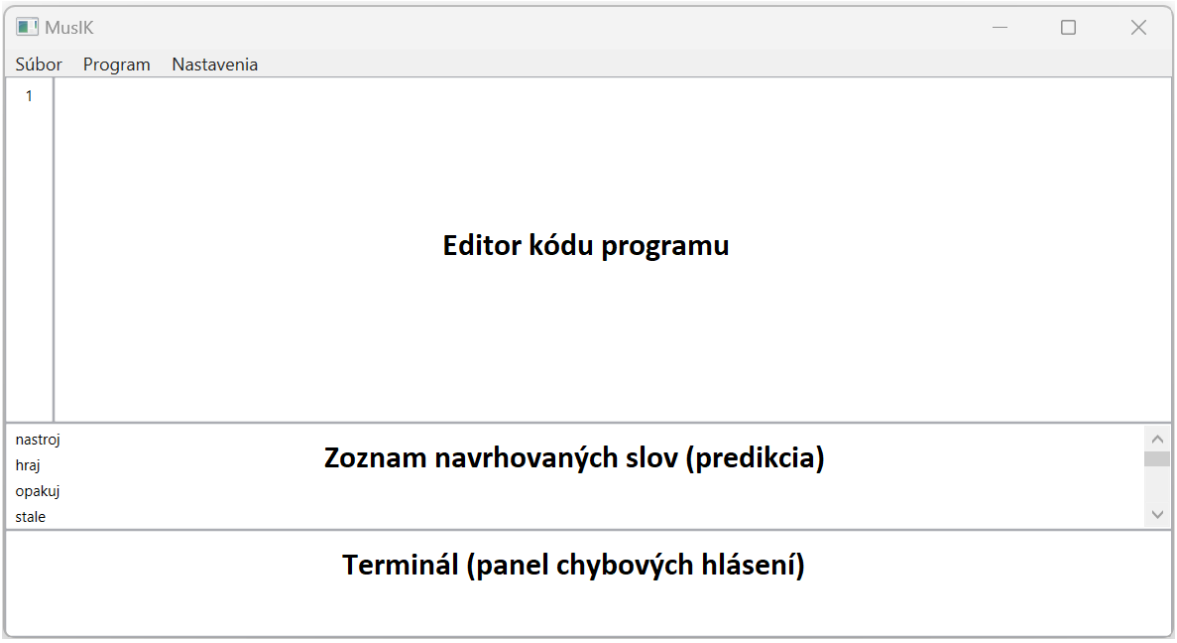
Pri tomto je dobré používať už známe a zaužívané klávesové skratky, čo eliminuje potrebu používateľa sa nanovo učiť ovládanie, ktoré sa štandardne používajú v iných aplikáciach. Napríklad skratky na ukladanie, otvorenie nového súboru, zatváranie a ukončenie a podobne.

Pretože zrakové postihnutie pokrýva rôzne štádiá a stupne vnímania pomocou zraku, by sme mali poskytnúť aspoň minimálnu možnosť prispôsobenia vzhľadu - farebnými schémami, možnosťou zväčšovania a zmenšovania obsahu a zmenou veľkosti hlavného okna.

2.2 Rozloženie user interface-u

2.2.1 Hlavné okno

Najdôležitejším prvkom nášho UI je hlavné okno. Jeho obsah je ďalej rozložené tak, aby sa podobal rozloženiu populárnych programovacích prostredí a štandardnému rozloženiu okien aplikácií v operačnom systéme Microsoft Windows. Teda je rozložený vertikálne tak, aby sa na spodku okna nachádzal chybový panel, najväčšiu časť zaberá editor kódu, na vrchu okna sa nachádza menu.



Obr. 2.1: Rozloženia elementov v hlavnom okne

Menu

Najvyššie v okne je uložený element menu. V ňom má používateľ prístup ku kategóriám položiek, pričom sme sa opäť snažili čo najviac priblížiť k organizovaniu menu, ako býva štandardne vo windowsových aplikáciach.

Menu býva zvyčajne dostupné po stlačení klávesy ALT, čo je implementované aj v tejto aplikácií.

Tieto kategórie organizujú položky menu do skupín, ktorá zahrňuje funkcionalitu s jednou časťou programu:

- Súbor

  Táto skupina zastrešuje akcie spojené so súborom a kódom programu. Poskytuje možnosť otvoriť nový súbor, otvoriť už existujúci súbor, uložiť aktuálne otvorený súbor, uložiť súbor ako, čo poskytuje možnosť otvorený súbor uložiť ako kópiu a vytvoriť zvukový spustiteľný výstup vo formáte MIDI, a ukončenie aplikácie. Ovládanie je prispôsobené bežným klávesovým skratkám vo Windowse:

  - Nový - CTRL + N
  - Uložiť - CTRL + S
  - Uložiť ako - CTRL + SHIFT + S
  - Ukončiť - ALT + F4

- Program

V skupine Program sa nachádza funkcionalita pracujúca s aktuálny programom, prípadne poskytujú všeobecné nápovedy k programu. Nájdeme tu možnosť spustenia alebo zastavenia bežiaceho programu a možnosť Pomoc, ktorá zobrazí vyskakovacie okno s nápovedou, kde si vieme prečítať syntax jazyka a prípadné vysvetlenie k príkazom.

Ovládanie spustenia sa líši od prostredia k prostrediu, preto nie je štandardná klávesová skratka na spustenie programu v rôznych vývojových prostrediach. Zvyčajne však táto možnosť býva nastavená na klávese F5, F6, F9 alebo F10. My sme zvolili možnosť spúšťania klávesou F5, nakoľko sme používali pri vývoji nástroj Visual Studio od Microsoftu v ktorom sa program spúšťa práve pomocou klávesy F5.

- Nastavenia

  Nastevenia sú skupina, ktorá obsahuje možnosti vypnutia alebo zapnutia tmavého režimu. Toto nastavenie je určené pre tých, ktorý majú aspoň čiastočný zrak a znižuje tak zaťaženie na oči. V budúcich verziách je možné doplniť ďalšie nastaviteľné možnosti a rozšírenia.

Editor kódu

Editačný panel s kódom zaberá najväčšiu časť okna, nachádza sa v strede okna a umožňuje písať kód. Do panela je možné vkladať odsadenia pomocou tabulátora, v prípade že používateľ má aspoň čiastočný zrak, čo poskytuje vizuálne organizovať kód. Syntax jazyka však takúto organizáciu kódu nevyžaduje a je možné ju ignorovať.

Vedľa editoru kódu sa nachádza číselník riadkov, ktorý uľahčuje orientáciu v kóde, prípadne vyhľadávanie chyby z terminálu.

Ponuka nápovedy

Panel s ponukou nápovedy (predikcie) je panel, ktorý na základe napísaného vstupu filtruje zo zoznamu príkazov, tie ktoré sa na daný vstup podobajú. Podobnosť vstupu a príkazov určujeme pomocou regulárnych výrazov (regexov). Ako vzor hľadania sa používa časť slova pred kurzorom. V prípade že sa kurzor nachádza na medzere alebo je kód programu prázdny, zobrazuje všetky príkazy.

Terminál

Terminál (panel s hláseniami) sa nachádza v hlavnom okne v spodnej časti. Slúži na výpis chybových hlášok alebo na výpis hodnôt premenných.

2.2.2 Vyskakovacie a dialógové okná

Vyskakovacie (označované aj ako *pop-up*) okná sú okná s informačným obsahom. Názov *vyskakovacie* znamená, že okno sa nanovo inicializuje a zobrazí, teda pri bežnom používaní aplikácie nie je vytvorené a ani zobrazené.

Okno s návodom

Po stlačení klávesy F1 sa používateľovi zobrazí okno; v jeho tele sa nachádza nachádza zoznam príkazov, ich popis, vysvetlenie a ukážky ich použitia. Správaním funguje podobne ako okno v starších windowsových aplikáciach, kedy stlačením F1 sa zobrazilo takzvané *help window*. V novších verziách operačného systému Windows sa väčšinou zobrazí internetová stránka s návodom a popisom aplikácií. Toto okno zatvoríme stlačením ľubovoľnej klávesy.

Dialógové okná

Dialógovým oknom označujeme také okno, prostredníctvom ktorého si do používateľa do aplikácie pýtame vstup. Najčastejšie získavanie cesty k súborom, názvy a podobne. V našej aplikácií používame dialógové okná práve na získanie cesty k súborom, či už pre načítanie súboru alebo jeho uloženie.

2.3 Návrh syntaxe aplikácie MusIK

Na tvorbu programov v našej aplikácií budeme používať jazyk, ktorý sme sami navrhli, využívajúc odporúčania (viď 1.5.1), aby bol náš jazyk dobre použiteľný pre našu cieľovú skupinu.

Nevyhneme sa však úplnému použitiu špeciálnych symbolov. V syntaxy nášho jazyka budeme pri predávaní parametrov využijú dvojbodky a pri generovaní náhodného čísla využijeme zátvorky.

Syntax bola zamýšľaná taktiež tak, aby pripomínala vety v slovenčine, čím sa žiakom ľahšie zapamätajú, spracujú a v prípade potreby upravia. Interpunkčné znamienka však v jazyku nepoužívame.

Program tvoríme pomocou príkazov. Tie môžu, ale nemusia obsahovať diakritické znamienka (mäkčeň, dĺžeň, vokáň, prehláska a iné.) Jediné miesto kde striktne rozlišujeme diakritiku je v názvoch podprogramov a premenných. Tie by mali byť unikátne pomenované, preto v nich diakritiku považujeme za dôležitú, veľkosť písma však nie je dôležitá.

Uvedieme si zoznam príkazov, ktorými môžeme tvoriť programy:

2.3.1 Príkazy jazyka

V tejto časti si uvedieme zoznam príkazov, popíšeme jeho funkcionalitu a ku každému si predstavíme príklad jeho zápisu a použitia.

Príkaz hraj

Základným prvkom našej aplikácie je prehrávanie tónov. Príkaz k ich prehraniu vieme zadať pomocou slova hraj tón, kde tón predstavuje pomenovanie niektorého tónu v rozsahu C1 až C3, vrátane.

Tomuto príkazu môžeme predať aj nepovinné parametre, ktoré menia dĺžku a hlasitosť prehratého tónu: parameter d: označuje modifikátor dĺžky tónu, parameter h: nastavujeme hlasitosť od v rozmedzí o 0 do 100. Dĺžka tónu sa však neurčuje hudobne v taktových jednotkách (doba, pol doby, celá doba a podobne) alebo určuje sa v milisekundách (ms).

Ako bolo spomenuté, tieto parametre nie sú povinné a tak k tomuto príkazu máme prednastavené hodnoty: 500 ms pre dĺžku, 100 pre hlasitosť.

Dvojbodky v tomto prípade spôsobia oddelenie prečítanej hodnoty a čísla. Čítač v tomto prípade teda hlasovo odlíši znak *d, h* od čísla a neprečíta ich ako spojené označenie.

Príklad príkazu hraj:

```
hraj c1 h:100 d:500
```

Príkaz nástroj

Príkazom *nástroj* nastavíme zvuk nástroja, ktorý prehráva tóny nasledujúce za týmto príkazom. Máme k dispozícií niekoľko možností zvukov, z ktorých si môžeme vybrať:

- husle
- bicie
- gitara
- organ
- spev, hlas
- trubka
- harfa
- akordeón
- flauta
- klavír, piano

Príklad príkazu nástroj:

```
nástroj hlas
```

Príkaz náhodný

Príkazom *náhodný* vieme vygenerovať náhodný tón v intervale od C1 do C3. V prípade, že vynecháme parametre *d:* a *h:*, bude mať tón aj náhodne vygenerovanú dĺžku a hlasitosť. Dĺžka vygenerovaného tónu je však obmedzená od 250 do 2500 milisekúnd.

Príklad príkazu náhodný bez parametrov aj s parametrami:

```
náhodný
náhodný d:500 h: 80
```

Príkaz pauza

Príkazu *pauza* predávame jediný parameter - dĺžka. Tým že je jediný neoznačujeme ho pomocou *d:*.

Príkaz spôsobí, že v danom vlákne pozastavíme prehrávanie tónov na danú dĺžku doby.

Tento parameter môžeme predať aj pomocou premennej. Fungovanie premenných a vlákien si vysvetlíme neskôr.

Príklad príkazu náhodný bez parametrov aj s parametrami:

```
pauza 500
```

Príkaz akord

Akord je príkaz, ktorým môžeme zahrať 2 až 4 tóny naraz. Za týmto príkazom nasledujú názvy tónov, oddelené medzerou. Taktiež je možné predávať parametre pre dĺžku a hlasitosť.

Príklad príkazu akord:

```
akord G1 H1 D2 G2 h:100 d:500
```

Príkaz výpis

Na vypísanie hodnoty niektorje našej premennej môžeme použiť príkaz *výpis* za ktorým nasleduje názov premennej.

Príklad príkazu náhodný bez parametrov aj s parametrami:

```
cislo = 12
výpis cislo
```

Príkaz losuj

Ak chceme premennej vygenerovať náhodnú hodnotu, alebo v rámci niektorej programovej konštrukcie chcem použiť náhodne vygenerované číslo, môžeme na to použiť príkaz *losuj*. Za týmto príkazom uvádzame interval pomocou 2 čísel, oddelenými čiarkami. Navyše tento interval musí byť uzavretý zátvorkami.

Uvedený interval je uzavretý, teda funkcia môže vygenerovať čísla vrátane tých uvedených v zátvorkách.

Príklad príkazu náhodný bez parametrov aj s parametrami:

```
kocka = losuj(1, 12)
výpis kocka
```

2.3.2 Programové konštrukcie

Programové konštrukcie sú základným riadiacim prvkom programov. Vďaka programovým konštrukciám rozhodujeme a riadime vykonávanie programu - môžeme jeho beh rozvetviť (*vetvenie programu*), opakovane vykonávať niektoré kroky programu (*cyklus*), opakovane používanú funkcionalitu zabaliť do podprogramov (*funkcia, procedúra*) a do premenných ukladať dáta (*premenné a príkaz priradenia*).

Okrem týchto štandardných programových konštrukcií v našej aplikácií predstavíme aj koncept vlákien (thread), ktoré nám umožnia súbežne vykonávať viacero krokov programu naraz.

Príkaz opakuj

Pomocou *opakuj* vieme zaviesť opakované vykonávanie príkazov a vytvoriť tak cyklus. Môžeme vytvoriť cyklus s daným počtom opakovaní, alebo cyklus ovládaný podmienkou.

Ak je cyklus riadeným počtom opakovaní, musí za výrazom nesúcim informáciu o počte nachádzať slovo *krát*. Výraz môže byť zadaný buď konštantou alebo premennou. Ak použijeme premennú, jej hodnota sa po ukončení cyklu nezmení.

Telo cyklu pozostáva z ďalších príkazov, ktoré sa budú vykonávať postupne zadaný počet krát; telo cyklu musí byť ukončené slovom *koniec*.

Príklad cyklu:

```
opakuj n krát
telo cyklu
koniec
```

Konštrukcia vetvenia

Vetvením programu vieme vytvoriť vetvy - rôzne postupnosti príkazov, z ktorých vieme vykonať buď jednu alebo druhú. Voľba vetvy sa vykonáva na základe podmienky. Koncept

vetvenia môžeme v rôznych literatúrach nájsť pod označením *if-else* alebo *ak-tak*. Vetva *else* pritom nie je povinná, teda sa môže stať, že počas behu programu sa celý blok úplne preskočí.

Vytvoríme teda dva bloky (postupnosti príkazov), ktorých telá sú opäť ukončené slovom *koniec*. Prvý blok začína slovom *ak* alebo *ked'*, za ktorým nasleduje podmienka. Tento blok je pre vytvorenie vetvenia povinný. Blok pre prvú vetvu ukončíme slovom *koniec*.

Druhá vetva - *inak*, nie je povinná. Vykoná sa iba v prípade, že podmienka nie je splnená a vynechá sa celá prvá vetva. Začína sa slovom *inak* a opäť končí slovom *koniec*.

Príklad vetvenia:

```
n = losuj(1, 10)
ak n > 5
nástroj husle
koniec
inak
nástroj gitara
koniec
```

Konštrukcia podprogramov

V programovaní sa môžeme dostať do situácie, kedy potrebujeme vykonávať tú istú postupnosť príkazov na rôznych miestach programu. Aby sme nemuseli opakovane písať tieto príkazy, môžeme takúto postupnosť pomenovať a v prípade potreby vyvolať. Vytvoríme tak *podprogram*.

V programovaní rozlišujeme 2 rôzne typy podprogramov - tie ktoré vracajú nejakú hodnotu alebo výsledok nazývame *funkcie*, tie ktoré výsledok nevracajú nazývame *procedúry*.

V našej aplikácií vieme zadeklarovať procedúru pomocou slova *urob* alebo *podprogram*, za ktorým nasleduje názov tejto postupnosti.

Aby bola naša aplikácia jednoduchá s ohľadom na potreby našej cieľovej skupiny a s ohľadom ňu ako takú, všetky procedúry sú bezparametrické, teda nie je možné do nich predávať parametre.

Nasleduje postupnosť príkazov a telo podprogramu opäť nasleduje ukončovacie slovo *koniec*.

Zapísaním takejto postupnosti sa podprogram nevykoná. Na vykonanie takejto postupnosti ju treba vyvolať, zadaním jej názvu.

Príklad podprogramu a jeho vyvolania:

```
urob stupnicaCDUR
hraj c
hraj d
hraj e
```

```
hraj f
hraj g
hraj a
hraj h
hraj c2
koniec
<rôzne ďalšie príkazy>
stupnicaCDUR
```

Konštrukcia vlákno

Vlákno predstavuje postupnosť príkazov, ktoré sa vykonávajú nezávisle od príkazov v inom vlákne. To nám poskytuje možnosť, napríklad hrať viacero tónov s rôznymi nástrojmi súčasne. Predstavuje veľmi zjednodušený a obmedzený koncept z iných jazykov, ktoré nám umožňujú asynchrónne alebo viacvláknové programovanie - *thread*.

Naša aplikácia je obmedzená na štyri vlákna, kvôli efektivite a z praktických dôvodov.

Deklarovanie vlákna označíme slovom *vlákno* a ukončíme slovom *koniec*.

Príklad vlákna:

```
urob vlaknoPrve
nástroj gitara
hraj c
hraj e
hraj g
koniec

urob vlaknoDruhe
nástroj piano
hraj c
hraj e
hraj g
koniec
```

Premenné

Premenné v našej aplikácií používame na uloženie číselného údaju.

Môžeme ho použiť ako počítadlo v cykloch alebo v kontextoch podmienok, napríklad pri vetvení alebo v cykloch s podmienkou.

Opäť, s ohľadom na našu cieľovú skupinu a jej potreby, všetky premenné ktoré počas behu programu vzniknú, sa nachádzajú v globálnom mennom priestore, teda keď premenné

vzniknú, vieme k nim pristupovať kdekoľvek v aplikácií. Aj keď premenná vznikne vo vnútri procedúry, nebude zaradená do lokálneho menného priestoru, ale do globálneho.

Premenná vznikne v momente priradenia hodnoty do nej. Okrem konštanty môžeme do nej priradiť aj náhodné číslo pomocou funkcie *losuj*.

Príklad vytvorenia premennej a jej použitia:

```
premenna = losuj(1, 20)
opakuj premenna krát
<telo cyklus>
koniec

ak premenna < 10
<príkazy ak je podmienka splnená>
koniec
inak
<príkazy ak nie je podmienka splnená>
koniec
```

Kapitola 3

Výskum

Súčasťou tejto diplomovej práce je výskum. Predmetom výskumu je použiteľnosť nami vytvoreného programovacieho jazyka a prostredia. Existujú štandardy, podľa ktorých je potrebné softvér pre nevidiacich študentov navrhnúť. Nehovoria však o tom ako sa má používať.

Výskum bude bude prevedený formou UX & usability research [10]. Je dôležitý pre celkový návrh dizajnu aplikácií alebo systémov. Jedná sa o spôsob výskumu, ktorý sa zameriava na správanie, preferencie a celkovú používateľskú skúsenosť cieľovej skupiny.

Výskum prebieha v iteráciach. Cieľovej skupine dáme softvér otestovať, zadáme im úlohy a popritom sledujeme ako si s úlohami vedeli alebo nevedeli poradiť.

Následne po testovaní získame spätnú väzbu, buď pomocou dotazníka, rozhovoru alebo pozorovania.

Následne spätnú väzbu zapracujeme a opäť dáme zmeny otestovať. Je na zvážení, či by skupina mala byť iná ako v predošlej iterácií alebo rovnaká. Používatelia majú tendenciu si aplikáciu pamätať a teda ich ovládanie už im môže byť známe.

3.1

Záver

Literatúra

[1] https://sonic-pi.net/, Accessed: 12.11.2023.

[2] https://musicblocks.sugarlabs.org/, Accessed:12.12.2023.

[3] Jaws. https://www.freedomscientific.com/products/software/jaws/, Accessed: 17.11.2023.

[4] Nvda. https://www.nvaccess.org/about-nv-access/, Accessed: 16.11.2023.

[5] Quorum programming language. https://quorumlanguage.com/, Accessed: 17.11.2023.

[6] Walter Bender, Devin Ulibarri, Ymca Malden, and Yash Khandelwal. Music blocks: A musical microworld. 2015.

[7] PhD. doc. RNDr. L'ubomír Salanci. https://edu.fmph.uniba.sk/~salanci/Kompilatory/index.html, Accessed: 12.12.2023.

[8] Texas School for the Blind and Visually Impaired. Braille display devices. https://www.tsbvi.edu/statewide-resources/services/braille/display, Accessed: 17.11.2023.

[9] Alex Hadwen-Bennett, Sue Sentance, and Cecily Morrison. Making programming accessible to learners with visual impairments: A literature review. *International Journal of Computer Science Education in Schools*, 2, 05 2018.

[10] Interaction Design Foundation IxDF. What is ux research? https://www.interaction-design.org/literature/topics/ux-research, Accessed: 1.12.2023.

[11] Lopúchová Jana. Základy pedagogiky zrakovo postihnutých.

[12] Iveta Krempaská. Výučbové prostredie na programovanie melódií prístupné pre nevidiacich žiakov zŠ. diplomová práca, FMFI UK, 2022.

[13] Christopher Petrie. Programming music with sonic pi promotes positive attitudes for beginners. *Computers Education*, 179:104409, 2022.

[14] Jaime Sánchez and Fernando Aguayo. Apl: Audio programming language for blind learners. In Klaus Miesenberger, Joachim Klaus, Wolfgang L. Zagler, and Arthur I. Karshmer, editors, *Computers Helping People with Special Needs*, pages 1334–1341, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[15] Andreas Stefik, Christopher Hundhausen, and Derrick Smith. On the design of an educational infrastructure for the blind and visually impaired in computer science. *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 05 2011.

Príloha A: obsah elektronickej prílohy

Príloha B: Používateľská príručka