Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Robot Karel ovládaný hlasem

2022

Bc. Jan Nejezchleba

Vedoucí práce:
doc. RNDr. Miroslav Kolařík, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

Bibliografické údaje

Autor:	Bc. Jan Nejezchleba
Název práce:	Robot Karel ovládaný hlasem
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2022
Studijní program:	Aplikovaná informatika, Specializace: Vývoj software
Vedoucí práce:	doc. RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	42
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliographic info

Author:	Bc. Jan Nejezchleba
Title:	Voice controlled robot Karel
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2022
Study program:	Applied Computer Science, Specialization: Software Development
Supervisor:	doc. RNDr. Miroslav Kolařík, Ph.D.
Page count:	42
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Byla vytvořena aplikace pro podporu výuky základů programování hravou formou skrze vývojové prostředí Robot Karel s možností hlasového ovládání vybraných částí aplikace. Obsah práce zahrnuje představení programovacího jazyka Karel, popis technického provedení vytvořené aplikace a její uživatelskou dokumentaci.

Synopsis

An application has been created for assisting in teaching basics of programming in playful way through development environment Robot Karel with possibility of voice controlling selected parts of the application. Contents of this thesis include the introduction of programming language Karel, description of technical specifications of the created application and its user documentation.

Klíčová slova: výuka programování; programovací jazyk; Robot Karel; hlasové ovládání; Kotlin; TornadoFX

Keywords: teaching programming; programming language; Robot Karel; voice control; Kotlin; TornadoFX

Chci poděkovat vedoucímu mé diplomové práce, doc. RNDr. Miroslavu Kolaříkovi Ph.D. za odborné vedení mé diplomové práce. Dále mé poděkování patří firmě Phonexia za poskytnutí licence jejich softwaru pro účely této práce a v neposlední řadě pak mé díky patří i Ing. Janu Pavlíkovi z firmy Phonexia za ochotný a vřelý přístup při úkonech spojených s propůjčeným softwarem.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

1 Úvod

Ve světle nové legislativy uzákoňující programování jako nedílnou součást osnov pro žáky základních škol, bude bezesporu nutné najít způsob, jak žáky během nové koncepce výuky zaujmout a motivovat. Jedním z důvodů mého zájmu o toto téma je skutečnost, že jazyk Karel, který původně vznikl právě za účelem realizování uživatelsky přívětivé a zábavné formy výuky základů programování, se jeví jako ideální učební pomůcka zapadající do nově vznikajících osnov.

Cílem aplikace vzniklé během této práce je převléknout vývojové prostředí robota Karla do moderního hávu, který by uživatele měl zaujmout a zároveň by měl dodat přidanou hodnotu skrze možnosti hlasového ovládání. Mimo jiné je kladen důraz i na myšlenku tvorby a sdílení uživatelských cvičení, která mohou být tvořena například jako součást výuky, což by k užívaní aplikace mohlo motivovat i samotné učitele.

V první části práce je popsán jazyk Karel, jeho historie a příklady existujících implementací. Následuje výčet použitých technologií a představení architektury aplikace včetně jednotlivých technických řešení. Na závěr je uvedena uživatelská příručka obsahující všechny potřebné informace pro snadné užívání aplikace.

2 Jazyk Karel

2.1 Historie

Vznik tohoto jazyka se datuje na přelom 70. a 80. let 20. století, kdy profesor stanfordské univerzity, Richard E. Pattis, poprvé přišel s myšlenkou výuky programování v tomto jazyce. Název jazyka je poctou českému spisovateli Karlu Čapkovi, díky kterému se slovo *robot* zapsalo do světových slovníků. Autor k jazyku vytvořil i učební materiál v podobě knihy *Karel The Robot: A Gentle Introduction to the Art of Programming*. [1]

Základní myšlenkou jazyka popsaného ve zmíněné knize bylo ovládat robota Karla, který se pohyboval na políčkách v imaginárním robotím městě. Město se skládalo z vodorovných *streets* a svislých *avenues*, které jsou v českém překladu označovány jako ulice a bulváry. [2]

Na tomto herním poli mohl Karel použít v originální verzi jazyka 5 příkazů:

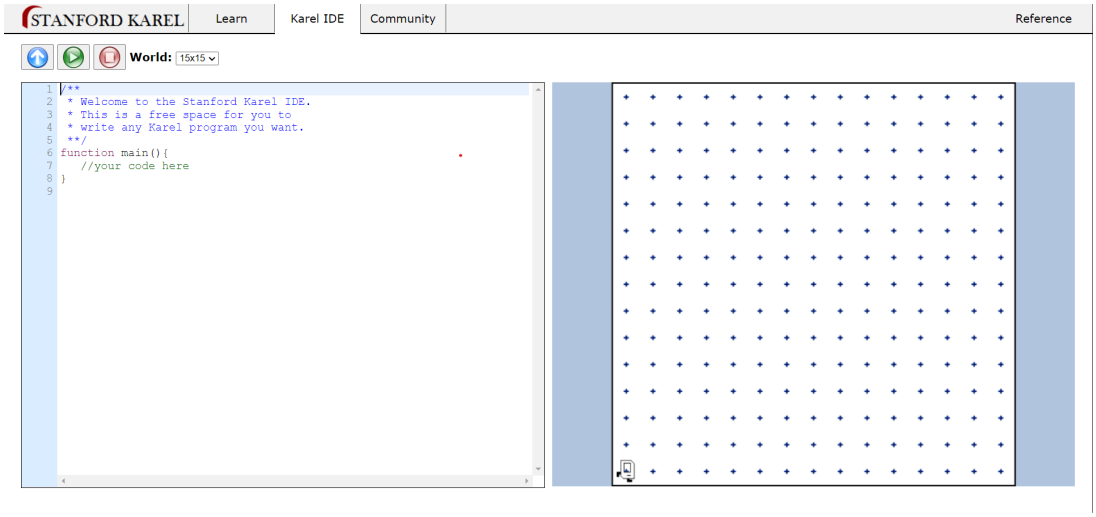
- move
- turnleft
- putbeeper
- pickbeeper
- turnoff

Dále mohl využít znalostí o svém okolí. Například jakým směrem je momentálně otočen, jestli se na aktuálním políčku nachází tzv. *pípák* nebo jestli se před ním nenachází zeď. Tyto poznatky pak v kombinaci s podmínkovými výrazy, cykly a základními příkazy umožňovaly definovat nové příkazy pro využití ve vytvářeném programu. [2]

Jazyk se k výuce na Stanfordské univerzitě využíval až do půlky 90. let 20. století, kdy byl nahrazen jazykem Java. To ale nezabránilo jeho následnému návratu do výukových materiálů, když se v roce 2005 začal vyučovat ve verzi implementované v jazyce Java. [3]

2.2 Příklady implementací

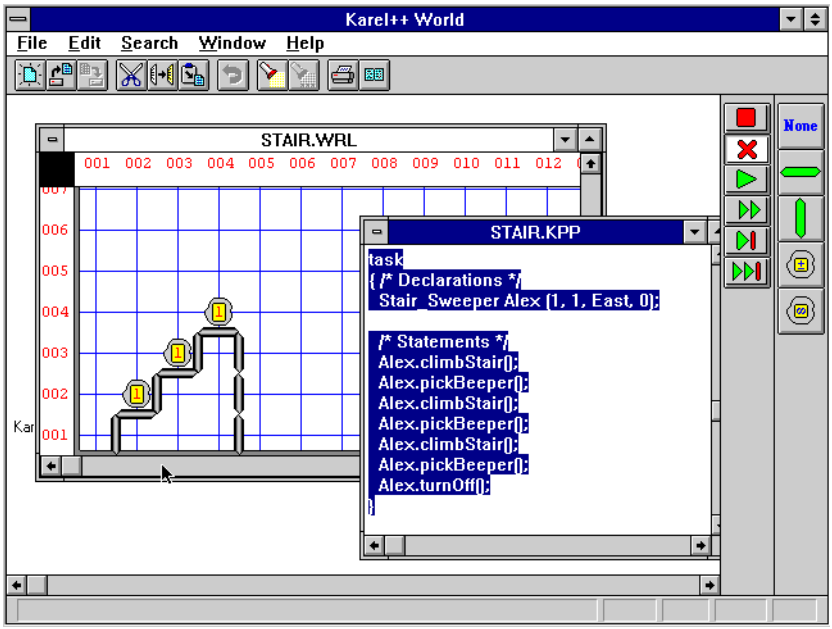
Jako většina programovacích jazyků se i Karel během let vyvíjel se vznikem nových modifikovaných prostředí. Řada z nich vznikala právě v tuzemsku, kde se tento jazyk těší značné oblibě. Programátoři pro své účely lehce poupravili podobu jazyka a vyměnili například tzv. *pípáky* za obecnější *značky* a přeložili příkazy do češtiny.



Obrázek 1: Vývojové prostředí Karel na stránkách Stanfordské univerzity (2022)

2.2.1 Karel++

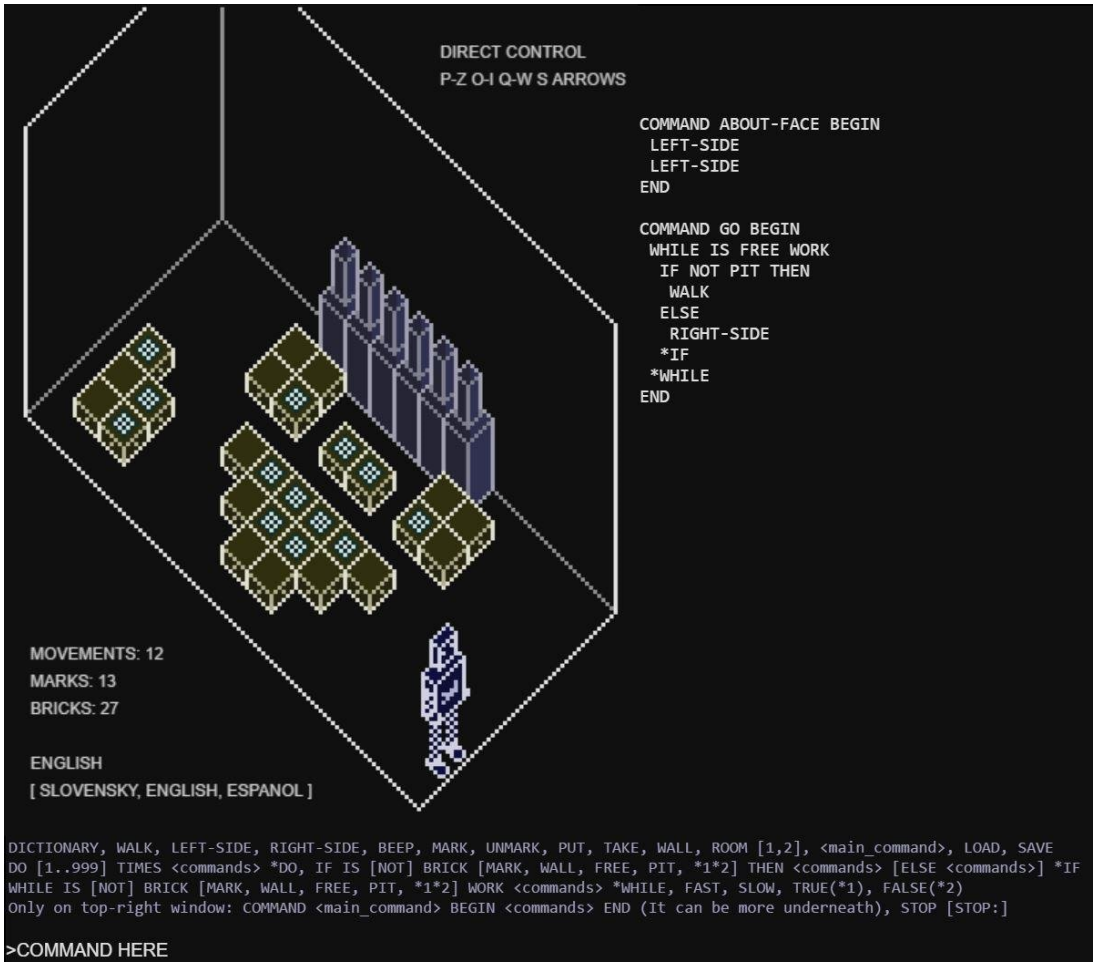
Tato obdoba jazyka Karel byla poprvé popsána v knize *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming* z roku 1996, jejímž spoluautorem je i tvůrce původního jazyka Richard E. Pattis. Jak už z názvu knihy vyplývá, tato implementace se soustředí na výuku objektově orientovaného programovacího paradigmatu. Samotný jazyk se svou skladbou více podobá jazyku Java či C++. [4]



Obrázek 2: Ukázka prostředí pro vývoj v jazyce Karel++ (Windows 3.1)

2.2.2 Karel 3D

Během let vzniklo několik implementací, které vývojové prostředí přeneslo do 3D. Tyto implementace přinášejí do Karlova robotího města možnosti vertikálního pohybu a s ním i celou škálu nových problémů k řešení. Jedním příkladem může být verze vytvořená na Slovensku, která implementuje jazyk Karel ve 3D prostředí a obsahuje jazykové mutace pro slovenštinu, angličtinu, němčinu a španělštinu. Webová verze aplikace je naprogramována v jazyce JavaScript a samostatně spustitelná aplikace poté v jazyce Java s pomocí frameworku JavaFX, podobně jako verze vytvořená v této práci. [5]

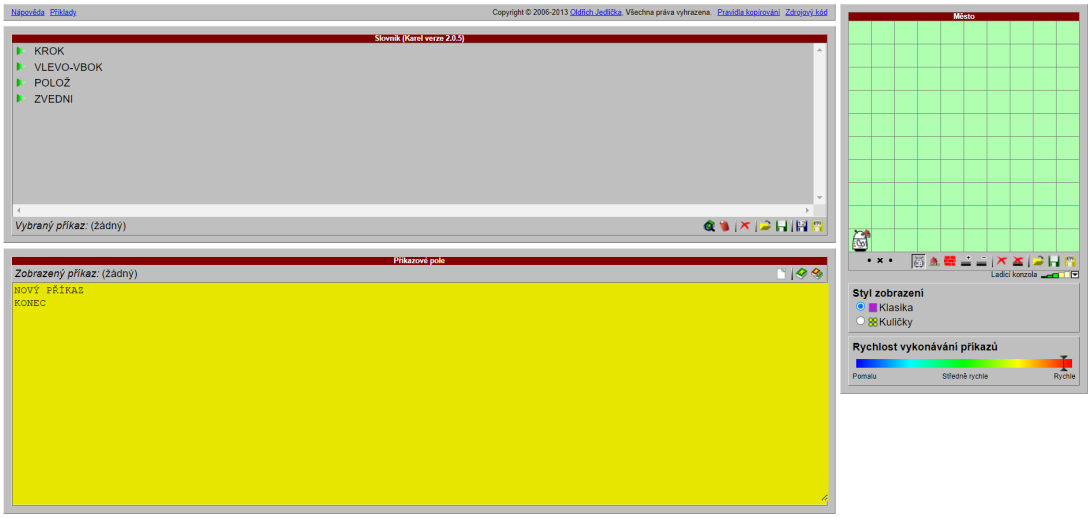


Obrázek 3: Ukázka 3D prostředí pro vývoj v jazyce Karel

2.2.3 Webové prostředí Robot Karel

Jednou z velkých inspirací pro tuto práci byla verze vytvořená českým vysokoškolským studentem Oldřichem Jedličkou, která funguje v prohlížeči a je celá napsána v jazyce JavaScript. Poslední verze této webové aplikace byla vydána

v roce 2006. Webové stránky hostující dané prostředí obsahují i návody a příklady pro snadnější pochopení aplikace i jazyka samotného. [6]



Obrázek 4: Ukázka webového prostředí pro vývoj v jazyce Karel

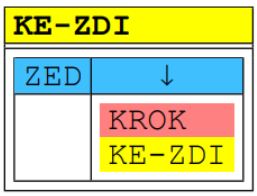
2.3 Využití

Jazyk Karel ve své základní podobě není určen pro produkční využití a programování složitějších aplikací. Jeho hlavní účelem je přiblížit programování začátečníkům, a to jak z řad dětí tak i dospělých. Je vhodný pro trénování logického myšlení uživatele a základních konceptů, se kterými se setká u programátorské praxe.

Pro mladší žáky prvního stupně ZŠ vznikl například učební materiál s názvem *Programovací jazyk Karel*. Jedná se o programovací příručku psanou z pohledu robota Karla, který čtenáři přibližuje práci s interpretem *PC-Karel*. Obsahuje nejrůznější problémy a řešení podaná hravou formou. Součástí materiálu jsou i speciálně vytvořené grafy pro znázorňování zdrojového kódu programu, které autoři nazvali *kopenogramy*. [7]

Kurióznější využití nalezla verze založená na jazyce Pascal, a to při programování robotů a kontrolérů od strojírenské skupiny FANUC. V této implementaci je možné provádět i složitější úkony jako je ethernetová komunikace, pokročilé zpracování logiky nebo správa souborů. Některé firmy ho využívají i jako nástroj, díky kterému mohou jejich zákazníci identifikovat problémy se svými pořízenými stroji. [8]

V neposlední řadě je i dnes využíván ve vysokoškolském prostředí ku příkladu právě na místě svého zrodu na kurzech programování Stanfordské univerzity, kde byl uzpůsoben výuce objektově orientovaného programování a stal se knihovnou pro jazyk Java a Python. [1, 9]




```
KE-ZDI ZNAMENA
  KDYZ JE ZED
  KONEC JINAK
    KROK
    KE-ZDI
  KONEC
KONEC
```

Obrázek 5: Ukázka kopenogramu vytvořeného ze zdrojového kódu

2.4 Syntaxe jazyka

V této části kapitoly bude představena mutace jazyka Karel použita ve výsledné aplikaci. Tato verze obsahuje několik přidaných příkazů, které usnadňují práci s jazykem a umožňují klást větší důraz na řešení složitějších problémů. Například je přidán povel k okamžitému otočení robota do protisměru, který musel v původní verzi být vždy simulován dvěma příkazy pro otočení. Dalším důležitým prvkem implementace, který nebyl přítomen v originálním jazyce, je možnost užití rekurze.

Veškerá syntaxe bude dále představena pouze pro českou mutaci jazyka. Při přepnutí do angličtiny jsou používaná přeložená klíčová slova, ale jejich význam v rámci jazyka Karel zůstává nezměněn.

2.4.1 Základní příkazy

KROK Karel se posune ve směru, kterým je otočen.

ČELEM-VZAD Karel se otočí opačným směrem.

VLEVO-VBOK Karel se otočí doleva.

VPRAVO-VBOK Karel se otočí doprava.

POLOŽ Karel položí značku na políčko, kde se aktuálně nachází, až do počtu maximálně 15 značek.

ZVEDNI Karel zvedne značku z políčka, kde se aktuálně nachází, pokud značka existuje.

ZASTAV Okamžitě zastaví vykonávání povelů a ukončí spuštěnou proceduru.

/* <KOMENTÁŘ> */ Mezi symboly /* a */ může být vložen libovolný komentář či kód, který bude při spuštění procedury ignorován. Může sloužit například jako vysvětlení části kódu. Jsou podporovány i víceřádkové komentáře.

2.4.2 Podmínky

Podmínky v jazyce Karel slouží k větvení programu, ale oproti jiným sofistikovanějším jazykům jsou omezeny jen na kontrolu aktuálního okolí, kde se robot Karel nachází. Karel může proto kontrolovat pouze počet a přítomnost značek na aktuálním políčku, případně zda se před ním nachází zeď nebo jestli se nachází na domovském políčku. Možné podmínky jsou následující:

JE-ZEĎ / NENÍ-ZEĎ Karel zkontroluje jestli se na políčku před ním nachází nebo nenachází zeď.

JE-ZNAČKA / NENÍ-ZNAČKA Karel zkontroluje jestli na políčku, kde se právě nachází, existuje alespoň jedna značka nebo zda neexistuje ani jedna značka.

JE-PLNO / NENÍ-PLNO Karel zkontroluje jestli je políčko, kde se právě nachází, zcela naplněné značkami nebo zda ještě není zcela naplněné značkami.

JE-DOMOV / NENÍ-DOMOV Karel zkontroluje jestli je políčko, kde se právě nachází, označené jako jeho domov nebo ne.

Využití podmínek pro větvení je realizováno skrze podmínkové výrazy, které je do sebe možné libovolně zanořovat. Jsou podporovány dva způsoby zápisu těchto výrazů:

1. `KDYŽ <PODMÍNKA>`
   `    <PŘÍKAZ>`
   `KONEC`

   Pokud je podmínka splněna provede se <PŘÍKAZ>, jinak je celý výraz ignorován.

2. `KDYŽ <PODMÍNKA>`
   `    <PŘÍKAZ-1>`
   `JINAK`
   `    <PŘÍKAZ-2>`
   `KONEC`

   Pokud je podmínka splněna provede se <PŘÍKAZ-1>, jinak je proveden <PŘÍKAZ-2>.

2.4.3 Cykly

Cykly slouží k opakovanému vykonávání kódu. V programovacích jazycích se zpravidla setkáme se třemi základními typy, a to s cyklem s podmínkou na začátku, s cyklem s podmínkou na konci nebo s cyklem ukončeným po předem stanoveném počtu opakování.

Cykly s podmínkou na počátku před každou další iterací kontrolují platnost podmínky a na základě toho vykonají kód nebo skončí, což je ve většině jazyků ekvivalentní struktuře cyklu *while*. Cykly s podmínkou za tělem cyklu jsou spouštěny alespoň jednou a po každé iteraci kontrolují platnost podmínky, což je často implementováno pomocí *do-while* cyklu. Cykly s předem daným počtem iterací mají na počátku udané libovolné číslo n rovno maximálnímu počtu opakování a proměnou i, která je nastavena na počátku na číslo 0 a postupně inkrementována, až dokud se $n = i$, kdy cyklus končí. Implementace tohoto druhu cyklu jsou označovány jako *for* cykly nebo *repeat* cykly.

Jazyk Karel podporuje v mutaci vytvořené pro tuto práci dva typy cyklů:

1. `DOKUD <PODMÍNKA>`
   `<PŘÍKAZ>`
   `KONEC`

   Karel vždy než vykoná <PŘÍKAZ> zkontroluje jestli je <PODMÍNKA> splněna, pokud není, přeskočí celý cyklus, jinak opakovaně vykonává <PŘÍKAZ>, dokud podmínka platí.

2. `OPAKUJ <ČÍSLO>(KRÁT)`
   `<PŘÍKAZ>`
   `KONEC`

   Karel zopakuje <PŘÍKAZ> tolikrát, kolik je uvedené <ČÍSLO>. Slovo KRÁT za počtem opakování nemusí být uvedeno.

2.4.4 Procedury

Procedury jsou základním stavebním kamenem jazyků spadajících do procedurálního programovacího paradigmatu, kterým je i jazyk Karel. Jsou to samostatné pojmenované části kódu, které často řeší jeden konkrétní problém a právě díky skládání více těchto částí kódu dohromady je programátor schopen řešit těžší a rozsáhlejší problémy.

Procedury jsou ve vytvořené aplikaci zpracovány pomocí tzv. knihovny procedur, skrze kterou lze procedury vytvářet, přejmenovávat a mazat. Pokud je procedura přidána do knihovny, je ji možno od této chvíle volat z jakékoliv procedury, která je v aktuální knihovně přítomna.

2.4.5 Rekurze

Rekurze představuje pokročilejší programátorský koncept, který pracuje s voláním procedury uvnitř sebe sama. Důležitou součástí rekurzivního volání je tzv. limitní podmínka. Její existence zaručuje, že řetězec volání bude při jejím splnění ukončen a nedojde k nežádoucímu zacyklení či přehlcení paměti zařízení.

Základní druhy rekurze, které můžeme rozlišovat jsou tyto:

- **Přímá** Procedura se drží výše zmíněného postupu a volá sebe samu ve svém těle. Přímá rekurze pro kód procedury s názvem *PROCEDURA-1* může vypadat například takto:

```
/* PROCEDURA-1 */
KDYŽ NENÍ-ZEĎ
   KROK
   PROCEDURA-1
KONEC
```

  Zde posloužila jako limitní podmínka existence zdi před robotem, pokud by se zde nenacházela, robot by pokračoval, dokud by nenarazil do zdi a kód by skončil chybou.

- **Nepřímá** Této rekurze se účastní více procedur, které svým vzájemným voláním tvoří kruh. Následující příklad převádí předchozí přímou rekurzi do nepřímé:

```
/* PROCEDURA-1 */
KDYŽ NENÍ-ZEĎ
   PROCEDURA-2
KONEC

/* PROCEDURA-2 */
KDYŽ NENÍ-ZEĎ
   KROK
   PROCEDURA-1
KONEC
```

Další dělení rekurze vychází z počtu rekurzivních volání procedury uvnitř sebe sama:

- **Lineární** V proceduře je uvedeno pouze jedno rekurzivní zavolání (přímé i nepřímé), které je vykonané za běhu procedury. To znamená, že můžeme mít i více volání v těle procedury, pokud jsou výlučně podmínkově odděleny. Příklady této rekurze jsou uvedeny výše.

- **Koncová** Jedná se o speciální případ lineární rekurze, kde je rekurzivní volání posledním příkazem v proceduře. To s sebou nese úspory paměti, jelikož se nemusí uchovávat příkazy následující za rekurzí. Tato rekurze může být nahrazena cyklem, což lze vidět na následujícím příkladu:

```
/* POKLÁDEJ-ZNAČKY-CYKLEM */
DOKUD NENÍ-PLNO
    POLOŽ
KONEC
```

```
/* POKLÁDEJ-ZNAČKY-REKURZÍ */
KDYŽ NENÍ-PLNO
    POLOŽ
    POKLÁDEJ-ZNAČKY-REKURZÍ
KONEC
```

- **Stromová** V proceduře je uvedeno více než jedno rekurzivní volání (přímé i nepřímé). V praxi se často setkáme například se dvěma voláními, kdy při grafickém znázornění vzniká binární strom volání, odtud také tento název. Pro n volání pak vznikají při znázornění stromy n-ární. Příkladem stromové rekurze může být následující procedura:

```
/* PROCEDURA-1 */
KDYŽ NENÍ-PLNO
    POLOŽ
    PROCEDURA-1
    KDYŽ NENÍ-ZEĎ
        KROK
        PROCEDURA-1
    KONEC
KONEC
```

3 Použité technologie

Technologie byly zvoleny takovým způsobem, který odpovídá koncepci multiplatformní desktopové aplikace běžící na platformě Java.

3.1 Jazyk Kotlin

Kotlin je moderní programovací jazyk vyvíjený firmou JetBrains, který svůj vznik datuje do roku 2011. Jedná se o multiplatformní, staticky typovaný jazyk využívající typovou inferenci a dedikované mechanismy pro práci s typy nabývajícími hodnoty *null*. Kombinuje v sobě objektově orientované paradigma s procedurálním paradigmatem, kdy lze například tvořit tzv. rozšiřující funkce nad objekty bez nutnosti tvorby dědičné třídy či tvořit proměnné a funkce mimo tělo třídy.

Kotlin se dostal do povědomí mnoha vývojářů v roce 2019, kdy se ujal místa preferovaného jazyka pro vývoj moderních mobilních aplikací pro zařízení se systémem Android, kde vystřídal jazyk Java.

Hlavní vývoj je soustředěn na platformu JVM (Java Virtual Machine), ale je možné v něm napsané zdrojové kódy kompilovat i do JavaScriptu či nativního kódu.

Velkou výhodou tohoto jazyka je jeho přímočará interoperabilita[1] právě s jazykem Java. To ve zkratce umožňuje využití již existujícího rozsáhlého ekosystému knihoven a funkcí pro platformu Java z prostředí moderního jazyka se všemi výhodami, které Kotlin programátorům přináší. [10]

3.2 SQLite

Pro účely perzistentního uchování dat aplikace byla zvolena knihovna SQLite. Ta implementuje robustní relační databázový systém v relativně malé knihovně napsané v jazyce C. SQLite databáze je specifická tím, že nevyužívá architektury klient-server jako komplexnější databáze typu například MySQL. Databáze místo toho uchovává jako samostatné soubory s příponou *.db*.

Jedná se o velmi oblíbenou verzi tzv. embedded databáze, která deleguje většinu managementu databáze na aplikaci, která ji využívá. Je oblíbenou volbou zejména pro mobilní aplikace a programy nevyžadující náročné databázové optimalizace a funkcionality. Zároveň funguje na všech operačních systémech, čímž splňuje podmínky pro výslednou multiplatformní aplikaci. [11]

3.3 Gradle

Gradle je nástroj pro podporu vývoje spravující automatizaci sestavování, testování, distribuce a publikování aplikací. Umožňuje tvorbu tzv. *tasks*, které mohou

[1] Zde ve smyslu možnosti volání kódu napsaného v jednom jazyce z kódu jiného jazyka a naopak.

být spouštěny paralelně či sériově na platformě JVM. Mimo jiné využívá orientovaných acyklických grafů pro zjišťování závislostí mezi balíčky a knihovnami zahrnutými v aplikaci, díky čemuž je schopen určit pořadí vykonání úkonů zajišťující správné sestavení aplikace. [12]

Staví na konceptech podobných nástrojů jako Apache Ant nebo Apache Maven a přidává možnost využití doménově specifických jazyků založených na Groovy nebo Kotlinu místo XML souborů vyskytujících se právě u zmíněného nástroje Maven.

Ve vytvářené aplikaci je Gradle využit pro správu závislostí, sestavování aplikace a tvorbu *.jar* souborů.

3.4 TornadoFX

Jde o aplikační rámec pro tvorbu uživatelských rozhraní. Je postaven na platformě JavaFX, která se využívá pro tvorbu desktopových aplikací pro jazyk Java. TornadoFX si klade za cíl zjednodušit tvorbu uživatelských rozhraní právě pomocí užití jazyka Kotlin, což vede k menšímu množství kódu a směřuje vývoj deklarativním směrem, podobně jako se tomu dnes děje u tvorby uživatelských rozhraní v mobilních a webových aplikacích, díky moderním knihovnám jako ReactJS, SwiftUI nebo Jetpack Compose. [13]

Hlavní předností TornadoFX je možnost využít přibalených funkcionalit jako je vkládání závislostí, delegované atributy, implicitní vícejazyčná podpora pro aplikaci, možnost jednoduchého rozšíření existujících ovládacích prvků z JavaFX nebo jednoduché mapování objektů datového modelu do formátu JSON. Zmíněné funkcionality jsou umožněny, právě díky užití moderní nadstavby jazyka Kotlin nad širokými základy platformy JavaFX. [14]

3.4.1 Sběrnice událostí

Ve vytvářené aplikaci je také hojně využíváno tzv. sběrnice událostí, která běží v pozadí TornadoFX a umožňuje striktnější oddělení objektů starajících se o vizualizaci aplikace od objektů zaštiťujících její logickou část. Celý proces pracuje na principu zasílání zpráv, což jsou objekty třídy *FXEvent*, které lze libovolně rozšířit o informace, které chceme, aby zpráva nesla. Pro reakci na odeslanou zprávu stačí registrovat odběratele uvnitř třídy, která má na konkrétní událost reagovat. Mimo jiné je i možné určit, zda chceme, aby se jednalo o událost vykonávanou na hlavním vlákně (zpravidla ovlivňující vizuální stránku) nebo na vlákně v pozadí (zpravidla náročnější či I/O operace). [15]

3.4.2 CSS

Pro změnu vzhledu aplikace můžeme podobně jako u JavaFX využít tzv. *stylesheets* neboli soubory obsahující CSS pravidla. Hlavní rozdíl spočívá v tom, že obsahem není prosté textové CSS, ale je možné ho psát pomocí jazyka Kot-

lin, čímž umožníme typovou kontrolu pravidel, našeptávání či rozdělení skupin pravidel do různých tříd. [16]

```
class MyStyle: Stylesheet() {

    companion object {
        val tackyButton by cssclass()

        private val topColor = Color.RED
        private val rightColor = Color.DARKGREEN
        private val leftColor = Color.ORANGE
        private val bottomColor = Color.PURPLE
    }

    init {
        tackyButton {
            rotate = 10.deg
            borderColor += box(topColor,rightColor,bottomColor,leftColor)
            fontFamily = "Comic Sans MS"
            fontSize = 20.px
        }
    }
}
```

Obrázek 6: Ukázka tvorby CSS stylů v TornadoFX

3.5 Rozpoznávání hlasu od firmy Phonexia

Tato práce vznikla ve spolupráci s firmou Phonexia, jejíž software pro rozpoznávání hlasu byl se svolením jejích zástupců využit pro realizaci hlasového ovládání přítomného v aplikaci.

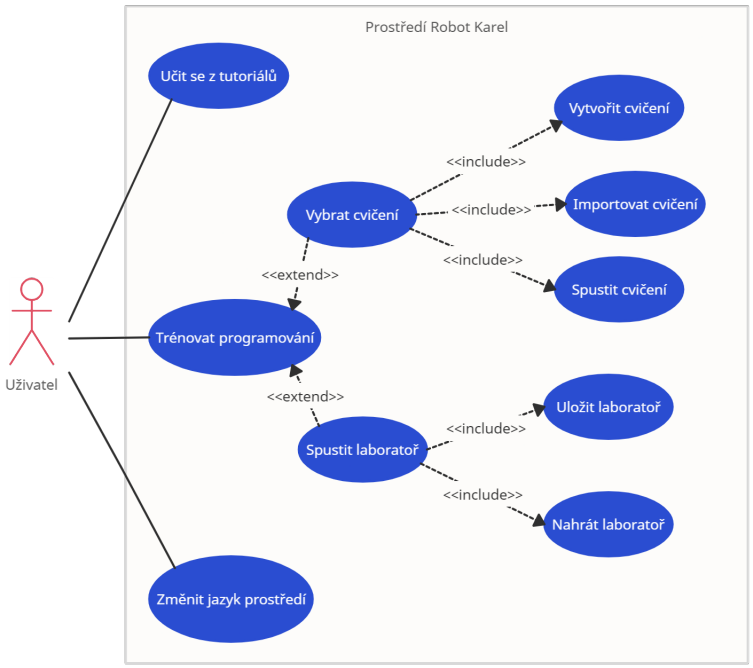
Firma poskytla komplexní řešení problematiky rozboru lidské řeči ve formě serverové aplikace. Ve spojení s vytvářenou aplikací byla využita výhradně část řešení zahrnující rozpoznávání klíčových slov ze souvislého mluveného projevu. To umožnilo realizaci funkcionalit jako diktování kódu, diktování pokynů pro pohyb v bludišti a možnost textové odpovědi na uživatelem položené tématické otázky vztahující se k aplikaci. Díky tomu, že propůjčená verze obsahuje jak model pro český, tak i pro anglický jazyk, bylo možné funkce implementovat pro oba zmíněné jazyky.

```
// Definujeme zprávu, která poběží ve vlákně na pozadí.
object CustomerListRequest : FXEvent(BackgroundThread)


// Ve třídě vizualizace definujeme tlačítko, které pro kliknutí rozešle zprávu.
button("Load customers").action {
    fire(CustomerListRequest)
}


// Definujeme zprávu, která v sobě jako argument nese seznam zákazníků a poběží na hlavním vlákně.
class CustomerListEvent(val customers: List<Customer>) : FXEvent()

// Ve třídě kontroleru registrujeme reakci na zprávu z UI spravovanou vláknem na pozadí.
// Po obdržení vykonáme na tom samém vlákně načtení z databáze.
// Nakonec odešleme novou zprávu na hlavní vláknou s načtenými daty pro aktualizaci UI.
class CustomerController : Controller() {
    init {
        subscribe<CustomerListRequest> {
            val customers = loadCustomers()
            fire(CustomerListEvent(customers))
        }
    }

    fun loadCustomers(): List<Customer> = db.selectAllCustomers()
}
```

Obrázek 7: Ukázka práce s událostmi v TornadoFX

4 Architektura aplikace

Cíl aplikace lze rozdělit do několika částí. Za prvé je potřeba uživatele seznámit s jazykem Karel a možnostmi, které mu dané prostředí nabízí. Za druhé je nutné mít prostor pro samotnou realizaci programování a vizualizaci bludiště včetně jeho editace. Za třetí by uživatel měl mít možnost změnit jazyk aplikace, což zahrnuje změnu textů, hlasového ovládání a samotného programovacího jazyka.



Obrázek 8: Diagram případů užiti aplikace

4.1 MVC

Celá aplikace se řídí architekturou MVC neboli Model-View-Controller, která se zaměřuje na jasné oddělení datové, logické a vizuální stránky aplikace. Model slouží pro definici doménových objektů a v aktuálním případě se jedná o modely bludiště, programovaných procedur, tutoriálů, cvičení a laboratoří. Názvem View (pohled) se označují jednotlivé třídy sestavující uživatelské rozhraní například podobu bludiště. Controller (ovladač) spravuje logiku aplikace a slouží jako prostředník mezi pohledy a modely. V aplikaci se vyskytuje několik ovladačů, například pro správu cvičení, bludiště nebo hlasového ovládání.

TornadoFX tuto architekturu podporuje, díky implicitním třídám *JsonModel*, *View* a *Controller*, které frameworku umožňují správu objektů vytvořených pomocí konkrétních odvozených tříd. To znamená umožnění automatické registrace těchto objektů ke sběrnici událostí a k jejímu užívání, možnost automatického

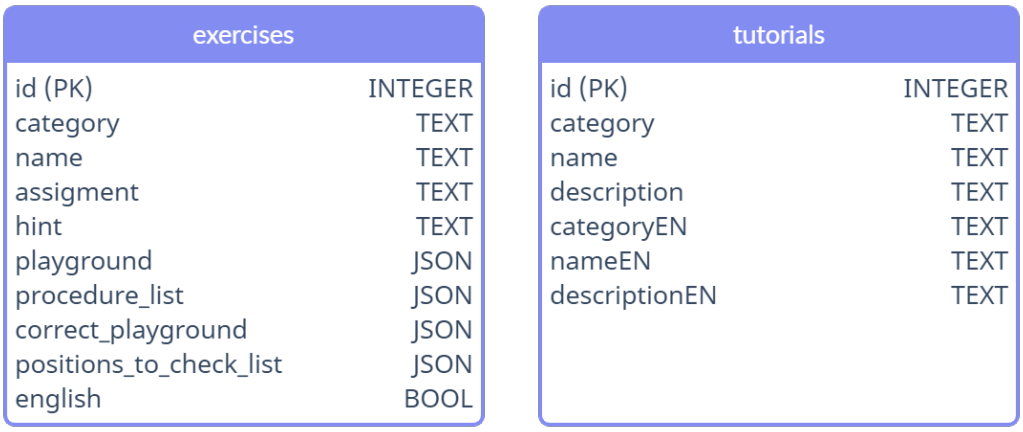
vkládání závislostí pro tvořené objekty či možnost převádění modelů z a do odpovídajících JSON reprezentací.

4.2 Databáze

Databáze obsahuje dvě tabulky s názvy *tutorials* a *exercises*, které uchovávají převážně textové informace o jednotlivých cvičeních a tutoriálech přístupných v aplikaci. Díky rozšíření přítomném v SQLite pro práci s JSON objekty jsou části cvičení popisující stav bludiště a programované procedury uloženy právě v tomto formátu pro větší kompaktnost.

Všechna uživatelsky vytvořená a importovaná cvičení jsou ukládána do odpovídající tabulky, oproti tomu tabulka s tutoriály je koncipována jako pouze pro čtení a během života aplikace se nemění.

K databázi je z kódu přistupováno skrze repositáře, které mají za úkol provádět všechny základní CRUD[2] operace nad existujícími tabulkami.



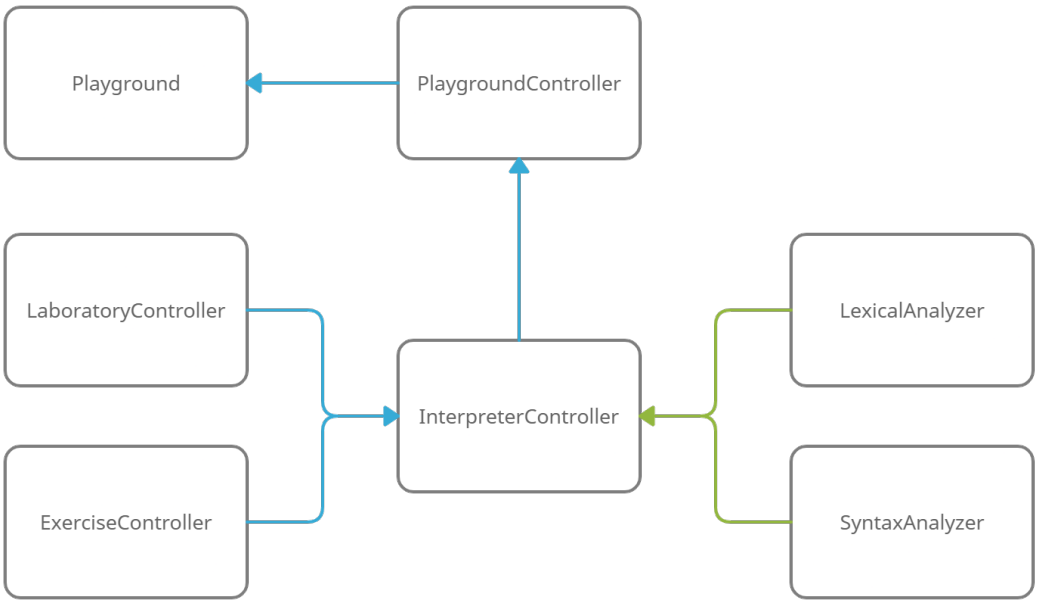
Obrázek 9: Schéma databáze

4.3 Prostředí jazyka Karel

Kompletní prostředí jazyka se skládá ze 3 částí: knihovny procedur, editoru kódu a bludiště. To platí stejně jak pro programování v laboratoři, tak pro programování cvičení. Pojítko mezi nimi tvoří třída *InterpreterController* volaná při spuštění procedury z editoru, která na uživatelský kód doplněný o aktuální knihovnu procedur aplikuje lexikální a syntaktickou analýzu[3] Nakonec výsledný kód, pokud je validní, interpretuje a převede na pohyb robota v bludišti. Toto prostředí je implementováno v části aplikace zvané *Laboratoř* a v sekci *Cvičení*.

[2]Shrnuje čtyři základní operace nad záznamem v trvalém úložišti: tvorbu, čtení, editování a mazání.

[3]Samostatná sémantická analýza je z důvodu jednoduchosti jazyka nepotřebná.



Obrázek 10: Znázornění závislostí tříd

4.3.1 Interpret jazyka Karel

Prvním krokem po zavolání interpretu je provedení lexikální analýzy nad kódem spuštěné procedury, a to včetně všech dalších v ní volaných procedur z knihovny. Během tohoto procesu je kód očištěn od komentářů a následně čten po řádcích[4], ve kterých jsou hledána klíčová slova. Při hledání je odkazováno na tabulku výčtových typů, které v rámci aplikace reprezentují klíčová slova. Z takto rozpoznaných klíčových slov jsou tvořeny lexémy (reprezentovány odpovídajícími výčtovými typy) a ty jsou ukládány do hašovací tabulky spolu s číslem řádku jeho výskytu a případným argumentem kontrolovaným až během dalšího kroku. Pokud během hledání došlo k chybě, kdy nebylo rozpoznáno klíčové slovo na řádku, je chyba zapsána do chybové tabulky s číslem řádku a druhem chyby. Pokud není po skončení analýzy tabulka chyb prázdná proces interpretování se přeruší a uživatel je na nalezené chyby upozorněn modálním oknem.

Výsledná tabulka z úspěšné lexikální analýzy je v dalším kroku předána syntaktickému analyzátoru, který postupně prochází jednotlivé položky od prvního do posledního řádku. Z validních položek tvoří syntaktické uzly obsahující kromě samotných klíčových slov i rozpoznané argumenty a čísla řádků svých koncových či příbuzných uzlů v případě výrazů cyklů a podmínek. Každý vytvořený uzel se nakonec vloží do tabulky uzlů. Během toho se navíc kontroluje, zda jsou případné argumenty příkazů ve validním tvaru a zda je přítomen správný počet uzavíracích klíčových slov KONEC. V případě, že je nalezena chyba, se stejně

[4]V jazyce Karel může jeden řádek obsahovat maximálně jeden příkaz případně příkaz s podmínkou či argumentem.

jako u lexikální analýzy proces dokončí a až poté je interpretace přerušena a uživateli zobrazeny všechny nalezené chyby.

Poté, co dostane interpret výslednou tabulku se syntaktickými uzly, se spustí samotný proces postupné evaluace příkazů. Tady je nutná spolupráce se třídou *PlaygroundController*, která zjišťuje aktuální stav bludiště a poskytuje dodatečné informace nutné pro evaluaci, například kam je natočen robot nebo jestli je na políčku před ním zeď. Chyba vyvolaná za běhu evaluace způsobí její okamžité přerušení a upozornění uživatele. Sám uživatel také může probíhající evaluaci přerušit opětovným kliknutím na tlačítko spuštění.

4.3.2 Bludiště

Bludiště je jedním z hlavních prvků programovacího rozhraní, na kterém je vizualizováno vykonávání uživatelského kódu. Sestává se ze sítě 10x10 políček a vizuálních prvků, které jsou na políčka umísťovány. V kódu je reprezentováno třídou *Playground* obsahující údaje o aktuální pozici a směru robota, jeho počátečním umístění, druhu značky, který má být v bludišti vykreslován, a nakonec i mapu pozic s políčky reprezentovanými třídou *PlaygroundTile*. Ta uchovává informace o stavu políčka promítaného do vizuální podoby, například jestli se na něm nachází zeď.

V každém bludišti jsou přítomny minimálně dva vizuální prvky, a to postavička robota a domeček, nejčastěji[5] značící počáteční místo robota. Dále se zde mohou vyskytovat zdi, které označují políčka, na která nelze vstupovat a při pokusu o vstup je ohlášena chyba.[6] Políčka bez zdí mohou obsahovat značky vyobrazeny s texturou jednoho ze čtyř předem připravených modelů až do maximálního povoleného počtu 15 značek na políčko.

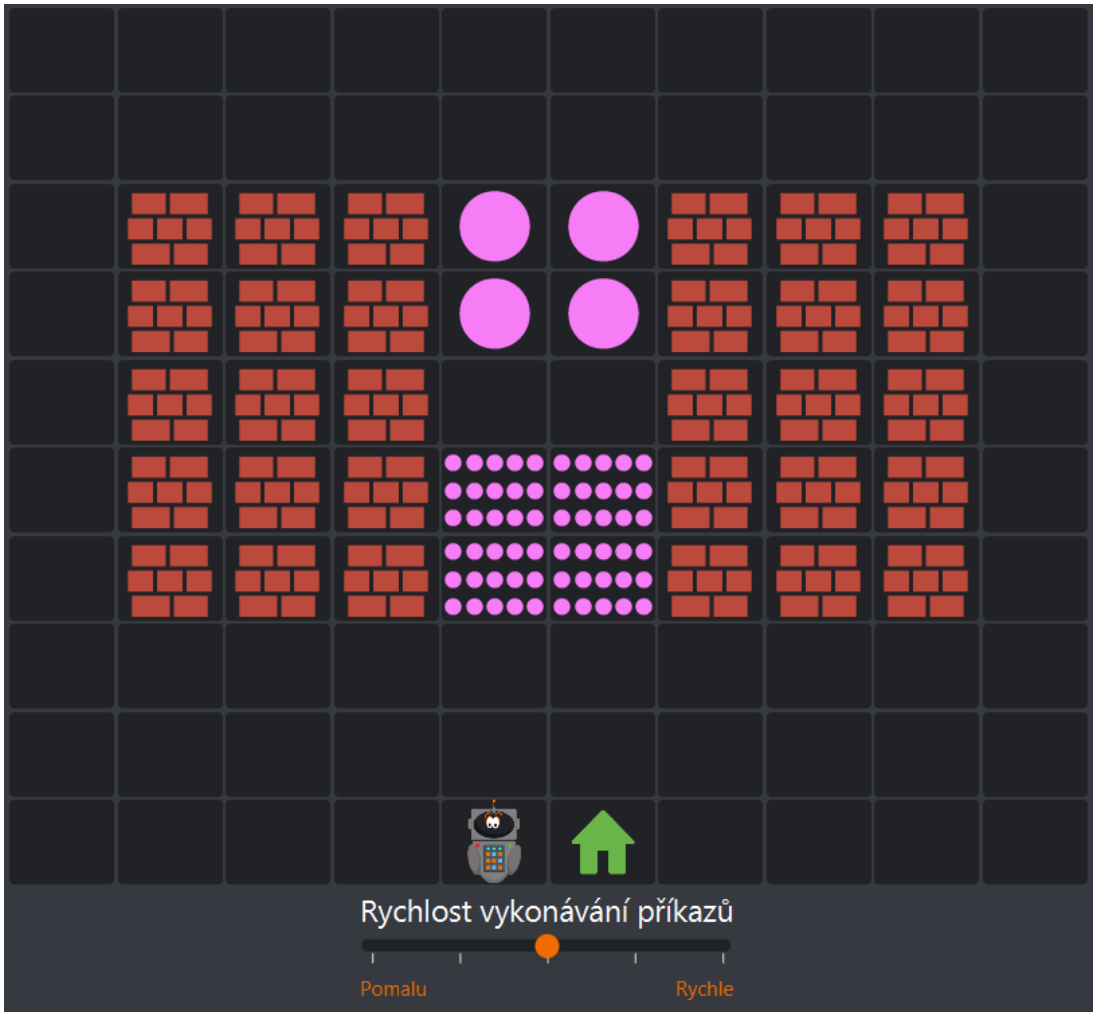
Dále se pod každým bludištěm nachází posuvník určující rychlost s jakou jsou vykonávány příkazy spuštěného kódu. Uživatel si může vybrat z pěti možných rychlostí od nejpomalejší s příkazovou latencí 200 ms až po nejrychlejší s latencí 10 ms.

4.3.3 Editace bludiště

O editaci bludiště se stará třída *PlaygroundEditController*, která na počátku editace zkopíruje výchozí stav bludiště a poté na nové kopii tohoto bludiště vykonává akce uživatele. Uživatelovy akce skrze zmíněnou třídu transformují jednotlivé objekty políček, například tím, že je do nich přidána zeď. Akce je poté pomocí zprávy zaslané do sběrnice událostí doručena k odpovídající třídě reprezentující vzhled bludiště, která aktualizuje vzhled políčka a vykreslí na něm zeď.

[5] Některá cvičení domeček využívají i pro znázornění správného konečného umístění robota.

[6] Samotné okraje bludiště jsou považovány za zdi, proto výstup mimo bludiště skončí také chybou.



Obrázek 11: Příklad bludiště

4.3.4 Laboratoř

O správu této části aplikace se stará třída *LaboratoryController*. Jejím hlavním úkolem je vytváření, ukládání a načítání laboratoří. Laboratoř je ukládána do souborů s koncovkou *.klab*, v nichž je uložen JSON objekt odpovídající třídě *Laboratory*, který obsahuje vše potřebné pro její opětovné načtení včetně jména, knihovny procedur a stavu bludiště. Výchozí místo pro ukládání je adresář */saved/lab* ve složce s aplikací.

Mimo jiné v sobě tato třída obsahuje závislosti na *PlaygroundController* pro aktuálně zobrazené bludiště, *IntepreterController* pro vykonávání kódu a dvě třídy pro správu hlasového ovládání *CodeDictionController* a *PlaygroundVoiceController*.

4.3.5 Cvičení

O správu cvičení se stará třída *ExerciseController*, která má za úkol načíst z přidružené databáze všechna spustitelná cvičení včetně těch, která si uživatel importoval či vytvořil. Dále má za úkol umožnit uživateli exportovat svá cvičení do souborů s koncovkou *.kcvi*, která obsahují podobně jako u laboratoře JSON objekt odpovídající třídě *Exercise*. Výchozí místo pro ukládání je adresář */saved/exercises* ve složce s aplikací.

Je důležité zmínit, že cvičení v sobě zahrnují proměnnou udávající, zda se jedná o cvičení vytvořené v anglickém prostředí aplikace. Jelikož se s přepnutím jazyka prostředí mění i verze jazyka Karel, nejsou spolu cvičení napříč jazyky kompatibilní. Pro import cvičení v jiném než aktuálně zvoleném jazyce je potřeba přepnout jazyk prostředí.

Jednou z dalších hlavních funkcí je i spouštění a kontrola cvičení. Kontrola probíhá mezi stavem bludiště po vykonání uživatelského kódu dodaného do řešení a stavem korektního výsledného bludiště, které je součástí každého objektu *Exercise*. Mohou nastat dva případy kontroly:

1. Je kontrolováno každé políčko s každým a všechna musí navzájem korespondovat včetně konečné pozice robota (shoda směru není kontrolována).

2. Jsou kontrolována jen ta políčka, která byla vybrána při tvorbě daného cvičení.

Ať už dojde k jakékoliv z kontrol, při jejím úspěchu je zobrazeno modální okno potvrzující správnost řešení, v opačném případě je zobrazeno okno s miniaturou bludiště a popisem všech nekorespondujících políček.

4.3.6 Tvorba cvičení

Tvorba cvičení je řízena třídou *ExerciseCreationController*. Na počátku je vytvořen nový objekt třídy *Exercise* s výchozími hodnotami. Postupně při uživatelově průchodu procesem tvorby je uložena výchozí podoba bludiště, následně jsou dodány procedury a pomocí přepínače je označena procedura, která má být pro dané cvičení při kontrole spuštěna. Dále je vytvořeno korektní koncové bludiště, vůči kterému budou výsledná řešení porovnávána. Zde uživatel zvolí, zda chce kontrolovat všechna políčka, nebo pouze vybraná. Nakonec uživatel vyplní údaje jako název, zadání a volitelně i nápovědu k cvičení. Po vytvoření cvičení je uloženo do databáze a v aplikaci zobrazeno uživateli v kategorii označené jako *Vlastní*. Zde může svá cvičení spouštět, exportovat či mazat.

4.4 Hlasové ovládání

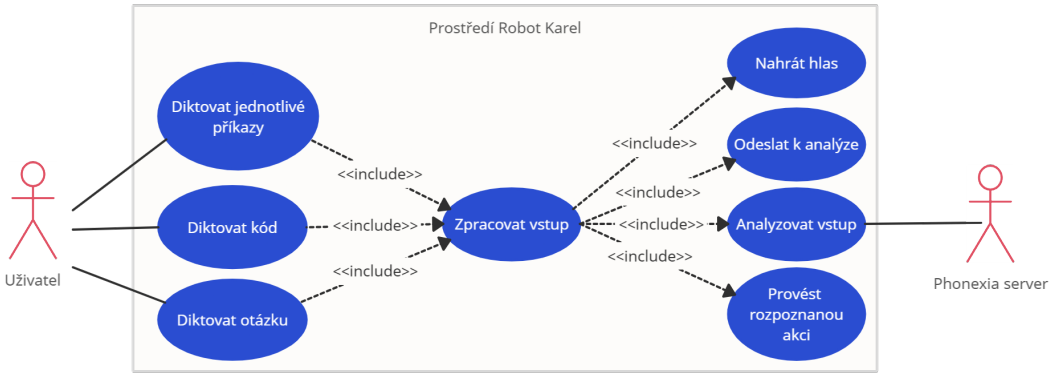
Hlasové ovládání se v aplikaci skládá z několika prvků. Prvním z nich je třída *MicrophoneRecorder*, jak z názvu už plyne, jedná se o třídu, která má na starosti nahrávání a ukládání uživatelova hlasu. Jako výstup je zvolen formát audio

souborů WAV, které v tomto případě nesou 1-kanálové mono audio o vzorkovací frekvenci 16kHz, a to z důvodu kompatibility se serverem Phonexia.

Klíčovou součástí celého procesu na úrovní infrastruktury je třída *PhonexiaAPI*. Ta komunikuje s vystavenou REST[7] API, kterou server od firmy Phonexia při spuštění zpřístupňuje. K tomuto účelu je na straně aplikace využit REST klient již zabudovaný v TornadoFX, přes který je vedena veškerá komunikace.

Při každém zapnutí aplikace je vytvořeno sezení, pod kterým aplikace se serverem komunikuje. Dále jsou při prvním spuštění odeslána všechna klíčová slova, která má server v aplikaci rozpoznávat. Ta jsou roztříděna do vytvořených seznamů na serveru podle kontextu jejich užití na *chatbot*, *diction* a *playground*. V tom samém volání je odesláno i pro jaký model jazyka jsou tato slova registrována v závislosti na současném jazyku prostřední. Navíc lze pro každé slovo určit do jaké míry si analyzátor musí být jistý (interval [0, 1]), aby slovo v odpovědi vrátil jako rozpoznané, což otevírá prostor optimalizacím rozpoznávání.

Hlavní třídou pro správu komunikace se serverem a jeho provoz je *PhonexiaController*, která využívá funkcí *PhonexiaAPI*. Nese v sobě metody, které s každým startem aplikace spustí proces serveru Phonexia, zkontrolují zda jsou klíčová slova na serveru načtena a při ukončení aplikace server opět vypíná. Je nutné podotknout, že toto chování je žádoucí pouze ve vývojovém a testovacím prostředí. Je předpoklad, že v produkčním prostředí bude server přístupný například ze školní sítě a nebude distribuován spolu s jednotlivými instalacemi aplikace.



Obrázek 12: Případy užití hlasového ovládání

Pro konkrétní hlasové funkcionality existují tři řídící třídy, a to *ChatbotVoiceControler*, *PlaygroundVoiceController* a *CodeDictionVoiceController*, které se starají o logiku nastíněnou v následujících kapitolách. Zároveň všechny tři ke svému fungování vyžadují připojený mikrofon. V případě, kdy není na aktuálním systému dostupný, je při pokusu o využití těchto funkcí zobrazena chybová hláška o chybějícím mikrofonu.

[7]Druh architektury rozhraní pro správu informací na serveru pomocí užití HTTP protokolu.

4.4.1 Chatbot

Na úvodní obrazovce je přítomen minimalistický chatbot[8], který umožňuje uživateli diktovat předem definované otázky pomocí hlasového vstupu. Celý proces funguje tak, že je po kliknutí na tlačítko mikrofonu zahájeno nahrávání hlasu a po opětovném kliknutí na tlačítko, nebo po maximálním intervalu 8 sekund se nahrávání ukončí. Následně je interně zpracované audio odesláno pomocí metody POST na server Phonexia. V dalším kroku je server dotázán na výsledek analýzy pomocí technologie rozpoznání klíčových slov. Odesílá se přitom požadavek typu GET obsahující argumenty s názvem analyzovaného audio souboru, jazykovým modelem a názvem seznamu klíčových slov, vůči kterému má analýza probíhat. Po obdržení výsledku je provedeno jeho vyhodnocení skrze porovnání s hodnotami výčtových typů otázek a při shodě je zobrazena odpověď.

4.4.2 Dikce příkazů v bludišti

V sekci *Laboratoř* je možné využít hlasového ovládání k zadání příkazů, které má robot vykonat přímo v bludišti bez nutnosti psát kód. Při spuštění nahrávání je se serverem Phonexia otevřeno spojení přes HTTP stream[9], přičemž jsou stanoveny parametry odesílaných dat, konkrétně počet kanálů a frekvence audia.

Do streamu je po krátkých časových intervalech kontinuálně pomocí metody PUT zasílán uživatelský hlasový vstup a zároveň dochází k opakovanému dotazování na průběžné výsledky analýzy daného streamu. V případě navrácení výsledku a jeho úspěšného porovnání s výčtovým typem jednoho z povolených příkazů je kontaktován *PlaygroundController*, který provede daný příkaz. Dikce může být zastavena uživatelem nebo automaticky po uplynutí maximální doby 3 minut. Po ukončení je HTTP stream na serveru uzavřen.

Mezi podporované příkazy patří:

- KROK
- ČELEM-VZAD
- VPRAVO-VBOK
- VLEVO-VBOK
- ZVEDNI
- POLOŽ
- DOMŮ

[8] Označení pro software, který slouží pro automatizovanou komunikaci s uživateli.

[9] Technika postupného odesílání dat ze serveru na klienta přes jedno HTTP spojení, které zůstává otevřeno po neomezenou dobu. Pro použití této techniky je nutné, aby server do hlavičky požadavku specifikoval parametr *Transfer Encoding: chunked*. [17]

4.4.3 Dikce kódu

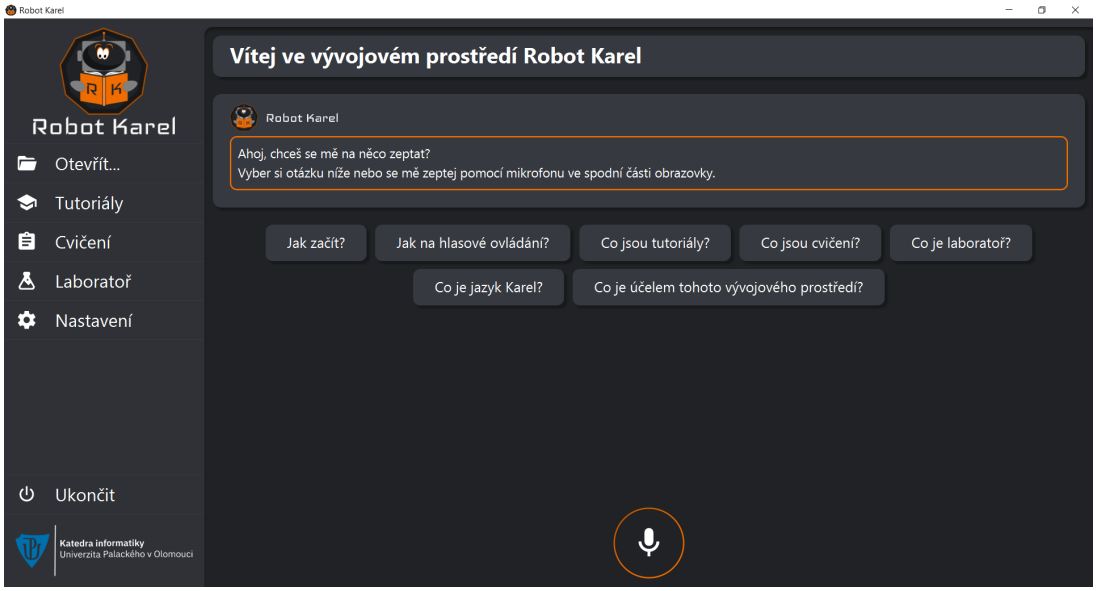
V každém editoru kódu v aplikaci je možnost zapnout funkci diktování kódu. Jeho fungování je po technické stránce vesměs ekvivalentní se dříve zmíněným diktování příkazů jen s tím rozdílem, že je využit jiný seznam klíčových slov, který v sobě obsahuje všechny příkazy a podmínky obsažené v jazyce Karel.

Poté, co je z HTTP streamu obdržen výsledek, porovná se s výčtovými typy všech klíčových slov a při shodě jsou slova postupně přidávána na konec textového pole obsahujícího kód zvolené procedury. Pokud je rozpoznáno několik validních klíčových slov za sebou je aplikováno i formátování kódu, konkrétně jsou přidávány mezery před podmínkami, nové řádky za příkazy a odsazení závislé na úrovni zanoření příkazů.

5 Uživatelská příručka

Následující sekce se věnuje představení prostředí z uživatelského pohledu spolu s uvedením funkcí, které jednotlivé obrazovky nabízí. Uživatel se mezi jednotlivými obrazovkami prostředí přesunuje pomocí menu umístěného po levé straně aplikace.

5.1 Úvodní obrazovka



Obrázek 13: Úvodní obrazovka

Při spuštění aplikace je uživateli zobrazena úvodní obrazovka. Na ní je přítomno několik tlačítek s otázkami, které by uživatel mohl mít během seznamování se s prostředím. Odpovědi na otázky jsou stylizované do formy chatu se samotným robotem Karlem.

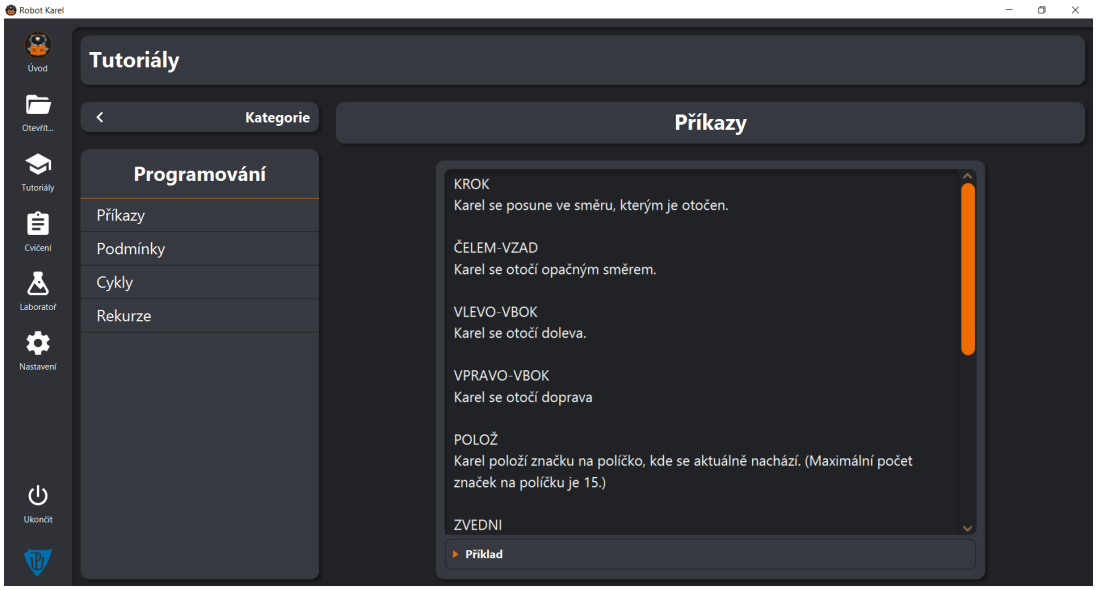
V dolní části je umístěno tlačítko pro spuštění hlasového nahrávání otázky, kterou uživatel, který má připojený mikrofon, může využít.

Cílem obrazovky je seznámit uživatele s prostředím, ve kterém se ocitl, a případně mu představit možnost hlasového ovládání aplikace.

5.2 Funkce otevření souborů

V menu je tato funkce přítomna pod položku označenou jako **Otevřít...**. Ta umožňuje načtení laboratoře či cvičení (dle vybraného typu souboru) skrze průzkumník souborů. Při vybrání souboru je uživatel přesunut na obrazovku laboratoře/cvičení, kde může začít s prostředím ihned interagovat.

5.3 Tutoriály



Obrázek 14: Tutoriály

Obrazovka tutoriálů má za úkol uživatele seznámit s obsahem a způsoby interakce s aplikací. Tutoriály jsou rozděleny do tří kategorií:

Aplikace Zde se uživatel dozví o jednotlivých částech aplikace.

Rozhraní V této kategorii jsou představeny možnosti interakce s programovacím rozhraním včetně hlasového ovládání.

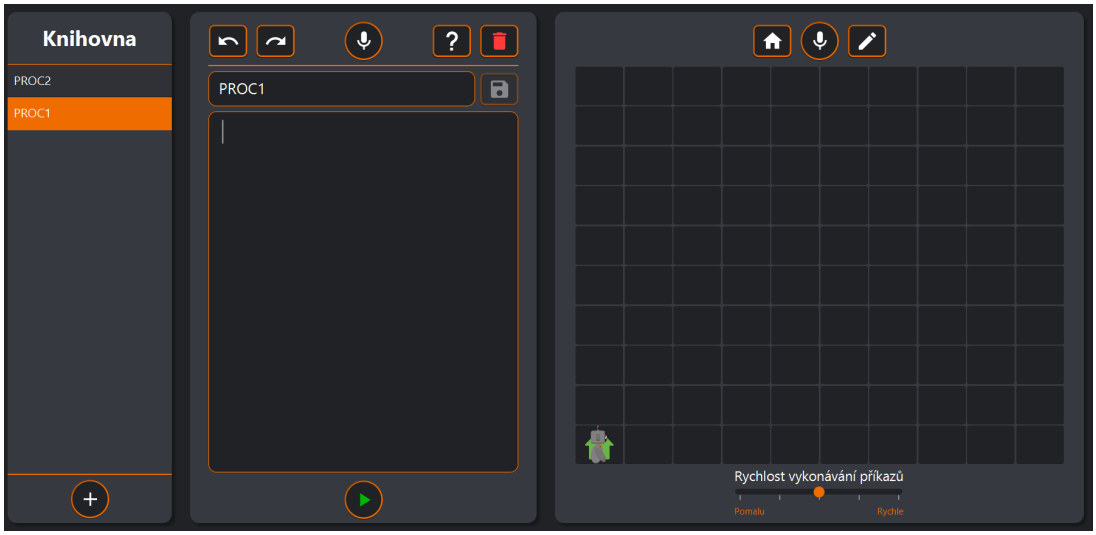
Programování Zde se uživatel může seznámit s jazykem Karel a základními koncepty programování, které lze při používaní tohoto jazyka aplikovat. K jednotlivým tutoriálům jsou v této sekci přidány i obrázkové příklady.

5.4 Programovací rozhraní

Jedná se o obrazovku složenou z několika částí, která je přítomna na více místech napříč aplikací. Části, které ji tvoří jsou knihovna procedur, editor kódu a bludiště.

Knihovna procedur se nachází po levé straně rozhraní a jsou v ní uchovány všechny vytvořené procedury v rámci aktuálního kontextu rozhraní. Uživatel může přidat novou proceduru pomocí kliknutí na tlačítko plus v dolní části knihovny, což vygeneruje proceduru a přidá ji na vrchol seznamu procedur.

V prostřední části rozhraní je přítomen editor kódu. V jeho vrchní části je řada tlačítek počínající dvěma pro akce vrácení a opětovného aplikovaní poslední akce v editoru. Dále je zde tlačítko pro spuštění dikce kódu, které se po kliknutí



Obrázek 15: Ukázka programovacího rozhraní v kontextu *Laboratoře*

změní na červenou tečku značící probíhající nahrávání. Během dikce jsou omezeny některé funkce včetně vpisování kódu a jeho spouštění. Poslední tlačítko je pak určeno ke smazání aktuální procedury z knihovny procedur.

Pod řadou tlačítek je přítomno textové pole s názvem zobrazené procedury, který lze přepsat[10] a následně uložit pomocí tlačítka umístěného vedle.

Samotný editor kódu je textové pole, které v sobě implementuje našeptávání klíčových slov a názvů procedur z knihovny. Ve spodní části se nachází tlačítko, které spustí aktuální proceduru, a v případě kdy procedura již běží, ji po kliknutí zastaví.

Pravá strana zahrnuje bludiště, na kterém jsou vykreslovány akce robota. Horní část se mění na základě kontextu zobrazení rozhraní, možné kontexty jsou tyto:

Laboratoř V laboratoři jsou přítomná tlačítka pro vrácení robota na počáteční pozici, pro spuštění diktování příkazů pomocí hlasu a tlačítko editace bludiště.

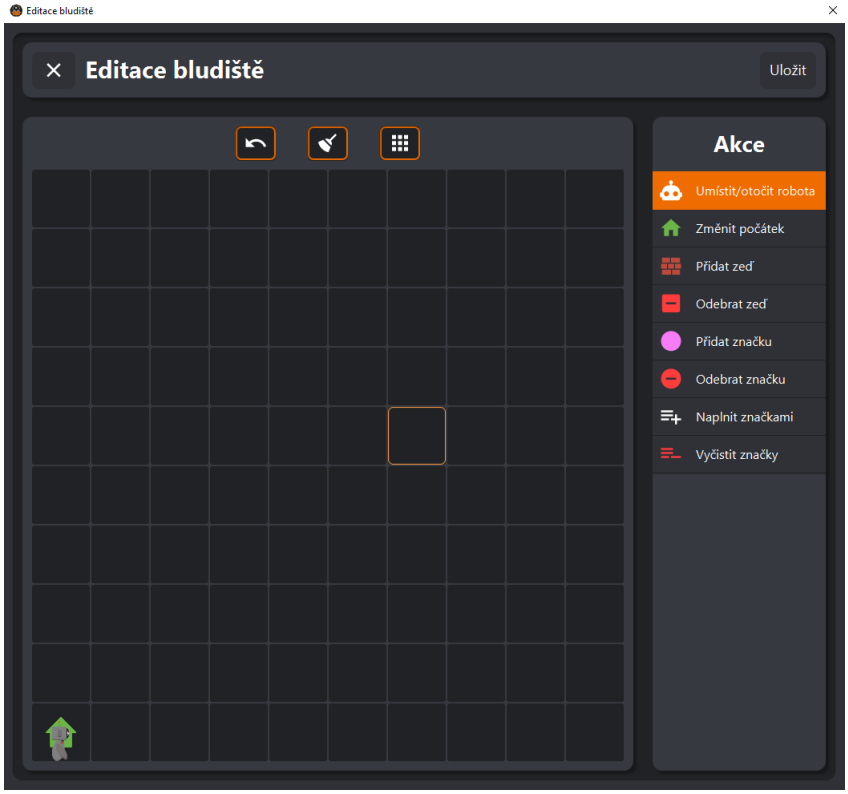
Cvičení Během spuštění či tvoření cvičení je přítomno pouze tlačítko pro obnovení výchozího vzhledu startovního bludiště.

Spodní část obsahuje pěti úrovňový posuvník, kterým uživatel může zvolit rychlost vykonávání příkazů.

5.4.1 Editace bludiště

Tato část rozhraní se stará o změnu podoby bludiště. Uživatel má k dispozici náhled aktuálního bludiště, které může ovlivňovat pomocí akcí uvedených v seznamu po pravé straně obrazovky. Nabízené akce se liší na základě kontextu,

[10]Názvy procedur mohou mít maximálně 30 znaků, kvůli lepší čitelnosti v rámci prostředí.



Obrázek 16: Ukázka editace bludiště v kontextu *Laboratoře*

ve kterém je editace použita. V kontextu sekce *Laboratoř* je možné využít plný rozsah akcí viz 16. V kontextu sekce *Tvorba cvičení* je obrazovka editace použita dvakrát. Poprvé k tvorbě startovního bludiště, kde jsou akce stejné jako u sekce *Laboratoř* a pro tvorbu korektního koncového bludiště, kde jsou odebrány možnosti, které nemůže uživatel pomocí Karla během cvičení vykonat, jako například přidávat zdi. Navíc je v druhém případě přidáno tlačítko pro označení kontrolovaných políček.

Nad editovaným bludištěm je umístěno několik tlačítek, které slouží jako rychlé akce ovlivňující celé bludiště. Tato tlačítka jsou taktéž ovlivněna kontextem užití editace. V sekci *Laboratoř* a u startovního bludiště ve *Tvorbě cvičení* jsou to tlačítka pro vrácení poslední akce, úplného vyčištění bludiště a změny vzhledu pokládaných značek. Během tvorby finálního bludiště ve *Tvorbě cvičení* je tlačítko pro vyčištění nahrazeno zaškrtávacím polem značícím, zda má být kontrolováno celé bludiště, a namísto tlačítka pro změnu značek je přítomna nápověda pro kontrolování polí.

5.5 Cvičení

Sekce *Cvičení* umožňuje uživateli spustit některé z přichystaných cvičení, která jsou rozdělena do kategorií *Jednoduché*, *Těžší* a *Vlastní*. Poslední ze zmíněných kategorií slouží pro zobrazení seznamu vytvořených a importovaných cvičení ulo-

Obrázek 17: Sekce cvičení

žených v současné instalaci aplikace.

Uživatel si může zobrazit náhled cvičení v libovolné kategorii, který se skládá ze zadání, miniatury startovního bludiště a případné nápovědy, pokud ji cvičení obsahuje. Pokud se uživatel rozhodne spustit vybrané cvičení kliknutím na tlačítko pod jeho názvem, je přesunut do programovacího rozhraní. V něm jsou na počátku do knihovny umístěny všechny předdefinované procedury pro cvičení, které jsou pouze pro čtení (označeny symbolem oka) a jedna hlavní procedura (označena zelenou vlaječkou), kterou je potřeba doprogramovat. Při zvolení hlavní procedury ji není možné přejmenovat a není ani přítomno tlačítko smazání, místo toho se na jeho místě nachází tlačítko pro zobrazení zadání cvičení.

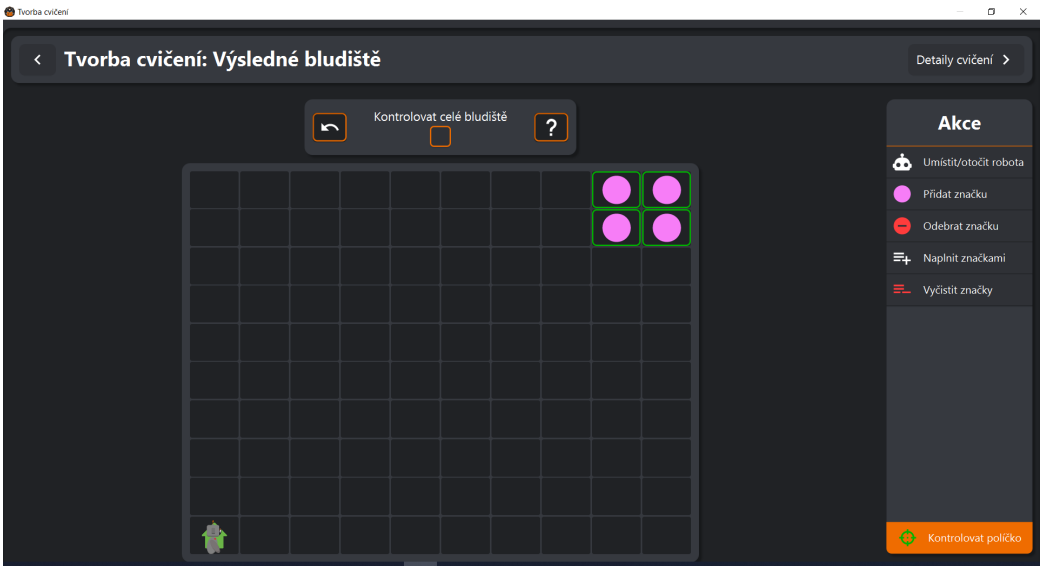
Jakmile je uživatel spokojen se svým řešením může kliknout na tlačítko *Zkontrolovat* v pravém horním rohu, což spustí hlavní proceduru a provede kontrolu výsledku. Pokud je výsledný stav bludiště shodný s tím, dosaženým je uživateli zobrazena hláška o úspěšném řešení, jinak je zobrazeno okno s vizualizovanými rozdíly oproti správnému řešení.

5.5.1 Tvorba cvičení

Aplikace umožňuje uživateli vytvořit své vlastní cvičení. Do *Tvorby cvičení* se uživatel dostane po kliknutí na položku *Vytvořit* v horním menu sekce *Cvičení*.

Tvorba se skládá z několika kroků. Nejdříve je zobrazena nápověda, jak má uživatel postupovat při tvorbě. Dále následuje tvorba počátečního bludiště cvičení. Po vytvoření bludiště je uživatel přesunut do programovacího rozhraní pro tvorbu procedur, kde může vytvořit všechny potřebné procedury pro účely cvičení a označit tu, která se má pro splnění cvičení doprogramovat. V předposledním

kroku je nutné vytvořit vzhled finálního bludiště odvozeného od toho počátečního, které bude sloužit pro kontrolu. Uživatel může zvolit jak samotná kontrola bude probíhat. Výchozí nastavení je kontrola celého bludiště, což lze změnit odškrtnutím políčka nad bludištěm a následným ručním výběrem kontrolovaných polí. Nakonec je uživatel přesunut na obrazovku detailu cvičení, kde doplní název, slovní zadání a případnou nápovědu. Formulář ve spodní části obsahuje i miniatury startovního a finálního bludiště pro vizuální porovnání. Po kliknutí na tlačítko *Dokončit* je cvičení uloženo do kategorie *Vlastní*, odkud může být ze svého náhledu exportováno do souboru nebo smazáno.



Obrázek 18: Tvorba výsledného bludiště s kontrolováním vybraných políček

5.6 Laboratoř

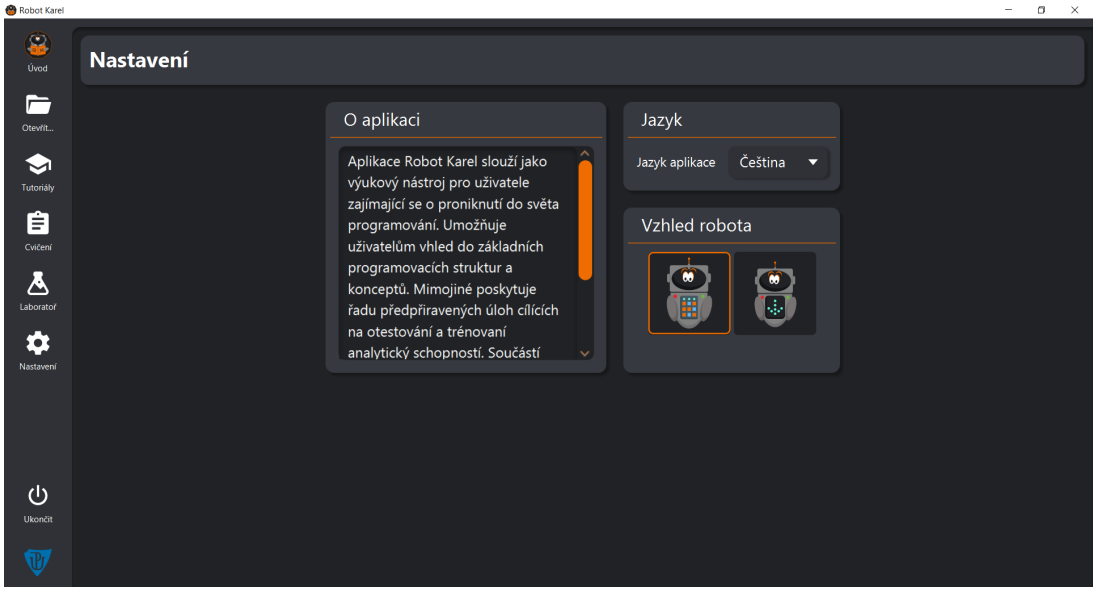
V sekci *Laboratoř* má uživatel plnou kontrolu nad všemi aspekty programovacího rozhraní. Může volně upravovat podobu bludiště, neomezeně tvořit, editovat a mazat procedury a využít hlasového ovládání pro pohyb v bludišti.

Pomocí vysouvacího menu v horní části obrazovky je možné vytvořit novou laboratoř, uložit tu stávající do souboru nebo načíst ze souboru již uloženou.

Jedná se o hlavní sekci aplikace, kde by měl uživatel trávit nejvíce času.

5.7 Nastavení

V nastavení uživatel může přepínat jazyk aplikace mezi češtinou a angličtinou. V případě přepnutí jazyka je kromě jazyka uživatelského rozhraní změněna i mutace jazyka Karel na anglickou verzi a zároveň dojde ke změně jazykového modelu pro rozpoznávání hlasových příkazů. K této změně je nutné aplikaci restartovat.



Obrázek 19: Nastavení

Dále je zde přítomný výběr vzhledu robota. Dvě možnosti výběru se liší způsobem, jakým je zobrazován aktuální směr robota Karla. V první případě je otáčen celý robot a ve druhém je měněna pouze šipka ukazující aktuální směr natočení robota.

Nakonec zde lze nalézt i krátký text nastiňující funkci a smysl vzniku aplikace.

Závěr

Byla vytvořena aplikace implementující vývojové prostředí pro programovací jazyk Karel, které využívá možnosti hlasového ovládání k zatraktivnění uživatelské interakce. Do aplikace byly vloženy funkcionality ke sdílení souborů se cvičeními a laboratořemi mezi uživateli, předdefinované cvičné úkoly a návody pro práci s vývojovým prostředím. Aplikace plně podporuje dva jazyky, a to češtinu a angličtinu. Jako součást této práce byla taktéž uvedena historie jazyka Karel včetně jeho různých implementací.

Možnost práce se softwarem Phonexia byla užitečnou zkušeností s rozpoznáváním a zpracováním lidského hlasu včetně poskytnutí náhledu do potenciálu, které takové programy naskýtají pro interakci se systémy vyvíjenými v dnešní době.

Conclusions

An application has been created that implements development environment for programming language Karel, which uses voice controls to make user interaction more atractive. Functionality has been added to the application for sharing exercise and laboratory files between users and predefined exercise tasks and tutorials on how to work with the development environment were included. Application fully supports both Czech and English languages. History and various implementations of language Karel were also included as a part of this work.

Possibility to work with Phonexia software has been a valuable experience in human voice recognition and analysis including the option to see the potential, which programs like these hold for the interaction with the systems developed today.

A Obsah přiloženého datového média

bin/
: Spustitelné soubory RobotKarel.exe, Karel_The_Robot-1.0.jar a startPhonexiaServer.cmd, spolu s adresářem resources obsahující složku /jre s verzí OpenJDK 11 nutné pro spuštění obou uvedených souborů. Dále pak složku /Phonexia obsahující nakonfigurovaný server od firmy Phonexia pro systém Windows zajišťující chod hlasového ovládání a soubor KarelDB, který obsahuje data potřebná pro běh aplikace.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty programu RobotKarel v .zip archivu.

README.txt
: Instrukce pro spuštění programu RobotKarel, včetně všech požadavků pro jeho bezproblémový provoz.

Navíc médium obsahuje:

assets/
: Adresář obsahující obrázky vytvořené pro účely aplikace.

licence/
: Smlouva uzavřená s firmou Phonexia pro užití jejich licencovaného softwaru pro účely této práce.

U veškerých cizích převzatých materiálů obsažených na médiu jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro všechny použité (a citované) materiály, u kterých toto není splněno a nejsou tak obsaženy na médiu, je uveden jejich zdroj (například webová adresa) v bibliografii nebo textu práce nebo v souboru `readme.txt`.

Literatura

[1] Piech, Chris; Roberts, Eric. *Karel the Robot Learns Python*. Stanford: Department of Computer Science, Stanford University, 2019.

[2] Pattis, Richard E. *Karel The Robot: A Gentle Introduction to the Art of Programming*. 2nd ed. 1995. 550 s. ISBN 978-0471597254.

[3] Roberts, Eric. *Karel the Robot Learns Java*. Stanford: Department of Computer Science, Stanford University, 2005.

[4] Bergin, Joseph; Stehlik, Mark; Roberts, Jim; Pattis, Richard E. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. 1st ed. 1996. 208 s. ISBN 978-0471138099.

[5] Mlej, Viktor. *KAREL 3D: Learning programming language for kids* [online]. 2018 [cit. 2022-4-15]. Dostupný z: ⟨https://sourceforge.net/projects/karel-3d/⟩.

[6] Jedlička, Oldřich. *Robot Karel* [online]. 2013 [cit. 2022-4-15]. Dostupný z: ⟨http://karel.oldium.net/⟩.

[7] Vejvoda, Michal; Rytíř, Miroslav. *Programovací jazyk Karel* [online]. 2001 [cit. 2022-4-15]. 72 s. Dostupný z: ⟨http://pckarel.sweb.cz/pdf/pjkarel.pdf⟩. ISBN 99972-03-09-7.

[8] Services, Tri Star CNC. KAREL: Understanding FANUC robot programming language. [online]. [Cit. 2022-4-15]. Dostupný z: ⟨https://www.tristarcnc.com/News/KarelProgrammingLanguage⟩.

[9] Piech, Chris; Roberts, Eric. *Karel the Robot Learns Java*. Stanford: Department of Computer Science, Stanford University, 2019.

[10] Akhin, Marat; Belyaev, Mikhail. *Kotlin language specification* [online]. 2020 [cit. 2022-7-10]. Dostupný z: ⟨https://kotlinlang.org/spec/introduction.html⟩.

[11] SQLite team. *About SQLite* [online]. [cit. 2022-7-10]. Dostupný z: ⟨https://www.sqlite.org/about.html⟩.

[12] Gradle team. *What is Gradle?* [online]. [cit. 2022-7-10]. Dostupný z: ⟨https://docs.gradle.org/current/userguide/what_is_gradle.html⟩.

[13] Steinberger, Peter. The new shiny: On the shift from imperative to declarative UI, and what it might mean for the apps we build today and tomorrow. *Increment* [online]. 2021, roč. 18, [cit. 2022-7-10]. Dostupný z: ⟨https://increment.com/mobile/the-shift-to-declarative-ui/⟩.

[14] TornadoFX team. *Why TornadoFX?* [online]. [cit. 2022-7-10]. Dostupný z: ⟨https://edvin.gitbooks.io/tornadofx-guide/content/part1/1_Why_TornadoFX.html⟩.

[15] TornadoFX team. *EventBus* [online]. [cit. 2022-7-10]. Dostupný z: ⟨https://edvin.gitbooks.io/tornadofx-guide/content/part2/EventBus.html⟩.

[16] TornadoFX team. *Type-Safe CSS* [online]. [cit. 2022-7-10]. Dostupný z: ⟨https://edvin.gitbooks.io/tornadofx-guide/content/part1/6_CSS.html⟩.

[17] PubNub team. *What is HTTP Streaming?* [online]. [cit. 2022-7-10]. Dostupný z: ⟨https://www.pubnub.com/learn/glossary/what-is-http-streaming/⟩.