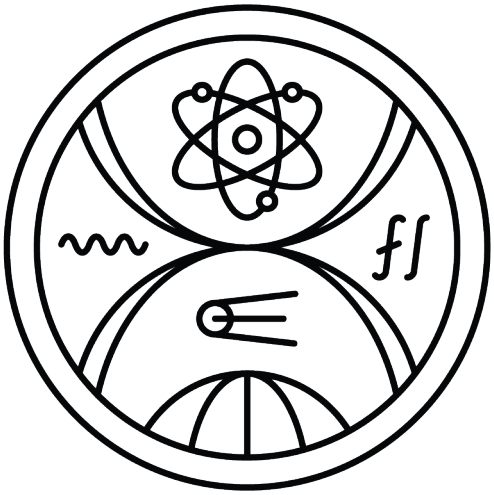
UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



# PROGRAMOVANIE VIRTUÁLNEHO OBJEKTU PRÍSTUPNÉ PRE NEVIDIACICH ŽIAKOV SEKUNDÁRNEHO VZDELÁVANIA

DIPLOMOVÁ PRÁCA

2023
BC. ANNA REBEKA SOJKA

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

# PROGRAMOVANIE VIRTUÁLNEHO OBJEKTU 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. Anna Rebeka Sojka


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc. Anna Rebeka Sojka
**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:** Programovanie virtuálneho objektu prístupné pre nevidiacich žiakov sekundárneho vzdelávania
*Virtual object programming accessible to secondary blind students*

**Anotácia:** Autorka vytvorí plne ozvučené edukačné programovacie prostredie s vlastným kompilátorom alebo interpreterom. Základné príkazy zabudovaného programovacieho jazyka budú slúžiť na presun virtuálneho objektu v požadovanom smere. Tieto príkazy bude možné použiť aj v rámci komplikovanejších štruktúr, ako je cyklus, príkaz vetvenia, podprogram. Okrem toho bude možné pracovať s celočíselnými premennými. Editor kódu bude mať zabudovanú kontrolu syntaxe a funkciu ponuky príkazov.
Výsledná aplikácia musí spolupracovať s čítačom obrazovky (napríklad NVDA - www.nvda-project.org) a bude plne ovládateľná pomocou klávesnice.

**Cieľ:** Vytvoriť programovacie prostredie plne prístupné pre nevidiacich umožňujúce programovať pohyb virtuálneho robota po ozvučenej štvorcovej sieti.

**Literatúra:** 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.
MILNE, L. R. AND LADNER, R. E. Blocks4All: overcoming accessibility barriers to blocks programming for children with visual impairments. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (p. 69). ACM.

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

**Dátum zadania:** 26.10.2022

**Dátum schválenia:** 02.11.2022 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

**Poďakovanie:** V prvom rade by som chcela poďakovať mojej školiteľke docentke RNDr. Ľudmile Jaškovej, PhD. za jej odborné rady, trpezlivosť, čas a podporu. Jej osobitý a zároveň profesionálny prístup mi bol veľkou pomocou počas tvorby tejto diplomovej práce, ale aj na predmetoch počas štúdia na našej fakulte.

# Abstrakt

Táto diplomová práca opisuje vývoj vzdelávacieho programovacieho prostredia s vlastným kompilátorom. V prostredí sa využívajú príkazy vstavaného programovacieho jazyka na presun virtuálnych objektov v požadovanom smere. Tieto príkazy možno použiť aj v rámci zložitejších štruktúr, ako sú cykly, vetvenia a podprogramy. Jazyk poskytuje aj operácie s celočíselnými premennými. V rámci práce je implementovaný editor kódu s kontrolou syntaxe a funkciou návrhu príkazov. Výsledná aplikácia bude tiež kompatibilná s čítačom obrazovky ako napríklad NVDA a bude plne ovládateľná pomocou klávesnice. Cieľom tejto diplomovej práce je navrhnúť a implementovať prehľadné a prístupné vzdelávacie programovacie prostredie pre začínajúcich programátorov.

**Kľúčové slová:**

# Abstract

This master's thesis describes the development of an educational programming environment with its own compiler. The environment utilizes commands of an embedded programming language to move virtual objects in the desired direction. These commands can also be used within more complex structures, such as loops, branches, and subprograms. The language also provides operations with integer variables. As part of the thesis, a code editor with syntax checking and command design functionality is implemented. The resulting application will also be compatible with screen readers such as NVDA and fully controllable via the keyboard. The aim of this master's thesis is to design and implement a clear and accessible educational programming environment for novice programmers.

**Keywords:**

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Úvod

# Motivácia

text

# Kapitola 1

# Východiská práce

Pred začatím vývoja je nevyhnutné, aby sme si definovali ciele a vlastnosti vzdelávacieho programovacieho prostredia.

V tejto kapitole preskúmame existujúce nástroje, programovacie prostredia, jazyky a posúdime ich náročnosť pre žiakov základných škôl so zrakovým postihnutím. Pomocou výsledkov tejto analýzy potom budeme vedieť zostrojiť vlastnú syntax, ktorá bude umožňovať žiakom písať aj komplikovanejšie štruktúry, ale zachová si jednoduchosť a intuitívnosť vhodnú do vzdelávacieho prostredia.

## 1.1 Edukačný softvér

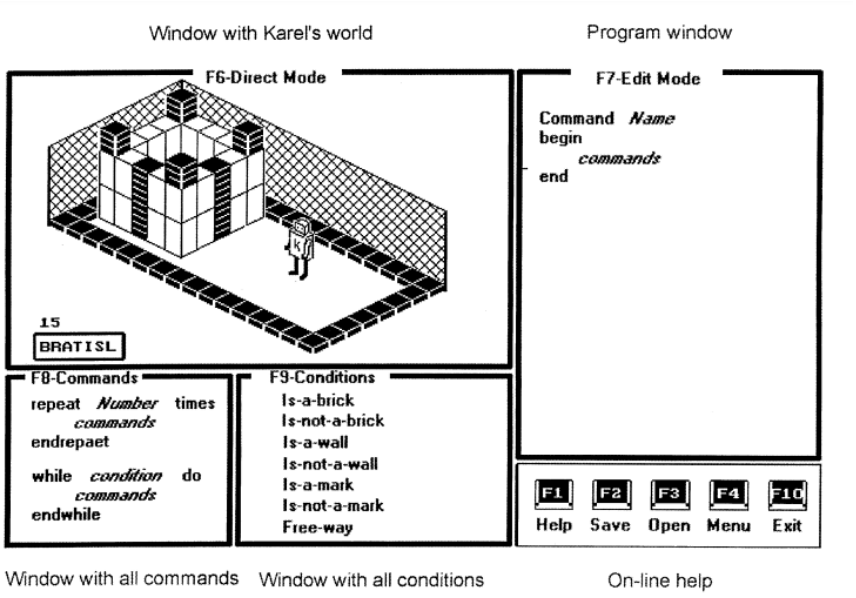
### 1.1.1 Znaky edukačného softvéru

Edukačný softvér je predovšetkým nástroj, ktorý má motivovať a podporiť začínajúcich programátorov - v našom prípade študentov základných škôl. To znamená prostredie s intuitívnym používateľským rozhraním pre prístup ku vzdelávaniu pútavou formou.

Programovacie prostredia spomenuté v tejto časti, sú určené pre vyučovanie na základných školách.

### 1.1.2 Robot Karel

Robot Karel [4] je programovací jazyk vytvorený Pattis-om ako úvod do jazyka Pascal pre študentov vysokých škôl. Môžeme ho zaradiť medzi jednoduchší jazyk, ktorý je dizajnovaný na učenie základných programátorských konceptov. Nepracuje s premennými, typmi ani výrazmi narozdiel od komplexnejších programovacích jazykov. Jeho hlavným aktérom je robot, ktorý vykonáva úlohy na mape s cestami, stenami a značkami (ktoré slúžia ako interaktívne objekty, ktoré môže presúvať a zbierať). Karel môže vykonávať jednoduché akcie, ako sú pohyb, otáčanie, zdvihnutie a položenie značiek.




Obr. 1.1: Karel3D

### 1.1.3 Karel3D

Karel-3D (Hvorecký, 1992) je rozšírením myšlienky robota Karel. Karel-3D (Obr 1.1) rozširuje pôvodného Karla do niekoľkých smerov, aby bol vhodnejší pre špeciálne cieľové skupiny, ako sú študenti predškolského veku alebo stredoškoláci. [4]

V tomto programovacom prostredí môžu žiaci príkazmi programovať pohyb robota v trojrozmernom priestore. Ako vidíme na obrázku, používateľské rozhranie je rozdelené do viacerých okien, kde každé okno ma svoju funkciu. Jedno okno je určené na písanie kódu, ďalšie okná poskytujú používateľovi vizualizáciu pohybu, predikciu textu a menu.

## 1.2 Zjednodušenie prístupu pre osoby so zrakovým postihnutím

Primárnou cieľovou skupinou našej aplikácie sú zrakovo postihnutí. V tejto podkapitole popíšeme, aké vlastnosti by malo mať prístupné programovacie prostredie vhodné pre všetkých žiakov - prihliadajúc na rôzne úrovne zrakového postihnutia.

### 1.2.1 Kategorizácia osôb so zrakovým postihnutím

Podľa Únie nevidiacich a slabozrakých Slovenska (USS) môžeme osoby so zrakovým postihnutím členiť do štyroch skupín [21] :

1. **nevidiaci** - osoby s úplnou stratou zrakového vnímania; maximálne dokážu určiť zdroj svetla

2. **prakticky nevidiaci** - osoby so zachovaným zvyškom zraku; vnímajú svetlo, obrysy a tvary predmetov

3. **slabozrakí** - osoby s vážnym poškodením zraku; majú problémy s vykonávaním zrakovej práce

4. **ľudia s poruchami binokulárneho videnia** - osoby s poruchou funkčnej rovnováhy a fyziologickej spolupráce pravého a ľavého oka; majú problémy s priestorovým vnímaním

### 1.2.2 Kritériá funkčnosti podľa zákona

Nariadenie vlády Slovenskej republiky zo 6. júla 2023 má ustanovené kritériá funkčnosti v prílohe č. 3 [14] Toto nariadenie má platnosť od 13.07.2023 a účinnosť od 28.06.2025. Aby softvér splnil kritérium funkčnosti pre zrakovo postihnutých používateľov musí spĺňať:

#### Používanie bez zraku

"*Ak služba poskytuje zrakové spôsoby používania, musí umožňovať najmenej jeden spôsob používania, ktorý si nevyžaduje zrak."*

#### Používanie v prípade obmedzeného zraku

"*Ak služba poskytuje zrakové spôsoby používania, musí umožňovať najmenej jeden spôsob používania, ktorý umožňuje používateľom s obmedzeným zrakom používať službu.*"

#### Používanie bez vnímania farieb

"*Ak služba poskytuje zrakové spôsoby používania, musí umožňovať najmenej jeden spôsob používania, pri ktorom sa od používateľa nevyžaduje vnímanie farieb.*"

#### Používanie bez hlasových schopností

"*Ak služba vyžaduje od používateľov hlasové pokyny, musí umožňovať najmenej jeden spôsob používania, ktorý si nevyžaduje hlasové pokyny. Hlasový pokyn zahŕňa akékoľvek zvuky vytvorené ústne, ako je reč, pískanie alebo tľoskanie jazykom.*"

#### Používanie v prípade obmedzenej motoriky alebo sily

"*Ak služba vyžaduje manuálne úkony, musí poskytovať najmenej jeden spôsob používania, ktorý umožňuje používateľom využívať ju alternatívnymi spôsobmi, na ktoré nie je nevyhnutná jemná motorika a manipulácia, sila rúk ani použitie viac ako jedného ovládacieho prvku súčasne.*"

**Používanie s obmedzeným dosahom**

"*Ak služba poskytuje manuálny spôsob používania, musí poskytovať najmenej jeden spôsob používania, ktorý funguje aj pri obmedzenom dosahu alebo obmedzenej sile.*"

**Používanie pri obmedzených poznávacích schopnostiach**

"*Ak služba vyžaduje využívanie poznávacích schopností, musí poskytovať najmenej jeden spôsob používania s prvkami, ktoré obsahujú vlastnosti zjednodušujúce a uľahčujúce jej používanie.*"

**Súkromie**

"*Ak služba zahŕňa prvky, ktoré sa poskytujú v záujme prístupnosti služby pre osoby so zdravotným postihnutím, musí poskytovať najmenej jeden spôsob používania, pri ktorom sa zachová súkromie pri používaní týchto prvkov.*"

### 1.2.3 Univerzálny návrh

Prístupná aplikácia by mala zahŕňať jednoduché ovládanie, intuitívne navigačné prvky a zreteľné reprezentovanie informácií. Dôležité je venovať pozornosť aj charakteru danej aplikácie; v našom prípade, že ide o edukačný softvér.

### 1.2.4 Vlastné nastavenia

Žiadaným rozšírením sú možnosti vlastného nastavenia, ktoré umožnia používateľom prispôsobiť si rozhranie podľa ich individuálnych potrieb. Čo môže zahŕňať zmenu veľkosti písma alebo a tmavého/svetlého režimu, čím sa zlepší prístupnosť pre ľudí s rôznymi úrovňami zrakového postihnutia.

### 1.2.5 Kontrastná a jasná vizuálna reprezentácia

Je nevyhnutné zabezpečiť, aby naše grafické prvky mali dostatočný kontrast. Takéto nastavenia pomáhajú používateľom so slabozrakosťou, ktorí môžu mať obmedzený zmysel pre kontrast a farby.

### 1.2.6 Klávesové skratky

Používatelia so zrakovým postihnutím nepoužívajú pri práci myš a sú zvyknutí na rýchlu navigáciu po aplikácii pomocou klávesových skratiek. Do aplikácie by sme mali pridať nápovedu so zoznamom klávesových skratiek a tiež ich uviesť v menu pri každej funkcii. Týmto spôsobom žiakom vieme uľahčiť pohyb v ponuke bez nutnosti zbytočného hľadania.

### 1.2.7 Čítač obrazovky

Okrem klávesnice je dôležitým asistenčným nástrojom aj čítač obrazovky [7]. Uľahčuje používateľovi vykonávať každodenné úlohy, či už ide o pohyb na internete alebo obsluhu akejkoľvek stolovej aplikácie. Aby dokázal čítač obrazovky odovzdať kompletnú informáciu o obsahu, je nevyhnutné dodržiavať normy a štandardy pre vizuálne prvky. V opačnom prípade vie byť používateľ ukrátený o množstvo informácií.

#### Microsoft Narrator

Microsoft Narrator [11] od spoločnosti Microsoft je čítač obrazovky, ktorý je súčasťou opreračného systému Windows.

Poskytuje služby ako:

- hlasový výstup - čítanie obsahu obrazovky,
- navigácia v používateľskom rozhraní,
- kompatibilita s Braillovým displejom,
- hlasové nastavenia (rýchlosť čítania, hlasový tón),
- prístupný aj počas inštalácie systému Windows.

#### NVDA

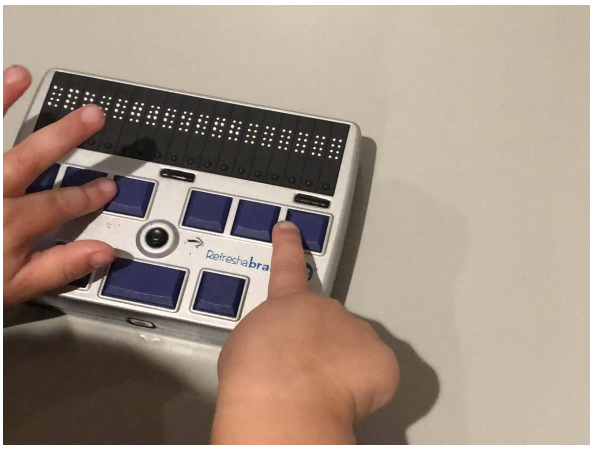
NVDA (NonVisual Desktop Access) [1] je voľne dostupný čítač obrazovky s otvoreným zdrojovým kódom. Softvér vyvinuli dvaja nevidiaci programátori Michael Curran a James Teh. Dnes je dostupný v 55 jazykoch vo vyše ako 175 krajinách.

Oproti MS Narratorovi má NVDA:

- otvorený zdrojový kód - neustály vývoj komunitou,
- lepšie možnosti konfigurácie (napr. nastavenia Braillových displejov),
- rozsiahle možnosti jazykov,
- nastavenia viacerých zvukových kariet - kontrola zvuku,
- podpora pre špecifické technológie a aplikácie.

### 1.2.8 Braillov displej

Braillov displej, alebo Refreshable Braille Display (RBD), je zariadenie zobrazujúce Braillove znaky dvíhaním a klesaním bodiek (na obrázku 1.2). Umožňuje čítanie aj písanie textu v spolupráci s počítačom či tabletom. [3]: "*Pre mnohých študentov, najmä pre začínajúcich čitateľov, ktorí majú problémy s hmatovým rozlišovaním alebo študentov s motorickými problémami, môže byť tradičné papierové braillovo písmo náročné na čítanie. Bodky na*



Obr. 1.2: Braillov displej

*RBD sú konzistentné na dotyk a jednoduchšie na čítanie."* Rovnako oproti starým Braillovým písacím strojom Perkins (pripomínajúcim klasické písacie stroje), RBD majú jemné tlačidlá, vďaka čomu sú pre žiakov ideálne na písanie.

Takéto zariadenie nie je cenovo dostupné pre všetkých. Preto sa veľká časť používateľov (ak nie väčšina) spolieha len na čítače obrazovky.

## 1.3 Programovacie prostredia pre zrakovo postihnutých

V tejto podkapitole preskúmame existujúce programovacie prostredia, ktoré sú navrhnuté pre používateľov so zrakovým postihnutím. Analyzujeme, ktoré programovacie jazyky podporujú, aké nástroje na úpravu kódu poskytujú a aké sú ich výhody a obmedzenia.

### 1.3.1 Quorum

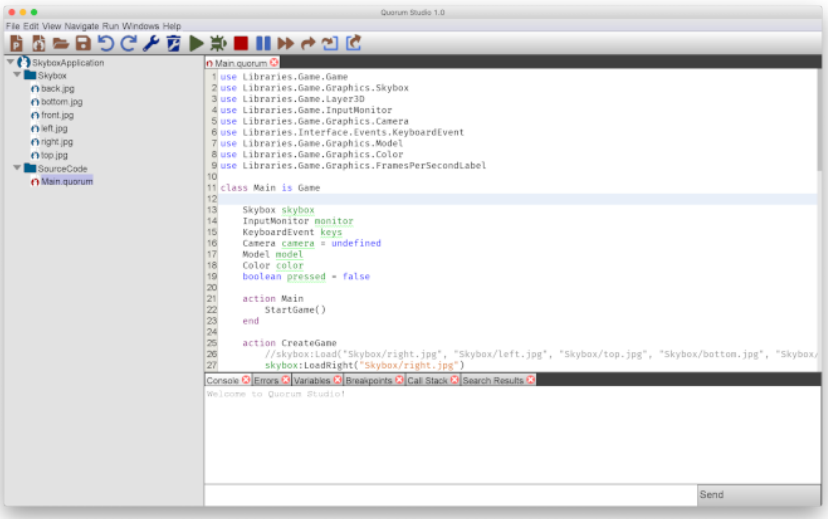
Quorum [16] je programovací jazyk navrhnutý pre začiatočníkov a študentov so špeciálnymi vzdelávacími potrebami, ktorý má vlastné prostredie (obr. 1.3). Tento jazyk je schopný podporovať rôzne formy audio vstupov a výstupov.

**Prínosy pre žiakov ZŠ:**

- Je k nemu voľne dostupný kurz, vhodný aj pre používateľa, ktorý sa s programovaním ešte nestretol.
- Audio vstupy aj výstupy.

**Možné nedostatky:**

- Neprináša samostatne okamžitú motiváciu prostredníctvom pútavej tematiky.
- Má obmedzenú podporu dostupných knižníc a nástrojov v porovnaní s inými jazykmi.




Obr. 1.3: Quorum

- Mimo školského prostredia je využívaný len veľmi zriedkavo.

### 1.3.2 CodeTalk a Visual Studio Code

CodeTalk od spoločnosti Microsoft [15] je významné rozšírenie pre Visual Studio Code (VSC), ktoré umožňuje programátorom so zrakovým postihnutím plnohodnotne pracovať s týmto populárnym vývojovým prostredím.

**Prínosy pre žiakov ZŠ:**

- Kontakt s populárnym vývojovým prostredím.
- Po otvorení súboru sa používateľovi vytvorí zhrnutie kódu
- Možnosť hlasového ovládania a prispôsobenia prostredia.
- Debuggovanie za pomoci zvukov

**Možné nedostatky:**

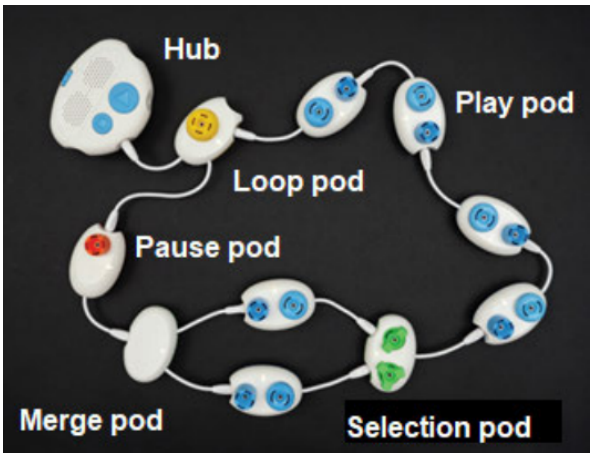
- Nespája sa s pútavou tematikou.
- Môže odradiť začiatočníkov, ktorí sú zvyknutí na svoje nástroje.

### 1.3.3 Code Jumper

Code Jumper predstavuje inovatívny prístup k výučbe. Ide o set fyzických zariadení, vyvinutých spoločnosťou Microsoft. Môžeme ho vidieť na obrázku 1.4.

Žiaci môžu vytvárať algoritmy rukami - zapájaním jednotlivých zariadení do seba, a tak programovať pomocou blokov prístupnešie než akoukoľvek inou technológiou. Vzniká

hmatateľný vzdelávací zážitok, ktorý nie len poskytuje znalosti, ale aj podporuje spoluprácu vo vzdelávacom prostredí.




Obr. 1.4: Code Jumper

Výsledky testovania takéhoto zariadenia môžeme nájsť v [9]: "*Zistili sme, že fyzický programovací jazyk Code Jumper je vhodnejší, ak študenti pracujú v pároch. Môžu lepšie spolupracovať. Avšak väčšia skupina nie je vhodná, pretože každý študent chce hmatom cítiť postupnosť fyzických blokov. Ideálne by každý študent mal hmatom cítiť, ktorý príkaz sa práve vykonáva.*"V článku je spomenutý aj postreh, že vzhľadom na obmedzený počet príkazových modulov boli žiaci motivovaní k používaniu cyklov. Špeciálne bol Code Jumper obľúbený medzi žiakmi, ktorým bola hudobná tematika blízka.

**Prínosy pre žiakov ZŠ:**

- nová forma výučby - fyzické bloky, práca vo dvojiciach,
- podpora kreativity - dáva žiakom priestor na experimentovanie s rôznymi kombináciami blokov,
- tematika a pútavosť,
- naučiť sa používať Code Jumper je pomerne rýchle a jednoduché.

**Možné nedostatky:**

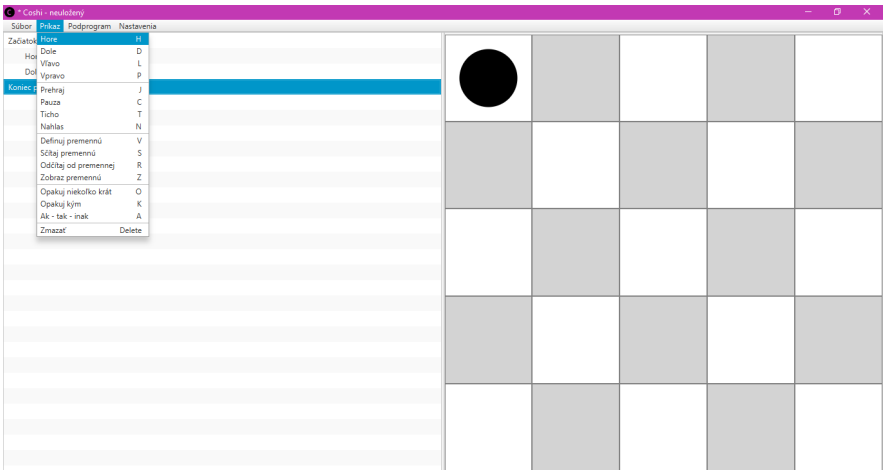
- finančné obmedzenia,
- obmedzené pokročilé možnosti.

### 1.3.4 Coshi

Vzdelávacie programovacie prostredie Coshi, bolo výsledkom bakalárskej práce Michala Kováča [10]. Vďaka pozitívnej skúsenosti žiakov aj učiteľov, bol inšpiráciou na rozsiahlejší projekt. Poslúžil tak, ako motivácia zadania tejto diplomovej práce.

Prostredie Coshi (obr. 1.5) je taktiež zamerané na riadenie pohybu virtuálneho objektu, konkrétne robota, po štvorcovej sieti. Písanie zdrojového kódu prebieha prostredníctvom blokového programovania. Táto metóda programovania využíva vizuálny prístup, kde používateľ nevpisuje kód textovo, ale skladá príkazy v podobe blokov, ktoré môžu potom tvoriť komplexnejšie štruktúry ako sú cykly, vetvenia alebo podprogramy.

Coshi bol vyvíjaný na základe výskumu prístupných edukačných softvérov za pomoci testovania aplikácie priamo na vyučovaní. Výsledky tohto testovania môžeme nájsť v článku [9]: "*Učitelia rôznych predmetov nás informovali, že zrakovo postihnutí študenti s nadšením hovorili o programe Coshi aj mimo vyučovacích hodín informatiky. To naznačuje pozitívne prijatie programu. Pokiaľ ide o užívateľské rozhranie, študenti aktívne používali klávesové skratky implementované v Coshi a úprava programu im nespôsobila žiadne vážne problémy. V prostredí Coshi je podmienené vetvenie implementované ako plné vetvenie, preto muselo mať oboje vetvy - IF a ELSE, aj keď vetva ELSE nie je potrebná. V tom prípade je jedna vetva prázdna. Táto implementácia sa ukázala ako problémová, pretože študenti sa nemohli ľahko orientovať v štruktúre podmienenej príkazovej štruktúry a neboli si istí, kam majú umiestniť svoje príkazy.*"




Obr. 1.5: Coshi

**Prínosy pre žiakov ZŠ:**

- Programovanie pomocou blokov v slovenčine.
- Tematika a zvuková odozva.
- Interaktívne úlohy.
- Možnosti prispôsobenia.
- Možnosť pridávať do aplikácie rozšírenia - vlastné mapy a zvukové balíčky.

**Možné nedostatky:**

- Keďže sa programuje v blokoch poskytuje menšiu variabilitu pri písaní kódu.
- Jazyk obsahuje zbytočné syntaxové podmienky.

## 1.4 Porovnanie dostupných nástrojov

V tejto časti preskúmame programovacie nástroje pre vidiacich a zrakovo postihnutých používateľov v perspektíve žiaka základnej školy. Zameriame sa na ich potenciál v oblasti vzdelávania, jednoduchosť ovládania a prispôsobiteľnosť špecifickým potrebám používateľa.

### 1.4.1 Programovacie jazyky

Niektoré svetovo rozšírené jazyky nie sú vhodnou voľbou pre nevidiacich programátorov. Ako je uvedené v [6] : "*Ďalším dôležitým hľadiskom je voľba programovacieho jazyka; mnohé bežne používané jazyky, ako napríklad C a Java, v širokom rozsahu využívajú nealfanumerické znaky, ako sú zátvorky a kučeravé zátvorky, čo môže byť náročné na prácu s čítačom obrazovky. Okrem toho komplexná syntax mnohých jazykov môže zvýšiť pravdepodobnosť chýb pri písaní a komplikovať ladenie kódu.*"Zatiaľ čo pri jazykoch ako Quorum môžeme vidieť, že minimalizujú používanie nealfanumerických znakov.

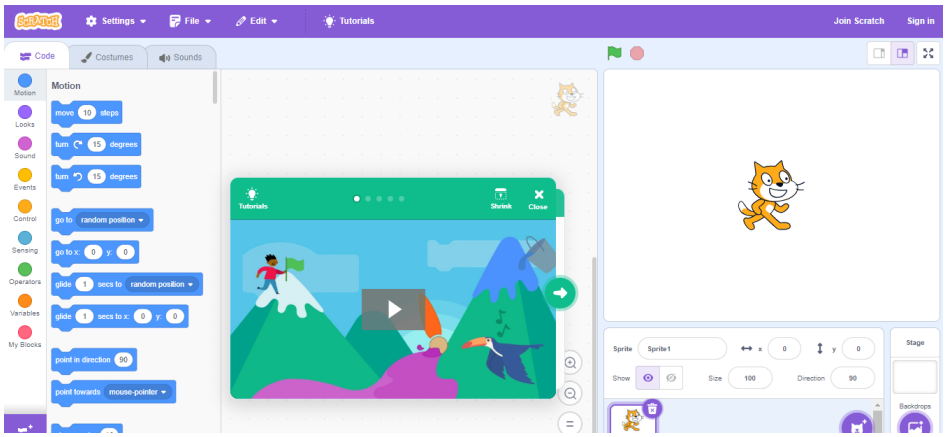
### 1.4.2 Programovacie prostredia

Vďaka vizuálnym prvkom programovania pomocou blokov je táto metóda pre začiatočníkov veľmi populárna vo vzdelávacom prostredí (ako napríklad jazyk Scratch [17] na obr. 1.6).

Sprístupnenie tejto formy je veľkou výzvou ako vidíme z výskumu [12]. V článku v spomenutých prostrediach veľa elementov nie je dobre prispôsobených na prácu s čítačom obrazovky. Navigácia bola náročná a miestami nereagovalo správne ani ovládanie pomocou hlasu. Napriek tomu žiaci zhodnotili zážitok pozitívne. Z tohto dôvodu sa javí prístup vytvárania hmatateľných vzdelávacích zážitkov ako vhodná voľba pre začiatočníkov (Code Jumper).

### 1.4.3 Zhrnutie

Vývoj prístupných technológií prináša nové nástroje na syntetizáciu reči, ovládanie hlasom, ovládanie zrakom, čím sa eliminujú bariéry spojené so zdravotnými obmedzeniami. Môže byť dobrou stratégiou prioritizovať edukačné systémy, keďže ide o programátorov začiatočníkov, ktorí okrem zrakového postihnutia majú aj prekážky v podobe nedostatku skúseností.




Obr. 1.6: Scratch

## 1.5 Použité technológie

V tejto časti predstavíme technológie, ktoré budeme používať pri vývoji našej aplikácie. Zameriame sa na jazyk C# a framework .NET, ktoré poskytujú vhodné nástroje na tvorbu prístupných programovacích prostredí.

### 1.5.1 Jazyk C#

C# (C-Sharp) je objektovo orientovaný jazyk vyvinutý spoločnosťou Microsoft, určený najmä pre vývoj aplikácií pre Windows. Je obľúbeným jazykom pre tvorbu stolových (desktop) aplikácií, webových stránok aj mobilných aplikácií. Jeho syntax sa podobá jazykom C++ a Java. C# je vhodnou voľbou aj vďaka svojej kompatibilite s ostatnými technológiami od Microsoftu.

### 1.5.2 Platforma .NET

.NET je voľne dostupná platforma, ktorá má otvorený zdrojový kód. Je vhodný na vývoj softvéru pre Windows, Linux, aj macOS, ale dá sa využiť aj na tvorbu aplikácií pre mobilné zariadenia - Android a iOS.

V .NET je implementovaná funkcia "garbage collection"na automatické riadenie pamäte, ktorá uvoľňuje pamäť, keď už nie je potrebná. Vďaka tomu sa zlepšuje výkon aplikácií a programátor sa môže sústrediť na funkčnosť bez starosti o správu pamäte.

## 1.6 Kompilátory a interpretre

V tejto podkapitole sa budeme hlbšie venovať kompilátorom a interpreterom. Popíšeme, ako sa navrhujú a vyvíjajú. Nakoniec porovnáme ich rozdiely a tiež objasníme, pre ktorú

verziu sme sa rozhodli a prečo. Okrem preštudovaných článkov bolo veľkou pomocou aj absolvovať predmet "Kompilátory a interpretre", ktorý vedie pán doc. RNDr. Ľubomír Salanci, PhD.

Najprv si definujme kľúčové pojmy podľa [5] :

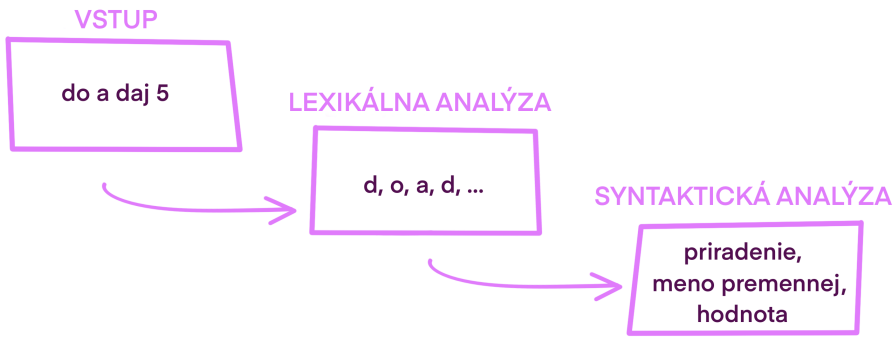
- **Procesor -** je počítačový program, ktorý spracováva iné počítačové programy,
- **Kompilátor -** je procesor, ktorý prekladá program napísaný v programovacom jazyku (nazývaný "zdrojový kód") do kódu, ktorý môže počítač vykonať (nazývaný objektový kód),
- **Interpreter -** je procesor, ktorý prijíma program napísaný v zdrojovom kóde, prekladá ho do niektorej priamo vykonateľnej formy, a hneď ju riadene vykonáva. Vykonateľná forma môže byť alebo nemusí byť objektový kód,
- **Translátor -** je procesor, ktorý prekladá jeden zdrojový kód do iného.

Pre účely tejto podkapitoly, použime zjednodušenú verziu nášho programovacieho jazyka.

Príkazy pre pohyb: *hore, dole, vpravo, vľavo*.
Definícia premennej: *do a daj 5*.
Operácie s premennými: *ku a pričítaj 2*
Cyklus s daným počtom opakovaní: *Opakuj 3 krát* ...telo cyklu... *koniec*

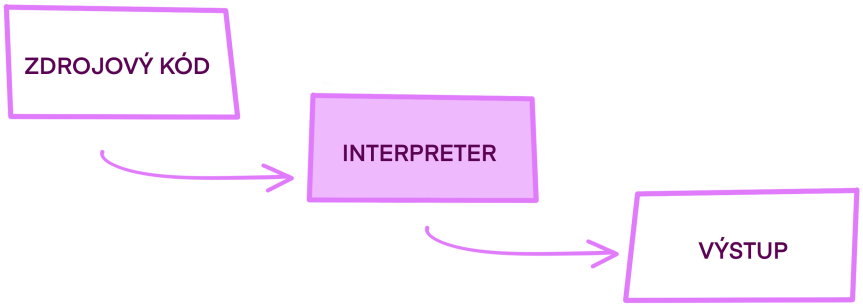
### 1.6.1 Interpreter

Na interpreter (obr. 1.8) sa môžeme pozrieť tak, že sa vykonáva v troch krokoch: lexikálna analýza, syntaktický analýza a vykonávanie (interpretácia) [13]. Ako sa spracuje vstup od používateľa, vidíme na obrázku 1.7, kde vstup je používateľom zadaný program v ľubovoľnom jazyku (ktorý interpreter pozná).




Obr. 1.7: Analýza vstupu cez vrstvy

- **Lexikálna analýza (skener):**
  - Najprv prebehne lexikálna analýza, čo znamená čítanie po znakoch zo vstupu.
  - Z týchto znakov sa skladajú základné stavebné prvky jazyka, nazývané lexémy (slová, čísla, symboly).
  - Medzitým sa preskakujú medzery, nové riadky a iné znaky, ktoré nie sú dôležité pre beh programu.
- **Syntaktická analýza (parser):**
  - Vstup potom prechádza syntaktickou analýzou.
  - Nájdené lexémy sa spracujú ako programové konštrukcie.
  - Rozhoduje sa, či ide o jednoslovný príkaz ako *hore*, alebo o priradenie do premennej, cyklus, definíciu podprogramu a podobne.
- **Vykonávanie príkazov:**
  - Posledným krokom je samotné vykonávanie príkazov, keďže ide o interpreter.
  - Ak sa dostane k rozpoznanej lexéme ako príkazu *hore*, vykonáva ho okamžite.
  - Vynímky sú prípady, keď sa nachádza vo väčšej konštrukcii, ako je vetvenie, podprogram a iné.



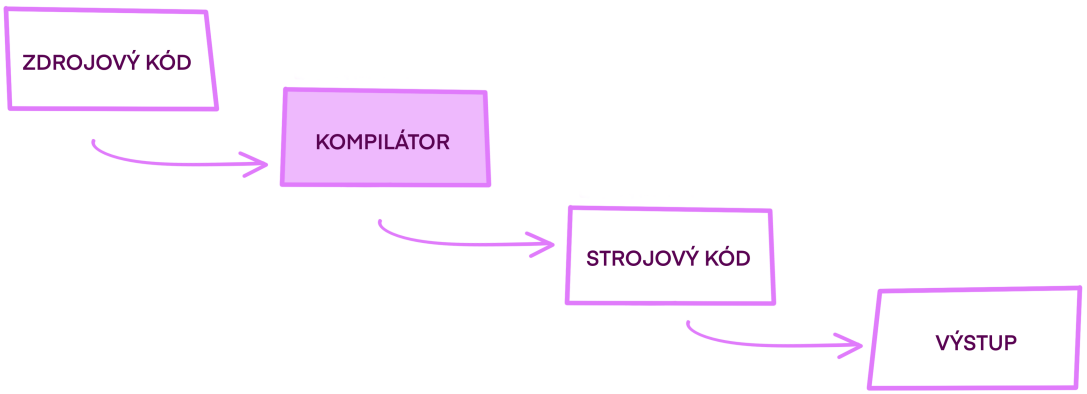

Obr. 1.8: Interpreter

### 1.6.2 Kompilátor

Kompilátor narozdiel od interpreteru príkazy hneď nevykonáva, ale prekladá ich do strojového kódu (obr. 1.9). Vo všeobecnosti môžeme fázy kompilátora popísať nasledovne [13]:

- **Lexikálna analýza** - číta text a delí ho na tokeny, z ktorých každý zodpovedá napríklad názvu premennej, kľúčovému slovu alebo číslu,

- **Syntaktická analýza** - vezme zoznam tokenov vytvorených lexikálnou analýzou, usporiada ich do stromovej štruktúry (nazývanej syntaktický strom), ktorá odzrkadľuje štruktúru programu,
- **Kontrola typov** - analyzuje syntaktický strom a určuje, či program neporušuje syntax alebo logiku jazyka (napríklad či sa premenná používa, ale nie je deklarovaná)
- **Generovanie medzikódu** - program je preložený do jednoduchého strojovo nezávislého jazyka,
- **Prideľovanie registrov** - symbolické názvy premenných používané v medzikóde sú preložené na čísla, kde každé číslo zodpovedá registru v cieľovom strojovom kóde,
- **Generovanie strojového kódu** - medzikód je preložený do assemblerového jazyka pre architektúru daného stroju,
- **Assemblovanie, spájanie** - kód v assemblerovom jazyku je preložený do binárnej reprezentácie.




Obr. 1.9: Kompilátor

### 1.6.3 Virtuálna mašina

Pretože assembler je závislý na architektúre stroja, môžeme naraziť na problém s kompatibilitou medzi strojmi s rozličnými platformami. Na riešenie tohto problému vznikla idea virtuálnej mašiny. Proces kompilácie sa v tomto prípade nedostane až ku prepisu do strojového kódu, ale medzikód spracuje virtuálna mašina.

Tá bude mať pamäť, procesor, algoritmus simulácie a bude vedieť rozpoznávať príkazy na základe ich kódu. Procesor takéhoto virtuálneho počítača bude vykonávať inštrukcie v pamäti, kde jedna inštrukcia bude postupnosťou čísel (bytecode). Ďalšou výhodou pre vývojára je, že virutálna mašina umožňuje krokovať alebo zastavovať vykonávaný program.

### 1.6.4 Rozdiely medzi kompilátorom a interpreterom

V tabuľke 1.1 môžeme vidieť niekoľko zásadných rozdielov medzi kompilátorom a interpreterom spomínané v [18].

Dôvod, prečo sme sa rozhodli pre náš jazyk programovať kompilátor (pre bytecode) vznikol po konzultáciách na predmete "Kompilátory a interpretre". Bolo odporúčané, že pokiaľ chceme implementovať aj vlastné podprogramy a iné zložitejšie konštrukcie, bude riešenie s kompilátorom a virtuálnou mašinou 1.6.3 elegantnejšie aj zaujímavejšie pre vývoj.

Táto cesta sa aj neskôr ukázala ako lepšia kvôli zvoleným technológiám .NET a C#. Podrobnosti sú popísané v kapitole 3.

| **Rozdiely** | **Kompilátor** | **Interpreter** |
|---|---|---|
| Výhody | Program je už preložený do strojového kódu. | Jednoduchší na učenie, vhodný pre začiatočníkov. |
| Nevýhody | Program nie je možné zmeniť bez návratu k zdrojovému kódu. | Kód na interpretáciu potrebuje mať na zariadení príslušný interpreter. |
| Strojový kód | Ukladá strojový kód na disk. | Vôbec neukladá strojový kód. |
| Rýchlosť | Už preložený kód stačí spustiť. | Pri každom spustení je nutné prekladať kód. |
| Generovanie programu | Generuje spustiteľný súbor, ktorý môže byť vykonaný nezávisle od zdrojového kódu. | Nevytvára výstupný program, prekladá a vykonáva priamo v čase behu programu. |
| Optimalizácia | Efektívnejšia, pretože vidí celý kód vopred. | Nie tak efektívna, vidí len riadok po riadku. |
| Spracovávanie chýb | Zobrazuje všetky chyby a upozornenia počas kompilácie. Chybný kód sa nedá spustiť. | Zobrazí chybu, keď sa ku nej dostane. |
| Programovacie jazyky | C, C++, C#, Java. | Python, JavaScript, Ruby, PHP. |

Tabuľka 1.1: Rozdiely medzi kompilátorom a interpreterom

## 1.7 Metódy vývoju softvéru

Vývoj našej aplikácie prebieha vo viacerých krokoch:

- Výskum v oblasti danej problematiky - ako sa vyvíja edukačný softvér, ako sa vyvíja softvér pre osoby so zrakovým postihnutím a aké sú nároky na takúto aplikáciu,

- Návrh aplikácie na základe výsledkov prvej výskumnej časti našej práce,
- Pokračovanie výskumu formou opakovaného testovania a hľadania najlepších riešení spoločne so skupinami žiakov aj učiteľov.

Takáto metóda vývoju je kombináciou vodopádového modelu s agilným vývojom. Vodopádový model je lineárnym prístupom k vývoju softvéru, keď jedna fáza (analýza, návrh, implementácia, testovanie) nasleduje priamo za druhou. Každú fázu najprv dokončíme než začneme ďalšiu.

Agilný vývoj je sústredený na spoluprácu a schopnosť prispôsobiť sa zmenám. Vývoj prebieha v krátkych iteráciách, kde sa opakovane dopĺňajú malé časti funkčného softvéru. Vďaka tomu vieme rýchlo reagovať na nové požiadavky.

Náš edukačný softvér teda vyvíjame postupne ako vodopádový model. Následne ho testujeme v triede a opakovane zapracovávame zlepšenia na základe spätnej väzby ako pri agilnom vývoji.

# Kapitola 2

# Návrh

# Kapitola 3

# Implementácia

# Kapitola 4

# Výskum

Typ nášho výskumu sa označuje ako užívateľský výskum (UX Research) [8]. Ide o dôležitý základ procesu vývoja akéhokoľvek softvéru. Takýto výskum sa zameriava na sledovanie a pochopenie preferencií, potrieb a správania požívateľov.

V tejto kapitole podrobne popíšeme priebeh nášho UX výskumu a jeho výsledky v snahe vytvoriť užívateľsky príjemný a predovšetkým prístupný softvér pre nevidiacich používateľov. Preskúmame naše metodické prístupy, analyzujeme výsledky a vyvodíme závery, ktoré nám budú slúžiť ako základ pre ďalší vývoj a zdokonalenie našej aplikácie.

Prostredníctvom opakovaného testovania a zapracovania pripomienok sa nám podarí identifikovať kľúčové oblasti, ktoré robia našu aplikáciu prístupným a vhodným edukačným softvérom.

# Záver

# Literatúra


[1] NV Access. Nvda. https://www.nvaccess.org/about-nv-access/, Accessed: 6.11.2023.

[2] Soma Bhaduri. Making learning inclusive: A quick reference guide on accessibility standards. https://elearningindustry.com/making-learning-inclusive-a-quick-reference-guide-on-accessibility-standards Accessed: 20.10.2023.

[3] Diane Brauner. Benefits of using a braille display with emerging readers. https://www.perkins.org/resource/benefits-using-braille-display-emerging-readers/, Accessed: 14.11.2023.

[4] Peter Brusilovsky, Eduardo Calabrese, Jozef Hvorecký, Anatoli Kushnirenko, and Philip Miller. Mini-languages: A way to learn programming principles. *Education and Information Technologies*, 2:65–83, 03 1997.

[5] R. L. Glass. An elementary discussion of compiler/interpreter writing. *ACM Comput. Surv.*, 1(1):55–77, mar 1969.

[6] 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.

[7] Michal Homola. Kompenzačné technológie a postupy nevidiacich prekladatel'ov a tlmočníkov. *A TLMOČENIA*, 2017.

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

[9] Ľ. Jašková. Three programming environments friendly to blind students in lower secondary education. In *EDULEARN22 Proceedings*, 14th International Conference on Education and New Learning Technologies, pages 8099–8108. IATED, 4-6 July, 2022 2022.

[10] Michal Kováč. Programovacie prostredie na ovládanie pohybu virtuálneho objektu prístupné pre nevidiacich žiakov sekundárneho vzdelávania. bakalárska práca, FMFI UK, 2019.

[11] Microsoft. Complete guide to narrator. `https://support.microsoft.com/en-us/windows/complete-guide-to-narrator-e4397a0d-ef4f-b386-d8ae-c172f109bdb1`, Accessed: 3.12.2023.

[12] Lauren R. Milne and Richard E. Ladner. Blocks4all: Overcoming accessibility barriers to blocks programming for children with visual impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, page 1–10, New York, NY, USA, 2018. Association for Computing Machinery.

[13] Torben Ægidius Mogensen. *Basics of compiler design*. 2010.

[14] NRSR. Zákon o prístupnosti výrobkov a služieb pre osoby so zdravotným postihnutím a o zmene a doplnení niektorých zákonov, 2022. § 6.

[15] Suresh Parthasarathy, Venkatesh Potluri, Gopal Srinivasa, Swami Manohar, and Priyan Vaithilingam. Codetalk:. In *ACM CHI 2018*, April 2018.

[16] Quorum. Getting started with quorum. `https://quorumlanguage.com/tutorials/language/gettingStarted.html`, Accessed: 2.11.2023.

[17] Scratch. About scratch. `https://scratch.mit.edu/about`, Accessed: 2.11.2023.

[18] John Smith. Compiler vs interpreter – difference between them. `https://www.guru99.com/difference-compiler-vs-interpreter.html`, Accessed: 30.11.2023.

[19] Andreas M. Stefik, Christopher Hundhausen, and Derrick Smith. On the design of an educational infrastructure for the blind and visually impaired in computer science. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 571–576, New York, NY, USA, 2011. Association for Computing Machinery.

[20] Codecademy Team. What is .net? `https://www.codecademy.com/article/what-is-net`, Accessed: 10.11.2023.

[21] Únia nevidiacich a slabozrakých Slovenska. Zrakové postihnutie. `https://unss.sk/zrakove-postihnutie/`, Accessed: 28.11.2023.

# Príloha A: obsah elektronickej prílohy

# Príloha B: Používateľská príručka