VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

KAREL 3D – APLIKACE PRO VÝUKU PROGRAMOVÁNÍ

KAREL 3D – APPLICATION FOR TEACHING OF PROGRAMMING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VOJTĚCH ČOUPEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ZBYNĚK KŘIVKA, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Čoupek Vojtěch**
Program: Informační technologie
Název: **Karel 3D - aplikace pro výuku programování**
Karel 3D - Application for Teaching of Programming
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou výuky programování na základním a středním stupni škol a prozkoumejte aplikace, které s tímto pomáhají.
2. Dle konzultací s vedoucím navrhněte rozšiřitelnou aplikaci, která bude sloužit k výuce programování se zaměřením na moderní verzi jazyka/robota Karel známého z operačního systému MS DOS.
3. Dle návrhu aplikaci implementujte jako multiplatformní (např. webovou) a vícejazyčnou.
4. Proveďte srovnání s ostatními podobnými aplikacemi, získejte zpětnou vazbu od uživatelů a celý projekt zhodnoťte včetně návrhu na další rozvoj.

Literatura:

- PATTIS, Richard E. *Karel The Robot: A Gentle Introduction to the Art of Programming*. 2nd edition. Wiley, 1995. 160 s. ISBN 0-471-59725-2.
- KAREL - Stránka o robotu Karlovi, (c) 2003-2007. Dostupné on-line na http://karel.webz.cz/uvodni-strana [cit. 2020-10-12]
- BOHÁČ, Libor. Tvorba webového interpretu jazyka Karel [online]. Brno, 2011. Dostupné z: <https://is.muni.cz/th/j86kf/> [cit. 2020-09-21]. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Radek Pelánek.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a prototyp k bodu 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Křivka Zbyněk, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Práce se zabývá přehledem dostupných prostředků pro pedagogy informatiky, kteří seznamují studenty s programováním. Cílem je implementace prostředí pedagogického jazyka Karel inspirovaného prací Andreje Blaha a jeho kolegů pro platformu MS-DOS. Byla navržena aplikace ve webovém prostředí, která rozšiřuje funkčnost předlohy o proměnné a zahrnuje moderní prvky uživatelského rozhraní, jako jsou blokové programování, obarvování syntaxe jazyka, syntaktickou kontrolu a 3D grafiku. Současně také pedagogům nabízí sadu základních příkladů a dovoluje jim tvořit si vlastní zadání.

Abstract

The work provides a survey of the tools available for the teachers of Informatics who teach students the basics of programming. The main aim is to create the implementation of Karel pedagogic language inspired by the work of Andrej Blaho and his colleagues for MS-DOS platform. The new web application is introduced that expands the functionality of the original language with variables and includes the up-to-date elements of the user interface, such as block programming, highlighting the language syntax, syntax checking and 3D graphics. The application also provides a set of basic examples for teachers and allows them to create their own tasks.

Klíčová slova

Výuka programování, pedagogický programovací jazyk, jazyk Karel, blokové programování, webová aplikace, kontrola syntaxe, 3D grafika, střední škola, základní škola, začátečníci v programování

Keywords

Teaching of programming, pedagogical programming language, Karel language, block programming, web application, syntax checking, 3D graphics, secondary school, primary school, beginners in programming

Citace

ČOUPEK, Vojtěch. *Karel 3D – aplikace pro výuku programování.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Zbyněk Křivka, Ph.D.

Karel 3D – aplikace pro výuku programování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zbyňka Křivky Ph.D. Další informace mi poskytli Mgr. Petr Čoupek a Mgr. Roman Ondrůšek. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vojtěch Čoupek
9. května 2021

Poděkování

Chtěl bych moc poděkovat vedoucímu práce, Ing. Zbyňku Křivkovi Ph.D., za cenné rady a pomoc při tvorbě této bakalářské práce. Dále bych chtěl také poděkovat magistrovi Romanovi Ondrůškovi, který mi poskytl materiály pro tuto práci, a magistrovi Petru Čoupkovi za pomoc při realizaci projektu.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problematikou vyučování základů programování a programovacího myšlení na druhém stupni základních škol, gymnáziích a středních školách. Jelikož v dnešní době je většina běžně používaných programovacích jazyků pro úplného začátečníka příliš komplexní, složitá a neumožňuje přímou vizualizaci toho, co přesně se děje, existuje celá řada jednoduchých pedagogických nástrojů a jazyků, které tento problém pro úplné začátečníky řeší. Mezi takové prostředky patří i jazyk Karel, kterým se tato práce bude zabývat nejvíce. Existuje mnoho dalších prostředků, jako jsou Scratch, Baltazar a jiné, o kterých bude řeč v sekci 2.3. Na internetu lze nalézt i několik implementací jazyka Karel, ovšem nosnou verzí a inspirací této práce bude verze robota Karel na MS-DOS popsaná v kapitole 2.2. Práce si dává za úkol tuto verzi převést do moderního prostředí, zařídit její multiplatformnost, přidat moderní prvky, které se při novodobém vyučování programování využívají, a vytvořit tak nástroj, který by vyučování problematiky zjednodušil jak studentům, tak vyučujícím.

Motivací pro tuto práci je osobní zkušenost autora při prvním seznamování s úplnými základy programování, při kterých narazil právě na zmíněnou verzi Karla pro MS-DOS. Tato verze mě velmi zaujala, a díky ní jsem se více začal zajímat o informační technologie. Přestože je aplikace vysoce kvalitní, tak se na ní bohužel podepisuje čas. Není spustitelná na aktuálních operačních systémech bez emulace, nepodporuje ovládání myší a trpí dalšími omezeními, která vycházejí z limitů operačního systému MS-DOS. Tyto překážky úplným začátečníkům působí problémy. O toto znovuoživení jazyka jsem se už jednou pokusil v soutěži Středoškolské odborné činnosti, ve které práce sklidila kladné ohlasy, a proto navazuji touto prací, kde bych chtěl využít nabytých zkušeností a projekt dále přepracovat a rozšířit.

Touto prací čtenáře provede sedm kapitol. Po úvodu se ve druhé kapitole bude práce zaobírat historií jazyka Karel a jeho první verzí od pana profesora Richarda E. Pattise působícího na universitě v Kalifornii. Dále popíše již několikrát zmiňovanou verzi tohoto programovacího jazyka pro operační systém MS-DOS a následně jej v poslední části druhé kapitoly porovná se současnými výukovými programy, které se pro výuku používají. Dále práce představí v další kapitole vizi nového prostředí pro jazyk Karel. Navazuje čtvrtá kapitola, která popisuje návrh struktury aplikace, uživatelského rozhraní, a podobně. Samotná implementace je popsána v následující kapitole. Práci uzavírají kapitoly věnované testování a závěru.

Kapitola 2

Pedagogické jazyky a jazyk Karel

Tato kapitola je věnována problematice pedagogických jazyků a předvedení hlavních zástupců, které jsou používány v současné době na druhých stupních základních škol, středních školách a gymnáziích. Pedagogickými či didaktickými jazyky jsou myšleny jednoduché programovací jazyky, používané ve školách v hodinách informatiky, které mají za cíl seznámit začátečníky s problémem programování často zábavnou a hravou formou. Nejprve bude popsán jazyk Karel, a to jak jeho úplné začátky, kdy byl navržen, tak jeho následný vývoj ve verzi na MS-DOS, která je nosnou verzí této práce. V poslední části budou popsány současné prostředky využívané při vyučování, které se budou porovnávat s nosnou verzí této práce. Toto porovnání bude následně využito v další kapitole pro nastínění nové verze.

2.1 Začátky jazyka Karel

Programovací jazyk Karel se poprvé objevil v roce 1981, kdy jej představil ve své knize profesor Richard E. Pattis působící na Stanfordské univerzitě v Kalifornii: „Karel The Robot: A Gentle Introduction to the Art of Programming“[10]. Jazyk je velmi jednoduchý a dává si za úkol zasvětit studenty, kteří se s kódem ještě nesetkali, do základů programování a porozumění základních syntaktických konstrukcí. Pattis jej pojmenoval podle Karla Čapka, který obohatil svět slovem „robot“.

Jazyk je použit k programování robota nacházejícího se na šachovnicovém poli, které je ohraničeno zdmi. Toto pole je označeno jako město. Robot se po městě může pohybovat a pokládat omezený počet bzučáků ze svého batohu pomocí příkazů. Mezi tyto příkazy patří například: `MOVE` – přesuň se o políčko dopředu, `TURNLEFT` – otoč se vlevo o 90°, `PUTBEEPER` – polož bzučák, `PICKBEEPER` – zvedni bzučák, `TURNOFF` – vypni se. Dále se v programu může robot ptát na svůj aktuální stav a podle něj měnit průchod svým programem, k čemuž slouží podmínky. Mezi struktury, které robot umí, patří `IF <podmínka> THEN <blok příkazů` - provede se, když platí podmínka `> [ELSE <blok příkazů` - provede se když neplatí podmínka`>]`, podporuje cyklus s pevným počtem opakování `ITERATE <číslo> TIMES <blok příkazů>` a cyklus s ukončující podmínkou `WHILE <podmínka> DO <blok příkazů>`, kde je blok příkazů vykonáván, dokud podmínka platí. Celková stavba programu je velmi podobná konstrukci jazyka Pascal. Celý konstrukt uživatelem tvořeného programu je ohraničen klíčovými slovy `BEGINNING-OF-PROGRAM` a `END-OF-PROGRAM`, přičemž nejprve se očekává definice nových instrukcí pro robota ve formě `DEFINE <název instrukce> AS <blok příkazů>`, kde se blokem příkazů myslí buďto samotný jeden příkaz, nebo více příkazů ohraničených klíčovými slovy `BEGIN` a `END`. Po definici těchto instrukcí, které by se daly

připodobnit Pascalovským procedurám, následuje hlavní blok programu. Ten je vymezený pomocí klíčových slov `BEGINNING-OF-EXECUTION` a `END-OF-EXECUTION`, který je volán na začátku spuštění programu. Mezi dostupné podmínky patří například otázka, zda-li je před robotem volné políčko, nebo zeď `FRONT-IS-CLEAR` a `FRONT-IS-BLOCKED` (a směrové verze této pomínky `LEFT`, `RIGHT` a `BACK`), na orientaci v místnosti například `FACING-NORTH` případně `NOT-FACING-NORTH` a další možnosti směru, otázka kolik zbývá robotovi bzučáků v batohu `ANY-BEEPERS-IN-BEEPER-BAG` a nakonec zda-li políčko, na kterém se robot právě nachází, obsahuje bzučák `NEXT-TO-A-BEEPER`.

Pomocí těchto prostředků mohou studenti řešit různé úkoly a problémy, přičemž je vše graficky vizualizováno v ohraničeném městě. Jazyk sám o sobě neobsahuje žádné proměnné a jediným způsobem uložení hodnoty je položení bzučáku na některé z políček. Proměnné se měli studenti naučit až při přechodu na plnohodnotný programovací jazyk. Na internetu je dostupná celá řada projektů, které rekonstruují právě tuto Pattisovu verzi v několika různých světových jazycích. V nedávné době byla jazyku Karel také věnována magisterská práce Libora Boháče[2]. Mezi dostupnou českou mutaci této verze jazyku Karel bych zařadil také projekt Oldřicha Jedličky [6], které je vidět na obrázku 2.1.

Demonstrace

Pro lepší představu čtenáře bude v kapitolách, kde se rozebírá některý z dostupných jazyků, předvedena demonstrace daného řešení. Pro jazyk Karel ve 2D místnosti je vybráno vývojové prostředí vytvořené panem Oldřichem Jedličkou [6]. Sice se implementace mírně liší od původního Pattisova návrhu, na druhou stranu je jednoduše přístupná a má českou lokalizaci.

Vývojové prostředí je jednoduché a přehledné. Skládá se ze tří oken, přičemž horní okno nabízí seznam příkazů, které jsou pro robota momentálně definovány. Ty lze spouštět tlačítky po levé straně od názvu příkazu. Pod tímto oknem se nachází editační okno, kde může uživatel vytvářet své příkazy. Po pravé straně se nachází samotná vizualizace místnosti s robotem, možnosti přímé interakce s místností a nastavení aplikace. Robot do místnosti místo bzučáků pokládá značky a místo ve městě se pohybuje v místnosti. Uživatel může v místnosti definovat zdi, aby mohl fixní místnost o rozměrech 10∗10 políček upravit.

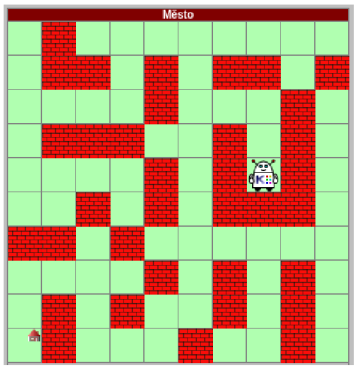
Demonstrován bude úkol, kdy robot bude umístěn náhodně do bludiště. V bludišti bude označena cílová pozice, která bude jiná než robotova aktuální pozice. Úkolem tedy bude robota dostat z jeho startovního bodu do cíle. Bludiště bude definováno v místnosti zdmi, tedy robot nebude moci vstoupit na jiné políčko, než je políčko bludiště. Žádná z chodeb v místnosti nesmí být kruhová a bludiště musí být řešitelné. Tento příklad je vybrán proto, že je řešitelný ve všech verzích jazyku Karel, o kterých bude řeč, a přitom je algoritmicky zajímavější než například `trojkrok`. Místnost pro demonstraci v aktuálním prostředí je vidět na obrázku 2.2 a prostředí, kde je příklad již vyřešený, je na obrázku 2.1.

Pro vyřešení tohoto příkladu je využit algoritmus, kde se bude robot držet pravé stěny a postupně prochází bludištěm, dokud nenarazí na svůj domov. Využijeme k tomu podmínku `JE ZEĎ`, která má hodnotu `PRAVDA`, pokud je před robotem zeď. Orientaci Karla lze poznat podle jeho natočení, přičemž obličej indikuje, kam se robot právě dívá. Jelikož je třeba jít po pravé ruce, algoritmus nejprve otočí robota vpravo, poté otáčí robota tak dlouho doleva, dokud platí podmínka `JE ZEĎ`, a poté udělá krok. To vše se opakuje, dokud nestojí robot na cílovém políčku. Jelikož v základu robot nezná příkaz `VPRAVO-VBOK`, je potřeba nejprve definovat ten, a poté lze naprogramovat již popsaný příkaz `PODÉL PRAVÉ ZDI`. Tyto příkazy jsou zapsány v jazyce Karel níže.

```
VPRAVO-VBOK
    VLEVO-VBOK
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
```

```
PODÉL PRAVÉ ZDI
    DOKUD NENÍ DOMOV
        VPRAVO-VBOK
        DOKUD JE ZEĎ
            VLEVO-VBOK
        KONEC
        KROK
    KONEC
KONEC
```

Obrázek 2.1: Vývojové prostředí pro jazyk Karel dostupné na http://karel.oldium.net/



Obrázek 2.2: Bludiště pro příklad - cílová pozice vlevo dole

2.2 Československá verze Karla pro MS-DOS

Na myšlenku profesora Pattise navázala skupina na Bratislavské universitě Komenského začátkem devadesátých let a vytvořila československou verzi jazyka Karel. Na tomto vývoji se podíleli Marian Vittek, Janka Chlebíková a Andrej Blaho. Nejprve pan Marian Vittek realizoval první verzi programu Karel 3D pro československý počítač PMD–85, který se používal ve školství na konci osmdesátých let. Ta je na obrázku 2.5. Nezměnila se však pouze lokalizace, ale byla upravena syntaxe a místnost, ve které robot operuje. Základní příkazy jsou téměř stejné, avšak přibyl příkaz `VPRAVO-VBOK`, který bylo potřeba do originálu doprogramovat. Nejznatelnější změnou prošlo interagování robota s místností.

Plocha není dvou-dimenzionální jako v originále, ale robot se nyní pohybuje ve tří-dimenzionálním prostoru. Proto je přidána možnost pokládání cihel, přičemž robot jich u sebe má neomezený počet. Jejich počet ovlivňuje pohyb robota následujícím způsobem:

- Každá cihla má výšku jedna
- Robot nemůže vystoupit výše než o jednu cihlu
- Robot může sestupovat libovolný počet cihle dolů, nevadí mu pády
- Robot nemůže položit více než deset cihel nad svoji aktuální výšku podlahy, na které stojí
- Robot může položit cihlu maximálně o jedna níže než je jeho aktuální výška podlahy, na které stojí
- Robot může zvednout pouze horní cihlu ze sloupečku cihel
- Robot může zvednout cihlu maximálně o devět cihel výše, než je jeho aktuální výška podlahy, na které stojí
- Robot může zvednou cihlu maximálně na stejné výšce, jako je jeho aktuální výška podlahy, na které stojí

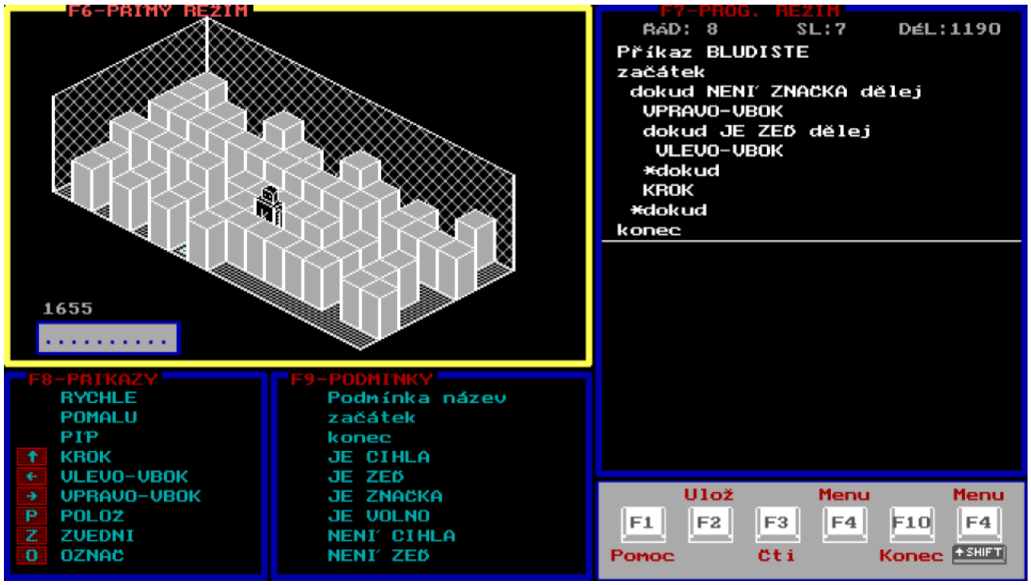
V programu je poté umožněno využít příkazy `POLOŽ` a `ZVEDNI` pro manipulaci s cihlami a podmínky říkající Karlovi, zda-li leží cihla `JE CIHLA` či neleží `NENÍ CIHLA` na políčku před robotem. Krom cihel je představen také objekt sloup, který má výšku pět (tedy pěti cihel), a nelze ho umístit při chodu programu, pouze v režimu přímého ovládání robota. Na tento objekt reaguje podmínka `JE/NENÍ ZEĎ`. Má tedy stejnou funkcionalitu jako zdi v předchozím demonstrovaném řešení. Poslední změnou prošly bzučáky, které jsou v této verzi předělány na značky. Opět jich u sebe má Karel nekonečně mnoho, ovšem na rozdíl od cihel je možno položit pouze jednu značku na jedno políčko. Pokud je na políčko se značkou položena cihla, značka se automaticky přesouvá na vyšší úroveň, tedy nezáleží na její výškové souřadnici a může být pouze jedna na políčku. Značka je plně dostupná při běhu programu pomocí instrukcí `OZNAČ` a `ODZNAČ` a vybavena sadou podmínek `JE/NENÍ ZNAČKA`, která je kontrolována na políčku, kde se Karel nachází.

S robotem je možné při přepnutí do manuálního ovládání v místnosti pohybovat pomocí klávesnice a místnost si editovat. Velikost místnosti je možno modifikovat v nastavení, ovšem součet rozměrů musí být méně než 25. Aplikace dovoluje místnosti ukládat a načítat.

Další změnou prošla syntaxe jazyka. Bylo odebráno obalení celého programu (klíčová slova `BEGINNING-OF-PROGRAM` a `END-OF-PROGRAM`, o kterých se hovoří v kapitole 2.1) a nově

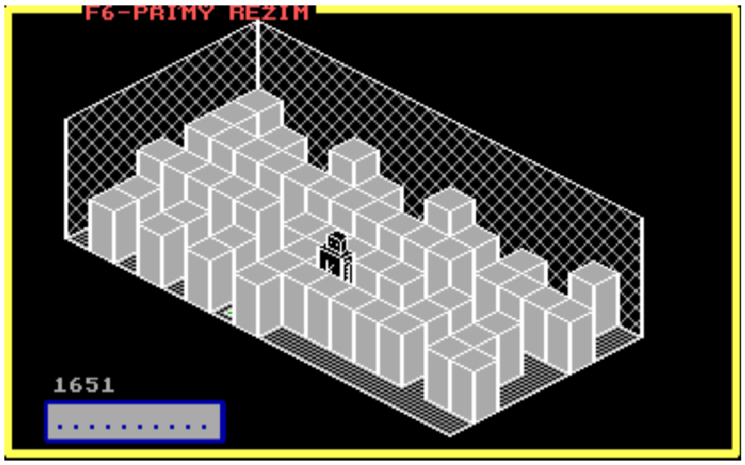
se pouze definují funkce, které poté uživatel specificky spouští (místo spouštění na bázi `main` funkce, jak tomu bylo v předchozí verzi). Tyto funkce jsou uvedeny konstrukcí `PŘÍKAZ <název příkazu> ZAČÁTEK <blok kódu> KONEC`. Byla přidána i možnost definovat podmínku, která se zapisuje pomocí konstrukce `PODMÍNKA <název podmínky> ZAČÁTEK <blok kódu> KONEC`. Dále bylo odstraněno uzavírání kódu do klíčových slov `BEGIN` a `END` (které zůstaly pouze implicitně v definicích příkazu a podmínky) a místo nich má každá syntaktická struktura ukončení pomocí spojení klíčového slova s hvězdičkou (například struktura `if` je v této verzi zapisována jako `KDYŽ <podmínka> TAK <blok kódu> [JINAK <blok kódu>] *KDYŽ`).

Tento projekt byl vytvořený pro operační systém MS-DOS, který podporuje ovládání pouze pomocí klávesnice, a lze jej na moderních zařízeních spustit jen v emulátorech, jako je například DOS-BOX[1]. Trpí i dalšími technickými omezeními, která systém udává, jako je například omezení velikosti souboru, kam se uživatelův kód ukládá. Stále se ale na některých školách aplikace využívá. Její podrobný popis je dostupný na stránkách Karla Klímy[7].

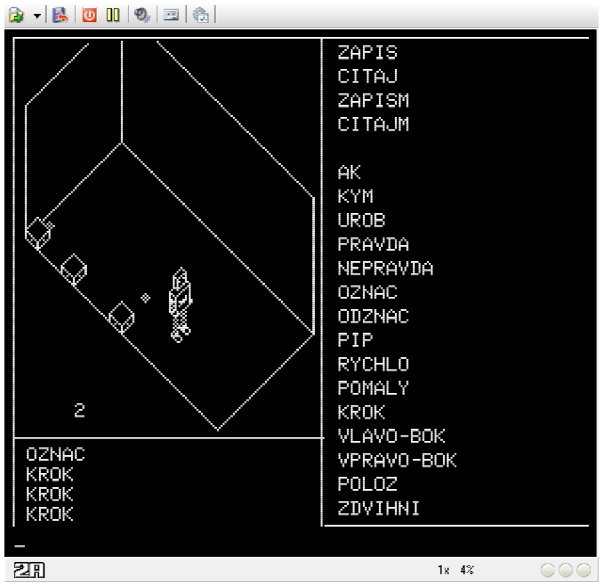


Obrázek 2.3: Uživatelské rozhraní československé verze robota Karel na MS-DOS

[1]Dostupné na https://www.dosbox.com/



Obrázek 2.4: Bludiště pro robota Karla na MS-DOS



Obrázek 2.5: Prvotní verze robota Karla 3D pro počítač PMD–85

Popis uživatelského prostředí

Uživatelské rozhraní prostředí pro jazyk Karel v operačním systému MS-DOS je patrné na obrázku 2.3. V levém horním rohu se nachází vizualizace robota v místnosti. S pohledem nelze nijak manipulovat, pouze přes toto okno lze pohybovat s robotem v přímém režimu, interagovat přes robota s místností a měnit rozměry místnosti. Také se v tomto okně v levém spodním rohu nachází počítadlo kroků a kolonka s názvem místnosti pro případné uložení.

Pod tímto oknem s vizualizací místnosti se nachází dvojice oken nabídky podmínek a příkazů. Pokud uživatel definuje nějakou svoji podmínku či příkaz, je do příslušného seznamu zařazen. U vestavěných příkazů jsou navíc zobrazeny klávesy, které daný příkaz provedou v přímém režimu ovládání robota. V režimu programování lze tato okna využívat

podobným způsobem jako v moderních editorech automatické doplňování, kdy aplikace porozumí tomu, že uživatel chce do kódu vložit příkaz a přesměruje jej do tohoto okna. Zde uživatel vybere příkaz, který chce právě použít.

Na pravé straně prostředí se nachází editační okno, kam uživatel zadává kód pro robota. Editor většinu syntaxe tvoří sám pomocí automatického doplňování. Například pokud chce uživatel vytvořit nový příkaz, vyjede kurzorem mimo poslední definovaný příkaz a editor automaticky vygeneruje potřebná syntaxe pro nový příkaz. Do kódu sám vkládá interaktivní slova „`příkazy`“, která poskytují již zmíněné automatické doplňování přes okna příkazů a podmínek. Sám se tak editor již při tvorbě kódu podílí na tom, aby výsledný kód obsahoval co nejméně chyb. Uživatel ale může psát kód sám bez doplňování. Pokud je před spuštěním nalezena chyba, aplikace nespustí interpretaci a text chyby zobrazí přes celé okno editoru. Nezobrazí však pozici chyby. Spouští se vždy příkaz, ve kterém se právě nachází kurzor, pomocí klávesy `F5`.

Pod editorem kódu se nachází kontextové menu, které mění své položky na základě toho, jaké okno má zrovna uživatel aktivované. Jak již bylo zmíněno, MS-DOS a aplikace Karel nepodporuje myš. Přepínání mezi okny a položky menu se proto vybírají pomocí funkčních kláves. Aktivované okno má vždy žluté ohraničení, zatímco neaktivní okna mají modré ohraničení.

Demonstrace

Opět je čtenáři předložen příklad programu pro toto prostředí podporující jazyk Karel. Úkolem je jako v předchozím případě dostat robota z jakéhokoliv políčka v místnosti na cílové políčko. Použit bude algoritmus, kdy se bude robot držet pravou ruku zdi jako v sekci 2.1. Bludiště je na obrázku 2.4. Cílová pozice je nyní určena značkou a nachází se uprostřed delší levé zdi. Program řešící tento problém zapsaný v jazyce Karel této verze vypadá takto:

```
Příkaz BLUDISTE
začátek
    dokud NENÍ ZNAČKA dělej
        VPRAVO-VBOK
        dokud JE ZEĎ dělej
            VLEVO-VBOK
        *dokud
        KROK
    *dokud
konec
```

2.3 Novodobé pedagogické jazyky

V této podkapitole budou vyjmenovány některé další nástroje, které se aktuálně používají pro výuku základů programování. Budou popsány a porovnány s MS-DOS verzí robota Karla. Nejvíce používaným moderním nástrojem tohoto typu je projekt Scratch.

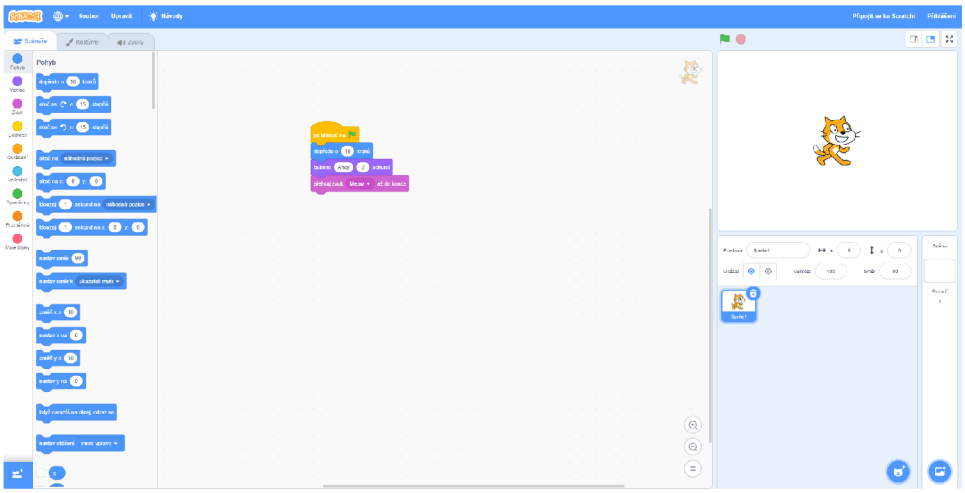
Scratch

Scratch[12] je freeware webová aplikace, sloužící úplným začátečníkům v oblasti programování. Základem aplikace je tvoření programů, které ovládají dvou-dimenzionální grafické rozhraní. Do tohoto rozhraní si žáci mohou nahrát různé postavičky, pozadí a další objekty, které mohou programy rozpohybovat, měnit jejich vlastnosti, spouštět nad nimi různé grafické efekty a podobně. Toto vše za podpory proměnných a možnosti běhu více procesů. V kódu lze nastavit i reakce na periferní zařízení. Uživatel tedy programuje svoji animaci, případně může vytvořit i jednoduchou hru. Scratch také podporuje spoustu světových jazyků včetně přeložených příkladů, nápadů a návodů pomáhajících uživatelům vniknout do světa programování.

Scratch, na rozdíl od Karla popsaného výše, nepoužívá k tvoření kódu textové podoby, ale podoby blokové. Programování pomocí bloků probíhá na pracovní ploše, kam uživatel bločky skládá. Tyto bloky zastupují stejnou funkcionalitu jako psaný kód, nicméně pro začátečníky jsou mnohem přívětivější. Bloky se ve výsledný kód spojují pomocí konektorů, které jsou reprezentovány jak vnitřně, tak graficky. Uživatel tedy na první pohled pozná, které spojení je možno udělat a které ne. Příklad bloků s různými konektory je vidět na obrázku 2.7. Tímto způsobem bloky hlídají syntax jazyka, neboť bloky nelze spojit do špatné struktury. Bloky jsou dostupné v bočním menu, kde jsou rozdělené podle funkčnosti do kategorií. Odsud je může uživatel na pracovní plochu přesunout pomocí funkce „drag-and-drop“. Tyto jednotlivé kategorie a bloky v ní mají přidělenou jednotnou unikátní barvu, která pomáhá rychle rozlišovat jednotlivé funkce bloků v kódu, podobně jako barevné zvýrazňování v moderních editorech kódu, jako jsou VS Code nebo Atom. Do menu kategorií lze přidávat rozšiřující kategorie dostupné v platformě, kterými je možno interagovat i s dalšími pedagogickými prostředími, jako například ekosystém Lego Mindstorms, Micro:bit a podobně.

Mezi nevýhody tohoto přístupu patří hlavně velká prostorová náročnost při vytváření složitějších konstrukcí. To se však u výukových jazyků nečeká. Pracovní plocha se pro uživatele při tvoření neustále rozšiřuje, a je tedy hlavně na uživateli, aby si na ní udržoval pořádek. Pracovní plochu je možné přibližovat, oddalovat a vrátit do výchozí pozice pomocí tlačítek dostupných v pravém dolním rohu. Bloky se z pracovní plochy mažou přetažením zpět do menu bloků. V pravém horním rohu pracovní plochy je průhledně zobrazen objekt, kterého se právě tvořený kód týká.

Další nástroje, které jsou poskytnuty, slouží k vytváření a upravování postaviček (případně je jich velké množství dostupné přímo v aplikaci), úpravě scény, ve které se postavičky pohybují, a úpravě zvuků, které mohou být při běhu programu spuštěny. Všechny tyto prvky jsou dostupné zdarma bez registrace uživatele. Po registraci se zpřístupní cloudové ukládání, management projektů a přístup k projektům ostatních uživatelů. Ani po registraci není potřeba žádných plateb. Projekty je možno zveřejnit a následně je mohou ostatní uživatelé spustit, prozkoumat, okomentovat a hodnotit podobně jako na sociální síti.



Obrázek 2.6: Prostředí Scratch

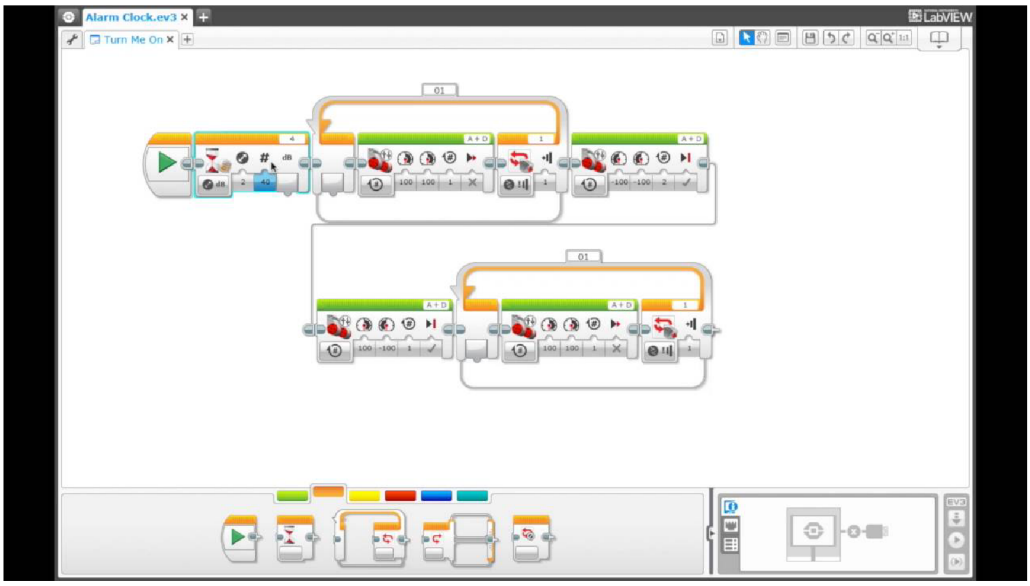
Obrázek 2.7: Příklad různých bloků v prostředí Scratch

LEGO Mindstorms

LEGO Mindstorms[2] je komerční platforma od dánské firmy LEGO. Uživatelům tato platforma nabízí možnost sestavení vlastního robota a programování jeho reakcí na okolní prostředí pomocí senzorů. Základní komponentou je speciální počítač firmy LEGO, ke kterému jsou dostupné rozšiřující moduly (senzory, motory apod.). Neučí studenty tedy pouze programování, ale před tímto krokem je také třeba robota navrhnout co nejlépe pro daný účel. Na stavbu robota jsou použity klasické komponenty, které LEGO používá i v ostatních stavebnicích. Co se týče vytváření programů, opět podobně jako Scratch používá blokového přístupu programování, kde uživatel na základě informací ze senzorů může nastavovat různé reakce výstupních zařízení, jako jsou motory, reproduktory, display apod. Počítač však není omezený pouze na tento typ programování, ale lze využít celou škálu jazyků, které jsou k tomuto účelu navrženy (většina je podobná běžně používaným programovacím jazykům počínaje C++ přes Javu, Python až třeba po Prolog).

Mezi klasické příklady pro takovou platformu je jízda po trase vyznačené čárou nebo náročnější řešení Rubikovy kostky pro zkušené uživatele. Výhodou této platformy je velké spektrum využití robota, problémem však je vcelku vysoká cena nejen základního kitu, ale i rozšiřujících komponent.

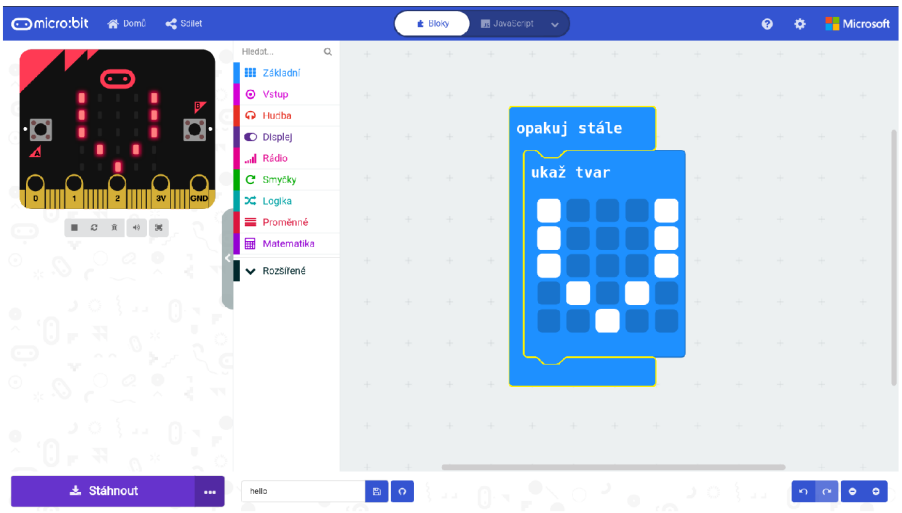
[2]K nahlédnutí na https://education.lego.com/en-us/

sp 

Obrázek 2.8: Prostředí Lego EV3 - obrázek převzatý z https://www.astramodel.cz/content/descriptions/lego/LEGO31313/LEGO31313_3.jpg

Micro:bit

Micro:bit je vývojová deska, která spolupracuje s prostředím MakeCode[3]. V prostředí je deska simulována a pomocí blokového editoru kódu, a je možné využívat komponenty, které deska obsahuje. Deska disponuje maticovým displayem o rozměrech pět × pět, dvěma tlačítky, rozhraním pro připojení dalších periferií, reproduktorem a dalšími prvky. Systém tedy umožňuje vytvořit program a nahrát jej do desky. Typickými úkoly pro toto prostředí je vytváření jednoduchých zařízení typu „wearables“ nebo základních robotů.

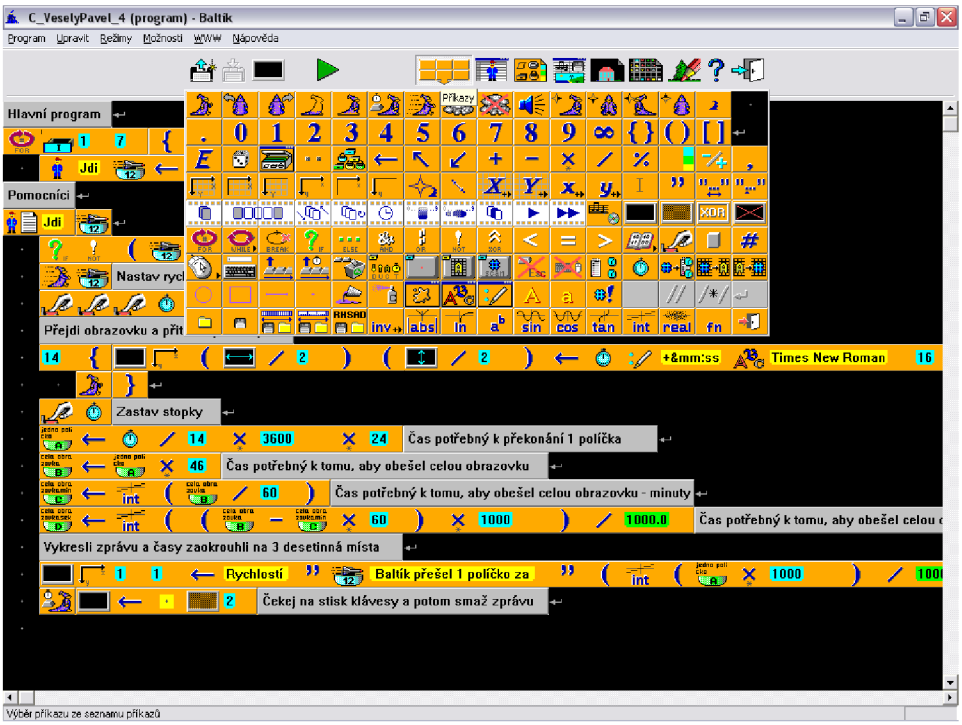


Obrázek 2.9: Prostředí MakeCode se simulací vývojové desky Micro:bit

[3]K nahlédnutí na https://makecode.microbit.org/#editor

Baltík

Baltík[13] je české komerční prostředí pro výuku programování, které vzniklo v devadesátých letech minulého století. Je založený na jazyce C. Kód se ovšem nezapisuje textem, ale pomocí ikon. Jedná se o mezistupeň mezi textovým a blokovým programováním. Prostředí umožňuje uživateli naprogramovat pohyb a interakci čaroděje po místnosti (podle verze je dostupná buď 2D nebo 3D místnost). V této místnosti může čaroděj vyčarovávat různé objekty a zaplňovat tak vyhrazený prostor. Prostředí je od doby vytvoření neustále aktualizováno a v dnešní době již existuje i jeho verze pro webové prostředí. Není však dostupné komukoliv, je ze třeba získat licenci.



Obrázek 2.10: Vývojové prostředí Baltík - obrázek převzat z https://cs.wikipedia.org/wiki/Balt%C3%ADk#/media/Soubor:Baltik3-screenshot1.PNG

Kapitola 3

Vize nové verze jazyka Karel

Pří svém studiu na střední škole jsem byl seznámen s verzí robota Karla pro MS-DOS. Už od té doby se mi koncept líbil, avšak často jsme při studiu naráželi na problémy spojované s jeho stářím, a tak jsme tak vyhledávali jiné možnosti, které byly dostupné. Většina dostupných programů však pracuje na zcela jiné bázi (většinou ve 2D bez místnosti, případně verze Karla jsou dostupné pouze ve 2D, na bázi Pattisovy verze), což znemožnilo použití již hotových zadání, která byla naší školou poskytnuta. V průběhu studia se k nám dostal studentský projekt, který měl za úkol předělat Karla do moderního prostředí, ovšem projekt nebyl moc kvalitní, díky čemuž zanikl a není již bohužel dostupný. Sám jsem se při studiu pokusil o vlastní verzi s webovým rozhraním[1]. Projekt jako takový byl úspěšný, ovšem limitem byly mé tehdejší programátorské znalosti a schopnosti.

Cílem této práce je navrhnout a implementovat novou verzi jazyka Karel. Jak již bylo řečeno, hlavní předlohou bude verze na MS-DOS popsaná v kapitole 2.2. Tato kapitola bude věnována vizi, jaká by aplikace měla být, aby mohla být nasazena do praxe výuky programování.

Možné cíle jsou následující:

- Věrnost originálu – Jelikož se jedná o novou verzi, měla by být kompatibilní s originální verzí, tedy syntaxe jazyka by měla zůstat co nejpodobnější. Důvodem je využití již existující sady příkladů a řešení, které mají učitelé středních škol pro Karla připravené.
- Místnost s robotem – Jedním z požadavků bude, aby program manipuloval s robotem ve 3D místnosti, jejíž rozměry budou uživatelem nastavitelné a bude možno v ní robota ovládat přímo před spuštěním programu jak pomocí klávesnice, tak na dotykových zařízeních.
- Odstranění omezení – Aplikace by měla být uživatelsky přístupná a neměla by omezovat jako zmíněná verze na MS-DOS, především by pak měla podporovat myš a případně i dotyková zařízení, která teoretický běžný uživatel budoucí aplikace využívá hojně, a neměla by bránit v tvořivosti, čili neměla by omezovat velikostí úložných souborů. Prostředí by mohlo nabízet jak ukládání lokální, tak možnost uložit soubor do cloudu, aby jej uživatel měl přístupný z více zařízení bez nutnosti složitého přenášení souborů.
- Moderní prvky – Aplikace by měla využít moderních prvků, které jsou přítomné jak v popsaných aplikacích, tak v běžných nástrojích, které dnes vývojáři používají.

[1] Výsledek je dostupný na http://smallm.cz/karel/

Mezi takovéto prvky patří hlavně blokové programování přítomné v drtivé většině moderních pedagogických jazyků. Neměl by to však být jediný způsob tvorby kódu, mělo by zůstat zachováno i textové rozhraní, které bude rozšířeno o obarvování syntaxe a zobrazování syntaktických chyb. Mezi těmito dvěma rozhraními by mělo jít přepínat bez nutnosti opětovného načítání aplikace. V textové editaci by pak měl být zajištěn debugging a krokování kódu.

- Překlad do více světových jazyků – Nová verze by také měla podporovat více světových jazyků (myšleno mluvených světových jazyků, ne programovacích) a mělo by být umožněno jejich plynulé přepínání za běhu aplikace. Ze začátku bude hlavním cílem implementovat českou a anglickou lokalizaci. Pro tento účel by mohl být vytvořen jednoduchý nástroj, který by umožňoval uživatelům vytvářet pro aplikaci své vlastní jazykové balíčky a použít je v aplikaci.

- Rozšíření schopností – Pokud zvážíme syntaxi, bylo již řečeno, že by měla být blízká originálu. To zajistí, že příklady, které již pro Karla existují, budou moci být opět použity v této nové verzi. Přesto by bylo dobré pokusit se jazyk rozšířit. Hlavním rozšířením, které by tedy mohlo být implementováno, jsou matematické výrazy a proměnné. U tohoto rozšíření je ale třeba dbát na vizualizaci vnitřních stavů, tedy hodnoty proměnných by měly být po celou dobu interpretace vhodným způsobem vizualizovány uživateli. Další rozšíření, které se týká syntaxe jazyka, by mohlo být mutace do zápisů, které budou podobné používaným jazykům v praxi (jako Python, C apod.). Rozšíření ovšem nesmí měnit jádro syntaxe nativního jazyka kvůli již zmíněné zpětné kompatibilitě se starými příklady pro jazyk Karel.

- Průvodce začátky – Na místě je i integrace průvodce jazykem a představení nejen hlavních prvků jazyka, ale i základních programovacích principů v sérii návodů a příkladů. Toto prostředí by mělo uživateli představit problém, určit mu prostředky, které by měl k jeho řešení použít, a poskytnout k těmto příkladům také možná řešení. Funkce by měla být dostupná i pro pedagogy, aby mohli vytvářet a editovat vlastní příklady a rozšiřovat tak materiál pro svoje žáky.

Tyto vize jsou předlohou pro návrhovou a implementační část aplikace a je možné, že ne všechny budou naplněny, ovšem výsledná aplikace by se jim měla co nevíce blížit. Jednotlivé body seznamu jsou uvedeny od nejdůležitějších po méně důležité a i průběh návrhu a implementace aplikace se jím bude v tomto pořadí řídit.

Kapitola 4

Návrh řešení

Tato kapitola se bude zaobírat návrhem vlastní aplikace. Nejprve bude popsáno, jaké technologie byly pro realizaci použity a z jakého důvodu. Následně budou představeny knihovny pro danou technologii, které jsou v projektu vyžity. Bude také popsán návrh vnitřní stavby aplikace, uživatelského rozhraní a systému kolem aplikace.

4.1 Výběr technologií

Prvním krokem pro tvorbu projektu je vybrat platformu, na kterou bude aplikace vytvářena. Jelikož je v požadavcích, aby aplikace podporovala co nejvíce zařízení bez emulace a dnes již běžné periferní zařízení jako jsou myši a dotykové displaye, bylo zvoleno prostředí webu. Je to prostředí, na které jsem se již pokoušel Karla programovat. Pro webové rozhraní jsou také dostupné knihovny, které budou oporou pro aplikaci. Další výhodou tohoto způsobu řešení je, že koncový uživatel nebude muset instalovat žádný software přímo na svém počítači a bude mu stačit pouze připojení k internetu a nainstalovaný webový prohlížeč, avšak bude možné aplikaci spouštět i lokálně po stažení. Hlavními nástroji pro tvorbu funkční části tedy bude jazyk JavaScript doplněný o HTML a CSS, jejichž úkolem bude vizualizace uživatelského rozhraní.

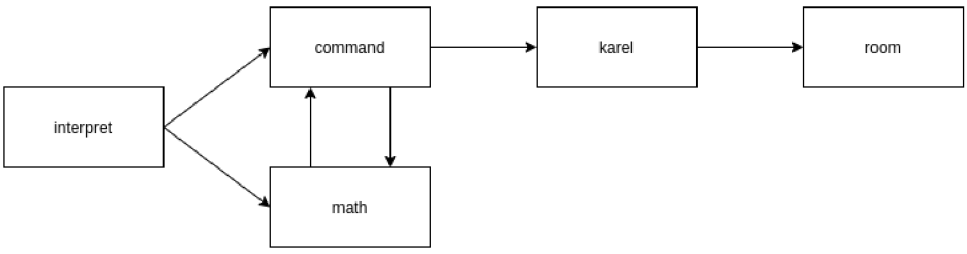
Tento způsob řešení také nabízí vytvoření jednoduchého informačního systému okolo aplikace. Ten by uživatelům mohl poskytoval soubor příkladů a jejich řešení, případně po přihlášení také ukládání aktuálního stavu aplikace na serverové úložiště a načítání těchto pozic. Tento koncept může být v budoucnu dále rozšířen. Pro realizaci této části bude potřeba zařadit jazyk PHP a databázi MySQL (případně pro interní potřeby databázi SQLite).

4.2 Návrh funkční části

Aplikační jádro je implementováno v JavaScriptu, konkrétně v jeho verzi ECMAScript 6[15], která je již moderními prohlížeči podporována[16]. JavaScript je objektově orientovaný skriptovací jazyk, proto je jádro aplikace rozděleno do koncepčních celků, které jsou zachyceny v jednotlivých třídách. V následujícím přehledu budou třídy popisovány podle referencí od zdola nahoru.

- `room` – Leží v úplném základu aplikace. Uchovává informace o aktuálním stavu místnosti a poskytuje základní operace s místností. Neobsahuje informace o pozici robota.
- `karel` – Ukládá informace o pozici a orientaci robota v místnosti, vkládá grafickou reprezentaci postavičky robota do místnosti. Dále poskytuje základní příkazy, které jsou v jazyku Karel dostupné. Uchovává referenci na instanci třídy `room`.
- `command` – Úkolem této třídy je ukládat vnitřní reprezentaci uživatelem definovaných příkazů a podmínek. Dále poskytuje kompletní sadu pro realizaci základních příkazů a podmínek jazyka Karel. Ukládá referenci na instance tříd `karel` a `math`.
- `math` – Stará se o rozšíření jazyka Karel, které implementuje matematické výrazy a proměnné. Tedy ukládá uživatelem definované proměnné a poskytuje nástroje k vypočítání matematického výrazu. Ukládá referenci na instanci třídy `command`.
- `interpret` – Realizuje kontrolu a překlad jazyka Karel a následnou interpretaci daného kódu. Ukládá reference na instance tříd `command` a `math`.

Podrobněji budou popsány funkce jednotlivých tříd v sekci 5.1 a jejich schéma zakresleno v diagramu na obrázku 4.1. Instance třídy `interpret` je vytvářena v pomocném `main.js` skriptu, který propojuje toto jádro aplikace s uživatelským rozhraním, nastavuje jazyk aplikace a podobně.



Obrázek 4.1: Diagram tříd aplikačního jádra

4.3 Použité knihovny

Tato sekce se bude zabývat knihovnami pro jazyk JavaScript, které byly při tvorbě projektu použity. Nejprve bude řeč o knihovně ACE (Advanced Code Editor), která tvoří textový editor pro vytváření programů pro Karla. Dále mezi ně patří knihovna Blockly, zajišťující blokové programování, a knihovna THREE.js, jejímž úkolem je vykreslování místnosti s robotem. Poslední dvě knihovny, Split.js a jQuerry s rozšířením jQuerry UI jsou v projektu využity hlavně pro realizaci uživatelského prostředí.

THREE.js

THREE.js[4] se stará v aplikaci o vykreslování místnosti s Karlem. Jedná se o knihovnu pracující nad WebGL a implementující 3D grafické rozhraní. Knihovna definuje plátno, na které se grafika vykresluje. Na toto plátno je aplikováno ovládání, které je také dostupné skrze jmenovanou knihovnu, což umožňuje místnost natáčet pomocí myši a zakliknutého levého tlačítka, přibližovat a oddalovat kolečkem myši a zakliknutím pravého tlačítka a pohybem myši hýbat s celou místností. Knihovna umožňuje i načítání vlastních grafických objektů. Jedním z takto načítaných objektů je postavička robota Karla, která byla vytvořena v aplikaci Blender.

Knihovna ACE

ACE, neboli Advanced Code Editor[1], je knihovna navržená pro editování kódů ve webovém prostředí. Je využívána celou řadou projektů, jako je Overleaf (pro editaci Latex dokumentů on-line), Wikipedia, AWS a další. Zajišťuje celou řadu funkcí, které se dnes běžně vyskytují v editorech kódů, jako jsou Visual Studio Code, Atom a další. Mezi tyto funkce patří obarvování syntaxe kódu, zabalování vnořených úseků kódu, volbu breakpointů pro debuging, zobrazování chyb syntaxe, vracení se zpět v úpravách kódu, vyhledávání a další. Aby knihovna pracovala s jazykem aplikace správně, je pro něj nutné vytvořit konfiguraci. V případě podpory více jazyků je třeba, aby konfigurace byla vytvořená tak, aby bylo možné měnit klíčová slova pomocí jazykových balíčků, které jsou k aplikaci přiloženy. Díky tomu se mohou měnit klíčová slova, ovšem je zachován jejich syntakticky význam, obarvení a zalamování kódu.

Knihovna Blockly

Knihovna Blockly[5] je v aplikaci využívána pro blokové programování. Blockly je používaným rozšířením pro velkou část jmenovaných pedagogických aplikací, jako je Scratch, MakeCode a spousta dalších. Knihovna definuje pracovní plochu, kterou poté může uživatel systémem „drag-and-drop“ zaplnit bloky. Obsah plochy je poté možné automaticky přegenerovat na textovou podobu daného jazyka. Bloky, které jsou pro danou aplikaci potřeba, je možné vytvořit v prostředí poskytnutém vývojáři této knihovny společně s generátorem textové podoby kódu[1]. Dále knihovna poskytuje nástroje k vytváření seznamu dostupných prvků a obsahuje svoje jazykové balíčky. Ty jsou v aplikaci poupraveny, aby fungovaly ve spojení s jazyky aplikace.

Split.js

Split.js[3] umožňuje pomocí JavaScript v aplikaci vytvořit rozdělená okna, kterým může uživatel měnit rozměry ve vertikálním nebo horizontálním směru pomocí tažení hran těchto oken. Do těchto oken jsou pak vloženy další prvky aplikace, jako například editory nebo vizualizace místnosti. Díky knihovně je tak vytvořeno jednoduché modulární prostředí, které uživatelům umožňuje měnit velikost nebo celkově skrývat různé prvky aplikace.

[1] Tento nástroj je dostupný na https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

jQuery a jQuery UI

JQuery[11] je knihovna pro Javascript, která se specializuje na interakci JavaScriptu a HTML obsahu stránky. V aplikaci je použito i její rozšíření jQuery UI, které poskytuje prvky pro uživatelské rozhraní. Konkrétně v projektu zprostředkovává dialogová okna, která se vyvolají například pokud uživatel chce změnit velikost místnosti, uložit pozici a podobně. Obsah těchto oken je definovaný v HTML souboru stylem definovaným pomocí CSS.

4.4 Návrh jazyka Karel

Před samotnou implementací je třeba určit, jaká syntaxe jazyka Karel bude v aplikaci podporována. Jak je již vytyčeno ve vizích projektu, bude snaha o to zachovat co nejbližší podobu verzi prostředí Karel na MS-DOS. Tato verze syntaxe ovšem obsahuje v některých strukturách přebytečná klíčová slova, která jsou v nové verzi odstraněna. V MS-DOS prostředí to není takový problém, jelikož obsahuje zmíněné automatické doplňování, které je popsané v sekci 2.2. V novém prostředí je také možné zakomponovat automatické napovídání a doplňování syntaxe, ovšem odlehčení syntaxe prospěje i kontrole, interpretaci a vytváření kódu.

První změnou jazyka je přidání komentářů do kódu jazyka. Dostupné budou jednořádkové komentáře uvozené znakem `#`. Obsah komentáře bude obarvován do tmavě zelené barvy jak je zavedeno v ostatních editorech programovacích jazyků. Komentáře budou při interpretaci ignorovány a uživatelé si do nich budou tak moci psát libovolné poznámky.

Dále byla z klíčových slov v českém jazyce odstraněna diakritika z důvodu, že ani pokročilé programovací jazyky neobsahují diakritiku, a je tak snaha na to uživatele navyknout. Bude možné ji ve speciálním jazykovém balíčku zavést, ovšem nebude součástí implementovaného základního českého balíčku. Dále bude společně s českou verzí jazyka Karel představen i návrh anglické jazykové mutace.

Pro jazyk jsou nejdůležitější příkazy, pomocí kterých robot interaguje s místností. V následujícím seznamu jsou popsány základní příkazy, které jsou v jazyce dostupné. Jednotlivé položky obsahují nejprve české klíčové slovo, následně anglické klíčové slovo a popis robotovy reakce na slovo.

- `krok` – `forward`: Robot se posune o jedno políčko ve směru, kam je otočený, pokud je to možné. Není možné provést krok mimo místnost nebo o více než jednu cihlu nahoru.
- `vpravo` – `right`: Robot se otočí o devadesát stupňů vpravo.
- `vlevo` – `left`: Robot se otočí o devadesát stupňů vlevo.
- `poloz` – `place`: Robot před sebe položí cihlu, pokud je to možné. Nelze pokládat cihly mimo místnost nebo na pole, kde je vrchní cihla políčka o devět a více výše nebo o dva a více níže, než je Karlova aktuální pozice.
- `zvedni` – `pick`: Robot zvedne cihlu z pole před sebou, pokud je to možné. Není možné zvedat z pole mimo místnost nebo z polí, kde je vrchní cihla o deset a více výše než je aktuální Karlova pozice nebo níže než než je Karlova aktuální pozice.
- `oznac` – `mark`: Označí aktuální pole, na kterém robot stojí. Pokud již označeno je, nic se neděje.

- `odznac` – `unmark`: Odebere značku z aktuálního pole, na kterém robot stojí. Pokud označeno není, nic se neděje.
- `pravda` – `true`: Nastaví aktuální výsledek podmínky na hodnotu pravda. Tento příkaz bude mít efekt pouze v podmínkách.
- `nepravda` – `false`: Nastaví aktuální výsledek podmínky na hodnotu nepravda. Tento příkaz bude mít efekt pouze v podmínkách.
- `rychle` – `fast`: Zvýší rychlost vykonávání instrukcí robotem.
- `pomalu` – `slow`: Sníží rychlost vykonávání instrukcí robotem.
- `pip` – `beep`: Robot přehraje výstražný zvukový signál.

Mimo příkazy jazyk obsahuje i podmínky, které je možné využít v rozhodujících strukturách a cyklech. Podmínka je vždy v jazyce uvedena s prefixem `je` a nebo `neni` pro určení negace. V anglické podobě jsou to slova `is` a `not`. Následující seznam obsahuje základní podmínky, které jazyk poskytuje. Opět je přítomno české klíčové slovo, anglické klíčové slovo a popis reakce robota.

- `cihla` – `brick`: Pokud je na poli před robotem alespoň jedna cihla, vrací pravdu, pokud na poli před robotem není cihla, nebo před robotem není pole z místnosti, vrací nepravdu.
- `zed` – `wall`: Pokud před robotem není pole místnosti, vrací pravdu, jinak vrací nepravdu.
- `znacka` – `mark`: Pokud je na robotově aktuálním poli značka, vrací pravdu, jinak vrací nepravdu.
- `volno` – `vacant`: Pokud může robot udělat krok vpřed, vrací pravdu, jinak vrací nepravdu.

Příkazy lze vkládat do cyklů a rozhodujících struktur. První z dostupných cyklů je struktura `opakuj`, která opakuje vnořené příkazy zadaný počet opakování. Cyklus má následující způsoby zápisu.

```
# česká verze zápisu cyklu
udelej [číslo] krat
    [blok příkazů]
*udelej
```

```
# english version of cycle
do [number] times
    [block of code]
*do
```

Méně triviálním cyklem využívajícím podmínky je struktura `dokud`. Ta přijímá podmínku, pokud ta vrací pravdu, provádí se vnořená posloupnost příkazů. Pokud podmínka vychází jako nepravda, přeskakuje se a interpretace pokračuje dále ve vykonávání následujících příkazů. Zápis tohoto cyklu vypadá následovně:

```
# česká verze zápisu cyklu
dokud je/neni [podmínka]
    [blok příkazů]
*dokud
```

```
# english version of cycle
while is/not [condition]
    [block of code]
*while
```

Následuje struktura `kdyz`, která provede zanořený blok příkazů, pokud se přiložená podmínka pravdivá, případně pokud je přítomna nepovinná větev `jinak` a je podmínka nepravdivá, provede se blok příkazů zanořený v této větvi.

```
# česká verze zápisu struktury
kdyz je/neni [podmínka]
tak
    [blok příkazů]
*kdyz

# s~nepovinnou větví jinak
kdyz je/neni [podminka]
tak
    [blok příkazů]
jinak
    [blok příkazů]
*kdyz
```

```
# english version of cycle
if is/not [condition]
then
    [block of code]
*if

# with else branch
if is/not [condition]
then
    [block of code]
else
    [block of code]
*if
```

Jazyk uživateli dále umožňuje vytvářet vlastní příkazy a podmínky a používat je v dalších příkazech či podmínkách jako podprogramy. Pro definici nového příkazu nebo podmínky je třeba použít jednu z následujících struktur:

```
# česká verze zápisu příkazu
prikaz [název]
    [blok kódu]
konec

# česká verze zápisu podmínky
podminka [název]
    [blok kódu]
konec
```

```
# english definition of command
command [name]
    [blok of code]
end

# english definition of condition
condition [name]
    [block of code]
end
```

Za název příkazu lze dosadit libovolný alfanumerický řetězec, který neobsahuje mezeru a neshoduje se s žádným jiným klíčovým slovem jazyka, nebo s jiným již definovaným názvem příkazu či podmínky. Pod tímto názvem jej pak lze volat v ostatních příkazech či podmínkách. Příkazy nemají žádné parametry ani návratové hodnoty. Podmínky také nemají parametry, ovšem očekává se, že v jejich definici bude určení návratových hodnot pomocí příkazů pravda a nepravda. Jazyk podporuje rekurzi.

4.5 Návrh rozšíření jazyka

Největší slabinou jazyka je, že nemůže mezi příkazy předávat žádná data a robot si není schopný zapamatovávat informace. Z tohoto důvodu je navrženo rozšíření obohacující jazyk o matematické výrazy a proměnné. Toto rozšíření umožňuje místo čísel a podmínek do kódu zadávat matematické výrazy, které buď určí počet opakování cyklu, nebo průchod programem (v případě struktury `kdyz`). Navrhované výrazy podporují celá nezáporná čísla, nad kterými lze použít následující operátory:

- `+` – součet
- `-` – odčítání
- `*` – násobení
- `/` – celočíselné dělení
- `%` – zbytek po celočíselném dělení
- `>` – větší než
- `>=` – větší než nebo rovno
- `<` – menší než
- `<=` – menší než nebo rovno
- `==` – rovno
- `!=` – není rovno
- `()` – závorky

Operátory porovnání jsou implementovány tím způsobem, že se rovnají hodnotě `1`, pokud platí, jinak se rovnají hodnotě `0`. Pokud se do cyklu `udelej [N] krat`, který opakuje zanořený blok kódu N-krát, dosadí záporné číslo, cyklus se bude chovat stejným způsobem, jako by mu byla dána hodnota `0`, tedy přeskočí se jeho obsah a neprovede se ani jednou. V případě, že matematický výraz nahrazuje podmínku, jeho výstup je definován jako pravda, pokud se výsledná hodnota různá od nuly, a definován jako nepravda, pokud se výsledná hodnota výrazu rovná nule.

Následně toto rozšíření obohacuje jazyk o globální a lokální proměnné, do kterých je možno ukládat celá čísla. Proměnné je nejprve třeba definovat. Globální proměnné se definují mimo definiční strukturu příkazu či podmínky pomocí následující syntaxe:

```
# česká deklarace globální proměnné
globalni [promenna] [název proměnné] = matematický výraz

# english declaration of global variable
global [variable] [identifier] = expression
```

Identifikátor definované globální proměnné se nesmí shodovat s žádným z klíčových slov jazyka, ani s názvem z uživatelem definovaných příkazů či podmínek. Po definici globální proměnné je možné stejný syntaktický konstrukt použít v příkazu nebo podmínce k přiřazení nové hodnoty proměnné a používat ji v matematických výrazech, kde se k vyhodnocení

výrazu použije její aktuální hodnota. Tyto globální proměnné lze použít k předávání parametrů mezi příkazy a podmínkami. Klíčové slovo `promenna` (nebo v angličtině `variable`) je možné vypustit, zabraňuje ovšem vytváření proměnných pod jménem `promenna`, což má pedagogický důvod.

Dále jsou dostupné i lokální proměnné, jejichž působnost platí pouze pro příkaz, v němž jsou definovány, a pro konkrétní běh příkazu. Vytvořit lokální proměnnou lze pomocí následující syntaktické konstrukce, která je zapsána v těle příkazu či podmínky, kde má být použita a dostupná.

```
# česká deklarace globální proměnné
lokalni promenna [název proměnné] = [matematický výraz]

# english declaration of global variable
local variable [identifier] = [expression]
```

Identifikátor definované lokální proměnné se opět nesmí shodovat s žádným z klíčových slov jazyka, názvem příkazu nebo proměnné, a ani identifikátorem žádné globální proměnné. Do lokální proměnné se přiřazuje stejným zápisem. Pokud je příkaz rekurzivní, při každém novém spuštění příkazu se vytvoří nové zanoření a s tím i nové rozsah působnosti proměnných, tedy pro nové spuštění se v paměti vytváří nová proměnná, která překrývá případnou starší stejnojmennou proměnnou z rekurzivního volání.

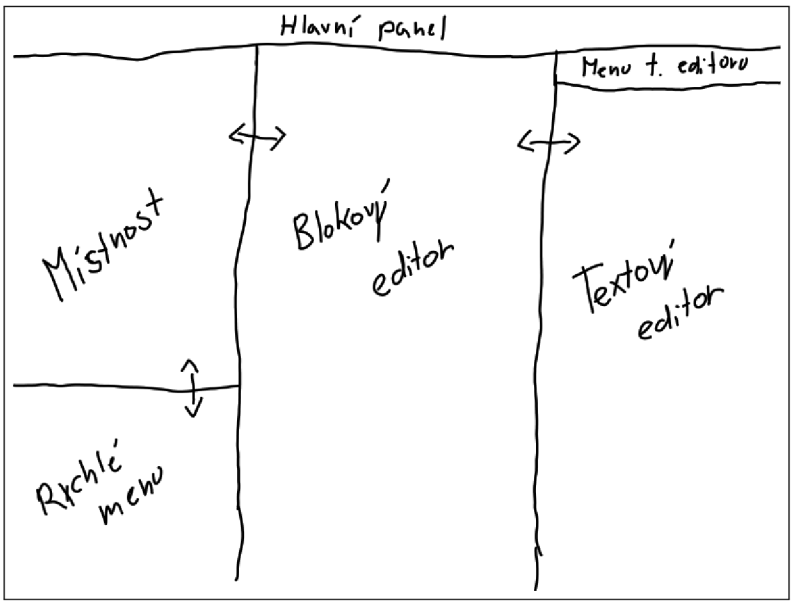
4.6 Návrh uživatelského prostředí

Hlavní inspirací pro uživatelské prostředí aplikace je opět verze robota Karla na MS-DOS, jež je vidět na obrázku 2.3. Zde je aplikace rozdělena do několika oken, z nichž každé plní svoji funkci, jako například okno s místností, okno pro editor kódu, a podobně. Okna jsou v nové verzi zachovány a obohacena knihovnou Split.js o změnu velikostí tažením jejich hran myší. Nová verze ale také potřebuje někam umístit nové blokové editování kódu. K tomu bylo v prostředí vytvořeno nové okno, které je vloženo mezi místnost a textový editor. Díky možnosti změny velikosti oken lze některé prvky aplikace zcela skrýt, pokud je uživatel zrovna nepotřebuje. Také je možné editovat program v blokovém editoru a živě sledovat přepis na text, pokud uživatel oba prvky zobrazil. Tím je vytvořeno jednoduché příjemné modulární uživatelské prostředí.

Původní verze měla pod oknem editoru prostor, ve kterém se zobrazovaly možnosti akcí pro funkční klávesy, odkud byla dostupná různá menu, funkce a podobně. Toto řešení bylo zvoleno, protože MS-DOS nepodporuje myš. Nová aplikace toto vstupní zařízení podporuje, a proto je implementován navigační panel na horním okraji obrazovky. Poskytuje přístup k funkcím jako je spouštění a zastavování interpretace, ukládání, změnu jazyka a další. Tento prvek je použit, neboť je realizovaný ve velké části aplikací, se kterou se cílová skupina pravidelně setkává (např.: Youtube, Facebook, Instagram a podobně).

V průběhu implementace byly vytvořeny další prvky, které bylo potřeba umístit do pracovní plochy aplikace. Bylo proto vytvořeno nové okno rychlého menu, které poskytuje některé z funkcí aplikace, a přehled o aktuálním stavu interpretace. Dále byl přidán panel pro nastavení textového editoru, který se nachází na horním okraji okna s textovým editorem. V tom je možné vybrat, zda-li je je editor v módu psaní kódu pro robota, nebo zda-li má zobrazovat přepis z blokové reprezentace. Je zde dostupné menu pro editor s dalšími funkcemi a nastaveními. Náčrtek tohoto návrhu uživatelského rozhraní je na obrázku 4.2.

Při vytváření uživatelského prostředí se klade důraz na to, aby aplikace neskrývala vnitřní stav interpretace a veškeré informace byly pro uživatele z pedagogických důvodů k dispozici. V prostředí by měly být přímo zobrazeny aktuální hodnoty proměnných, pozice interpretu v kódu a podobně.



Obrázek 4.2: Náčrtek navrženého uživatelského rozhraní

4.7 Návrh informačního systému

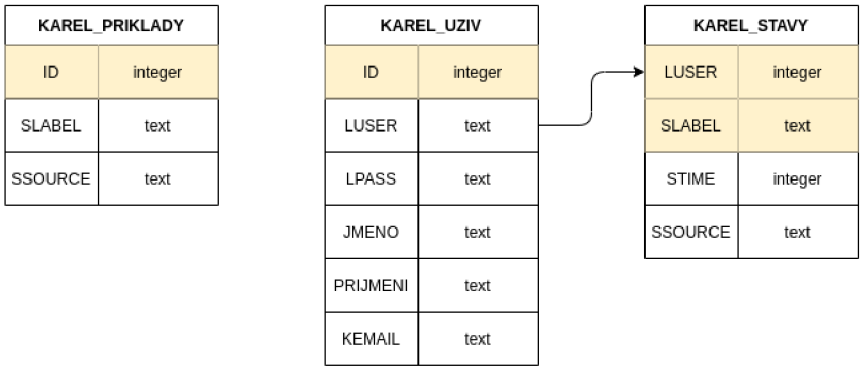
Aplikace by kolem sebe mohla mít postavený jednoduchý systém. Ten by mohl uživateli zpřístupnit seznam příkladů a vysvětlit některé z prvků aplikace. Dále by mohl poskytovat vedení uživatelských účtů, díky kterým by bylo možné ukládat stavy aplikace na serveru. Systém by ovšem neměl uživatelům komplikovat cestu k aplikaci, proto je na jeho hlavní straně jako první proklik do aplikace samotné. Dále se na hlavní straně nachází základní informace o aplikaci, jako je ovládání, popis uživatelského prostředí a podobně. V systému uživatele doprovází stejný ovládací panel jako v aplikaci, ovšem obsahuje lehce pozměněnou sadu tlačítek.

Tlačítko menu zůstává. Po kliknutí zobrazuje sekce hlavní strany, na které je po prokliku stránka přesunuta. Tlačítko jazyků zůstává stejné. Uprostřed panelu je odkaz na hlavní stranu systému, tedy na tuto stranu. Vpravo se nachází přihlášení do systému. Pokud uživatel přihlášený není, je ikona nevyplněná. Jakmile se uživatel ve vyskakujícím menu přihlásí, ikona se vyplní, vedle se zobrazí uživatelské jméno přihlášeného uživatele a přiložené menu změní své položky. Tento panel je dostupný v celém systému. Systém umožňuje vytváření uživatelských účtů, a pokud je uživatel přihlášený, poskytuje zobrazení uložených pozic. Náčrtek hlavní stánky systému je na obrázku 4.4.

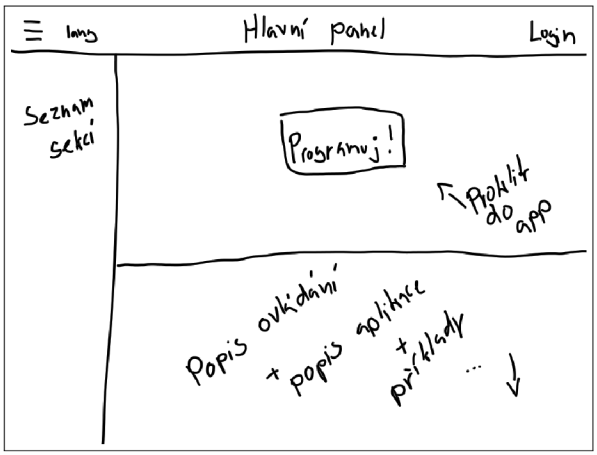
Data si systém ukládá do databáze. Ta má pro účely projektu velmi jednoduchou strukturu. Obsahuje jednu tabulku `KAREL_PRIKLADY` pro uložené pozice příkladů, tabulku `KAREL_UZIV` pro uložení uživatelských dat a tabulku `KAREL_STAVY` ukládající uložené stavy aplikace pro jednotlivé uživatele. Struktura databáze je znázorněna na obrázku 4.3.

Aplikace je vybavena tak, aby byla schopná požádat server o uloženou pozici pomocí jejího názvu. Server poté pošle aplikaci stav aplikace zakódovaný v zápisu typu JSON. Aplikace je schopná jej načíst, zpracovat a nastavit se podle něj. Příklad, který má být spuštěn, je definovaný přítomností názvu uloženého stavu za symbolem otazníku v url odkazu. Díky tomu lze na určité stavy aplikace odkazovat přímo pomocí odkazu, což bude využito i v tomto textu.

Texty, které jsou v systému zobrazené, a zadání všech příkladů jsou kvůli příjemné a rychlé editaci zapsány v jazyce Markdown. Systém si je při vytváření stránky přepíše do HTML a zobrazí uživateli. Pro příklady je také dostupné automatické generování knihovny v GUI prvku známém jako „accordion“. Tento prvek je automaticky generován podle obsahu složky `is/examples`, kdy složky v této struktuře tvoří jednotlivé kapitoly knihovny příkladů. Očekává se, že složky budou označeny číselně podle pořadí, a ve složce `is/examples` bude vedle těchto složek také přístupný soubor `aliases.txt`, který obsahuje názvy kapitol namapované na jejich čísla. Následné Markdown dokumenty v těchto složkách tvoří jednotlivé příklady. Každá kapitola může obsahovat soubor `intro.md`, jehož obsah se zobrazí před všemi příklady. Ostatní soubory typu Markdown jsou interpretovány jako příklady a první řádek tohoto souboru je pochopen jako název příkladu a je použit v záložce.



Obrázek 4.3: Schéma databázových tabulek – žluté podbarvení znázorňuje primární klíče, šipka napojení cizího klíče



Obrázek 4.4: Náčrtek hlavní stránky systému aplikace

Kapitola 5

Implementace vlastního řešení

Tato kapitola se bude zabývat vlastní implementací aplikace vytvářející nástroj pro tvorbu a interpretaci programovacího jazyka Karel. Projekt si dává za cíl vytvořit prostředí, ve kterém se budou moci v praxi žáci učit základům programování. Vize nové verze je popsána v kapitole 3.

5.1 Vnitřní stavba aplikace

V této kapitole bude probrána stavba aplikace. Jak bylo řečeno v kapitole 4.1, pro funkční část je využito jazyka JavaScript. Budou jmenovány jednotlivé třídy, které tvoří kostru programu, popsány jejich funkcionality a umístění v složce projektu. Soubory, o kterých bude řeč, se z velké části nacházejí ve složce `js/src`. Třídy byly již lehce naznačeny v kapitole 4.2 a ve stejném pořadí budou rozebrány i zde. Struktura a reference mezi třídami je znázorněna na obrázku 4.1.

Třída `room`

V úplném základu leží třída `room`, jejímž hlavním úkolem je vizualizace místnosti. Základní místnost se skládá z bílých dlaždic, kam se může robot postavit. Dále budou tyto dlaždice označovány jako bloky. Pro vytvoření instance třídy je potřeba konstruktoru předat objekt scény dostupné z knihovny Three.js, který určuje, kde bude místnost vykreslena. Tento parametr je uložen jako atribut objektu. Dále třída ukládá aktuální stav místnost v atributu `roomDataArray`, což je dvou-dimenzionální pole, které pro každý blok místnosti ukládá následující data:

- `blockObject` – Objekt, který ukládá grafický objekt bloku v místnosti, který je vygenerovaný knihovnou Three.js.
  - `cube` – Grafický objekt bílá dlaždice bloku.
  - `line` – Grafický objekt černého zvýraznění hran bloku.
- `bricks` – Počet cihel na políčku.
- `brickObjects` – Pole grafických objektů výplně cihel vytvořených grafickou knihovnou Three.js. Čím je cihla výše v komínku cihle na bloku, tím vyšší index má v poli.
- `brickObjectsLines` – Pole grafických objektů bílého ohraničení hran cihel, platí pro ně stejné vlastnosti jako pro grafické objekty výplně cihel.

- `mark` – Pravdivostní hodnota, `true` pokud je na bloku značka, `false` pokud ne.
- `markObject` – Grafický objet značky na daném políčku.
- `inRoom` – Pravdivostní hodnota, zda-li je blok v místnosti či není přítomen.

Podle indexu v dvou-dimenzionálním poli je určena pozice bloku v místnosti, a tím je ke každému bloku přiřazen jeden objekt ukládající popsaná data. Tato struktura je vytvořena při zavolání procedury `draw(controls, countX, countY)`. Ta vykreslí místnost do dané scény. Rozměry místnosti jsou definovány pomocí parametrů `countX` a `countY`. Při vytváření místnosti se nastavuje i ovládání a kamera, které se dostanou do základní pozice. Nakonec procedura `draw` přidává do každého rohu místnosti zdroj světla, aby vložené objekty v místnosti byly osvětleny a byly vidět jejich barvy. Opakem této procedury, která smaže zobrazení místnosti ze scény, je metoda `erase`.

Mimo generování a mazání místnosti třída také poskytuje metody, které přidávají či odebírají prvky do nebo z místnosti. Těmto procedurám se vždy předávají souřadnice, kam je žádáno objekt přidat, a současně s vytvořením a zobrazením objektu ve scéně se upraví výše popsaná struktura `roomDataArray`. Příkladem těchto funkcí jsou `addBrickToPos(posX, posY)` a opačná funkce `removeBrickFromPos(posX, posY)`. Pro zajištění správných pozic značek (tedy vždy na vrcholu komínku cihel, nebo přímo na bloku) je při pokládání a odebírání cihel volána procedura `updateMark(posX, posY)`, která na daných souřadnicích nastaví správné umístění značky.

Poslední funkcionalitou této třídy je ukládání a načítání stavu místnosti. Pro uložení stavu je vytvořena procedura `saveRoom()`, která navrací objekt podobný již popsané struktuře `roomDataArray`, avšak bez uložených grafických objektů. Pro načítání je vytvořena dvojice funkcí `checkLoadFileRoom(dataJson)` a `loadRoom(controls, dataJson)`. Nejprve je doporučeno načítaná data, předávaná jako parametr `dataJson`, zkontrolovat pomocí funkce `checkLoadFileRoom(dataJson)`, která vrátí pravdivostní hodnotu popisující, zda-li jsou data v pořádku. Následně se provede načtení funkcí `loadRoom(controls, dataJson)`. Té se k datům dostupných v parametru `dataJson` navíc předává objekt `controls` poskytnutý knihovnou Three.js, který se využije stejně jako v případě procedury `draw`. Třída neposkytuje možnost vložení objektu robota a jeho vykreslení v místnosti. K tomuto účelu je využita třída `karel`, která ukládá referenci na instanci této třídy.

Třída `karel`

Třída `karel` má za úkol vizualizovat robota v místnosti a realizovat základní příkazy, které jazyk Karel poskytuje. Konstruktor této třídy vyžaduje parametry `scene` a `controls` poskytnuté knihovnou Three.js. Při zavolání konstruktoru se vytvoří již popsaná instance třídy `room` a vytvoří se tak základní místnost.

Důležitou procedurou této třídy je funkce `draw(fun)`. Ta definuje atributy objektu, které určují souřadnice robota v místnosti a jeho natočení, dále načte grafickou reprezentaci robota z `.gbl` souboru, správně jej umístí do scény a uloží. Pro uložení pozice robota slouží atributy `orientation`, `positionX` a `positionY`. Natočení je reprezentováno celým číslem od nuly do tří, přičemž při spuštění aplikace je robot inicializován s otočením rovným nule. Při otáčení doprava se otočení inkrementuje a při otočení doleva dekrementuje. Pokud nastane přetečení z oboru hodnot, provede se korekce. Po načtení objektu a jeho vložení do místnosti se zavolá parametr `fun` funkce `draw(fun)`, pokud je funkcí. Toto se využívá při načítání definovaných stavů při spuštění aplikaci, protože po načtení a vložení grafického

objektu robota je již aplikace inicializovaná. Komplementární procedurou k `draw(fun)` je funkce `erase()`, která odstraní grafickou reprezentaci robota ze scény.

Třída dále poskytuje pomocné procedury jako opravení výšky robota podle políčka na kterém stojí (aby vždy stál na vrcholu komínku cihel) – `correctHeight()` a funkce definující políčko před Karlem – `tellPositionInFrontX` a `tellPositionInFrontY`. Následně jsou již k dispozici procedury určující chování jednotlivých příkazů jazyku Karel, jako je otáčení (`turnRight()` a `turnLeft()`), krok vpřed i s kontrolou proveditelnosti kroku (`goForward()`, pokládání a odebírání cihel a značek, přehrání zvuku pro příkaz `pip` a odstranění a navrácení bloku do místnosti. Implementovány jsou i podmínky (`isWall()`, `isBrick()`, `isMark()` a `isMark()`).

Dostupné jsou také procedury, které umožňují změnu místnosti pomocí `resizeRoom( valueX, valueY)`, kde se pomocí parametrů specifikují nové rozměry místnosti, navrácení kamery scény do počáteční pozice pomocí `homeCamera()` a sada procedur pro ukládání a načítání pozice. Mezi ně patří `saveRoomWithKarel()`, která vytvoří ukládací řetězec místnosti a přiloží pozici a orientaci robota v této místnosti.

Dvojice `checkLoadFileKarelAndRoom(dataJson)` a `loadRoomWithKarel(dataJson)` kontrolují a načítají uloženou pozici předanou v parametru `dataJson`.

Třída `command`

Další v hierarchii tříd je `command`. Tato třída má za úkol překládat klíčová slova instrukcí robota na akce, které robot skutečně v místnosti provede, a uložit uživatelem definované příkazy a podmínky. Doplňuje také kompletní sadu příkazů o určování rychlosti interpretace a ukládá rychlost interpretace. Při vytváření je potřeba konstruktoru předat parametr `karel`, který určuje robota, ke kterému se vztahuje toto ovládání, a parametr `math`, kde je očekávána třída řešící rozšíření o výrazy a proměnné – bude popsána jako příští. Jsou vytvořeny atributy `commandList` a `conditionList`, které jako slovníky ukládají uživatelsky definované příkazy a podmínky. Do těchto slovníků je během překladu vkládána vnitřní reprezentace kódu. Podrobněji bude popsána později v sekci 5.4. Prozatím stačí na vnitřní reprezentaci pohlížet jako na posloupnost tak zvaných tokenů, které jsou uložené v poli. Třída umožňuje jejich ukládání, vyhledávání a získávání pomocí sady procedur.

Metoda `executeCommand(dictKey)` umožňuje vykonávání vestavěných příkazů jazyka Karel, kde `dictKey` je klíč pro daný příkaz ve slovníku aplikace, který bude popsaný v kapitole 5.1. Podobně umožňuje vyhodnocení podmínek jak nativních, které jsou dostupné v jazyce Karel, tak uživatelsky definovaných, případně vyhodnocení matematických výrazů jako podmínek. Nakonec poskytuje sadu funkcí umožňujících změnu rychlosti interpretu jazyka.

Třída `math`

Tato třída byla přidána pro rozšíření jazyka o matematické výrazy a proměnné. Třída tedy poskytuje sadu funkcí, které umožňují kontrolu a výpočet matematických výrazů s proměnnými. Proměnné jsou uloženy v atributu třídy `variables`. Jedná se o slovník, který nejprve ukládá všechny rámce platnosti podle funkce, tedy obsahuje jako klíče názvy všech příkazů a podmínek, a k tomu přídavný globální rozsah. V těchto položkách je uložené jedno pole, jehož velikost symbolizuje zanoření v rekurzi. Nejvyšší index je vždy ten, který se aktuálně používá při interpretaci. Položkami pole jsou opět slovníky, které nyní mají strukturu, kde je názvu proměnné přiřazená hodnota. Třída je vybavena procedurami, které umožňují práci s touto ukládací strukturou.

Pokud je ze struktury čtena proměnná, využije se funkce `getVariable(name, scope)`, které se v parametru `name` předá název požadované proměnné a v parametru `scope` název příkazu či podmínky, ve které se právě interpret nachází. Funkce nejprve prozkoumá lokální rozsah příkazu či podmínky, a pokud ji nenajde zde, prohledá globální rámec a vrátí hodnotu nalezené proměnné.

Dále třída realizuje načítání a kontrolu výrazů pomocí precedenční syntaktické analýzy, během níž je vytvářena stromová reprezentace výrazu, která je uložena do vnitřního zápisu programu. Tato část bude rozebrána podobněji v sekci 5.4. Tuto stromovou strukturu je třída také schopná vypočítat pomocí rekurzivní procedury, o které bude řeč v sekci 5.5.

Nakonec třída umožňuje vytváření jednoduchého přehledu ukládací struktury proměnných. Obsahuje funkci `createVariableOverview(dictionary)`, která vytváří HTML tabulku. Ta obsahuje aktuální výpis všech definovaných proměnných ve všech rozsazích platnosti.

Třída `interpret`

Další třídou v posloupnosti je třída `interpret`, která vytváří instance tříd `command` a `math`, pomocí kterých komunikuje s robotem a řeší jím přiřazené podproblémy. Zároveň má tato třída přístup k textovému a blokovému editoru. Třída poskytuje funkce sloužící pro načítání kódu, kontrolu syntaxe a interpretaci programů jazyku Karel, které budou popsány v sekcích 5.4 a 5.5. Dovoluje interpretaci spouštět z různých editorů v různých režimech, které jsou v prostředí dostupné.

Interpret zaznamenává i nastavení o konkrétním světovém jazyku syntaxe pomocí nastavení příslušného jazykového balíku. Dále také umožňuje přepis textové podoby zápisu kódu v jazyce Karel do blokové podoby. Propaguje ukládací a načítací funkce popsané ve třídě `karel`, které rozšiřuje o ukládání kódu jak v textové, tak v blokové podobě ve formátu JSON. Ten poté umožňuje stáhnout jako soubor s koncovkou `.karel` a stejný soubor poté zkontrolovat a načíst. Nakonec umožňuje překládání kódů jazyka mezi jazyky, ke kterým je poskytnutý jazykový balík.

Jazykové balíky

Posledními jsou soubory jazyků dostupné ve složce `js/src/lang`. Ty jsou do aplikace načítány pomocí funkce `changeLanguage(langFile, firstRun)`, která přijímá parametr `langFile`, který určuje umístění souboru s jazykovým balíčkem, a `firstRun`, kde se očekává pravdivostní hodnota. Když je nastavena na `false`, provede se zároveň s načtením nového jazyka přeložení zapsaného kódu. Tato funkce je dostupná v souboru `main.js`.

Jazykové soubory obsahují slovník, kde je indexem anglická verze klíčových slov, a jako data jsou klíčová slova v požadovaném jazyku. Slovník obsahuje mimo klíčová slova jazyka i texty pro uživatelské rozhraní, texty pro blokové rozhraní, texty v dialogových oknech a hodnoty pro textový editor, na základě kterých se podbarvuje text, určuje zalamování, a podobně. V projektu je zatím dostupný český a anglický jazyk.

Soubor `main.js`

Soubor `main.js` je hlavní spouštěcí soubor aplikace. Vytváří všechny důležité prvky a spojuje je s uživatelským rozhraním. Je vybaven již popsanou funkcí měnící jazyk pomocí jazykového balíčku, který zvládá změny i při běhu aplikace. Vytváří blokový a textový editor aplikace a nastavuje jejich parametry. Umožňuje modifikování seznamu dostupných

bloků a jejich kategorií v blokovém editoru. Obsahuje také vykreslovací funkci knihovny Three.js, která je připravena i na změnu okna aplikace či okna místnosti.

5.2 Implementace textového rozhraní

Textové rozhraní zprostředkovává knihovna jazyka JavaScript s názvem Advanced Code Editor, dále jen ACE, která je popsaná v sekci 4.3. Všechny zdrojové kódy knihovny jsou dostupné ve složce `js/ace`. Aby byla schopná podporovat jazyk Karel, bylo potřeba vytvořit konfigurační soubor specifický pro tento jazyk. Tento soubor obsahuje pravidla pro podbarvování zdrojových textů, podporu pro skrývání částí zdrojového textu a automatické zalamování. Tyto funkce jsou vytvořené tak, aby podporovaly změnu jazyka pomocí již popsaných jazykových balíků.

Tyto editory jsou v aplikaci dostupné dva. Jeden slouží přímo pro editaci kódu jako textu, druhý je implementovaný pro zobrazování přepisu blokového rozhraní. Ten druhý je pouze v režimu pro čtení. I přesto jsou ale dostupné funkce regulárního editoru. Knihovna poskytuje tokenizátor, který je dále využit při načítání a kontrole kódu. Barvy editoru a podbarvení kódu jsou definovány v souboru tématu. Pro aplikaci je modifikován originální soubor poskytnutý knihovnou `theme-chrome.js`.

ACE je v aplikaci rozšířený o podporu breakpointů, které je možné zadávat do editoru pomocí kliknutí do levé lišty na řádek, kde uživatel chce interpretaci zastavit. Breakpoint se zobrazí jako zaoblené ohraničení čísla řádku, a pokud se interpret v ladícím režimu dostane na takový řádek, interpretace je pozastavena. Místo zastavení lze odstranit opětovným kliknutím na levý okraj řádku, kde je breakpoint k odstranění, nebo je v aplikaci dostupné tlačítko pro odstraněni všech breakpointů.

Dostupné je i automatické doplňování, které se zobrazuje při psaní kódu. To napovídá slova z dostupných slov, která jsou editoru zrovna definována, stejně jako ze slov, která již uživatel editoru zadal, nebo jsou definována jazykem. Některé funkce, které tu jsou popsány, je možné nastavovat z uživatelského prostředí. Pokud uživatel chce spustit nějaký program z textového rozhraní, provede se nejprve kontrola kódu. Její průběh bude popsaný v sekci 5.4. Pokud je ale při kontrole nalezena nějaká chyba v kódu, je zobrazena v editoru tak, že v levém okraji se objeví ikonka chyby a je červeně podtrženo slovo, které bylo určeno jako chybné. Po najetí na ikonku chyby se zobrazí text popisující chybu.

5.3 Implementace blokového rozhraní

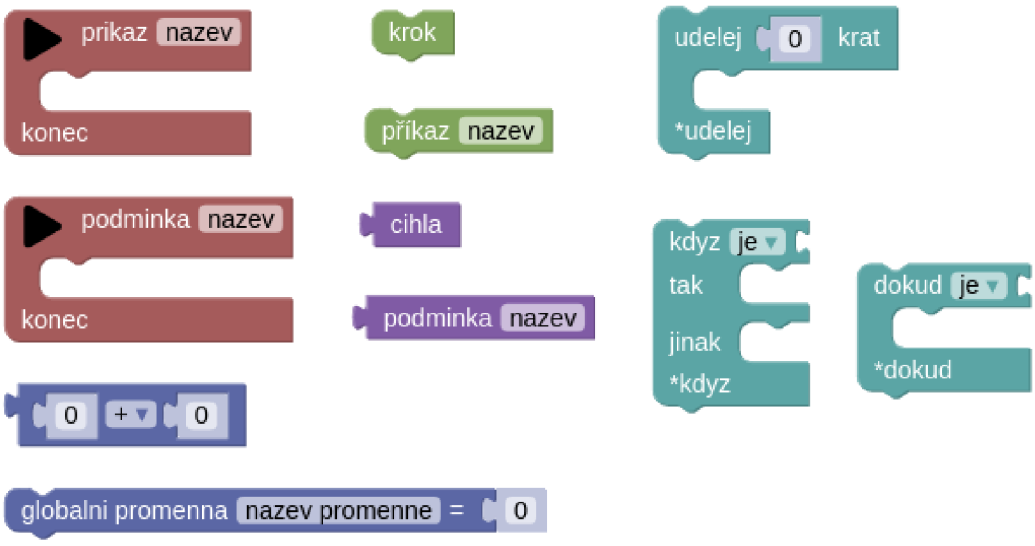
Blokové rozhraní je implementováno pomocí knihovny Blockly. Tato knihovna je dostupná ve složce `js/blockly`. Pro tuto knihovnu bylo potřeba definovat bloky a jejich přepis na textovou podobu kódu. Ten byl vytvořený s pomocí nástroje od vývojářů této knihovny[1]. Výsledné bloky jsou definovány v jazyce JavaScript v souboru `js/blockly/karel_blocks.js`. Jsou vytvořené tak, aby co nejvíce připomínaly textovou reprezentaci a uživatelé se tak mohli jednoduše po seznámení s prostředím přesunout do programování v textovém editoru. Každý tento blok má poté přidělený generátor, díky němuž je možné převádět bloky na textovou podobu kódu, kterou lze zobrazit a interpretovat v přiděleném textovém editoru. Ten je definovaný v souboru `js/blockly/karel_blocks_generator.js`. Oba dva skripty umí reagovat na jazykové balíky a měnit svůj výstup podle nich.

[1] Dostupný na https://blockly-demo.appspot.com/static/demos/blockfactory/index.html

V aplikaci jsou dostupné různé druhy bloků. Základním typem je blok, který má pouze horní a dolní konektor a vnitřní pevný text. Zástupci takových bloků jsou hlavně položky z kategorie „Příkazů“, jako například blok „krok“ na obrázku 5.1. Takovéto bloky je možné skládat za sebe a vytvářet pomocí nich posloupnosti příkazů. Dalším typem jsou bloky vstupů, které nemají horní a spodní konektor, ale boční. Jejich typickými zástupci jsou podmínky a matematické výrazy – na obrázku je například blok „cihla“. Tyto bloky lze připojit do některých bloků z kategorie „Řízení toku“, jako je blok struktury „kdyz“ nebo „dokud“. Pravý konektor očekává i blok přiřazení do proměnné, ale ten přijímá pouze bloky ze sekce matematika a podmínky připojit nejde, ovšem matematiku do bloků ze sekce „Řízení toku“ přiřadit lze. Nakonec jsou dostupné i bloky, které mají vnitřní konektory a dokáží obalit posloupnost jiných bloků. Mezi jejich zástupce patří uvození příkazu či podmínky z kategorie „Základ“ nebo bloky v kategorii „Řízení toku“. Ty z druhé kategorie mají také vnější horní a spodní konektor. Na obrázku 5.1 jsou vidět zástupci, kteří jsou zabarvení do světle modré barvy.

Blok může kromě prostého textu obsahovat i další prvky. Jedním z takových prvků je textové pole, kam může uživatel zadat požadované hodnoty. Většinou není omezeno, co lze do pole zadat, pouze u bloku čísla s výchozí hodnotou „0“ lze zadat pouze kladná čísla. Jinak jsou v těchto textových polích očekávány názvy příslušných příkazů, podmínek, a podobně. Dalším prvkem je rozbalovací menu, kde uživatel vybere položku ze seznamu. To je dostupné u bloků cyklů „kdyz“ a „dokud“, kde určuje předponu podmínky. Poté jsou také v matematických blocích, kde se jimi určuje operace ve výrazu.

Pokud je jako vstup očekávána matematika, je přítomný na jeho místě nejprve průhledný blok obsahující nulu, což urychluje tvořené kódu. Ten je možné nahradit jakýmkoliv jiným matematickým prvkem. Pokud totiž uživatel chce pouze dosadit číslo, může jej rovnou vyplnit a nemusí přetahovat nový blok pro číslo. Trikem, který se dá při zadávání matematických výrazů využít, je použít blok pro zadání proměnné a napsat do něj celý výraz, což ocení uživatelé s klávesnicí.



Obrázek 5.1: Příklady implementovaných bloků v prostředí Karel 3D

Knihovna poskytuje pracovní plochu, na které je dostupná sada nástrojů. V ní jsou umístěny bloky do kategorií. Bloky jsou zde rozděleny podle své funkcionality. Každá z těchto kategorií má přiřazenou svoji barvu, kterou respektují i bloky patřící do dané kategorie. Základní dostupné kategorie jsou tyto:

- Základ – červená: uvození příkazu nebo podmínky
- Příkazy – zelená: základní instrukce pro robota
- Řízení toku – světle modrá: cykly a větvení programu
- Podmínky – fialová: podmínky pro větvení
- Matematika – tmavě modrá: matematické výrazy a proměnné

V kategoriích „Funkce“ a „Podmínky“ se pak nachází prázdné bloky, pomocí kterých lze volat uživatelem definované příkazy, případně podmínky. Kategorie „Základ“ poskytuje bloky pro definování vlastních příkazů a podmínek. Jsou vybaveny tlačítkem pro spuštění interpretace daného kusu kódu. Pokud je načítán kód do blokového prostředí, definované příkazy a podmínky jsou načtené jako zabalené bloky. Rozbalení je dostupné v kontextovém menu, které se zobrazí po kliknutí pravým tlačítkem na daný blok. V tomto menu jsou dostupné i komentáře v blokovém prostředí. Většina těchto komentářů je přepsána i do textového prostředí (nejsou brány v potaz komentáře jednotlivých prvků matematických výrazů). Pro odstranění bloků z plochy je možné buď požadovaný blok pomocí „drag-and-drop“ systému přesunout zpět do panelu nástrojů, a nebo do koše umístěného v pravém dolním rohu editoru. Posledních pár smazaných bloků nebo struktur je dostupných po kliknutí na ikonku koše v pravém dolním rohu editoru. Nad touto ikonou je dostupné i přiblížení či oddálení pracovní plochy. Toto ovládání umožňuje i kolečku myši. Nad ikonami změny přiblížení je centrovací tlačítko, které přesune náhled pracovní plochy na prostředek a resetuje přiblížení na výchozí hodnotu.

5.4 Kontrola kódu

Pokud uživatel chce interpretovat libovolný kus kódu, je kód před samotnou interpretací zkontrolován a převeden na vnitřní reprezentaci, která je teprve interpretována. Nejprve je provedeno načtení kódu po jednotlivých slovech. Za tímto účelem je použita funkce knihovny ACE `TokenIterator`, která umí obsah editoru takto načítat. Jeho výstupem jsou jednotlivá slova zapsaná v editoru, ke kterým jsou připojena metadata jako řádek a sloupec, kde se dané slovo nachází, typ, a podobně. Z těchto informací se uloží ty podstatné, některé další jsou připojeny. Tvoří tak strukturu tokenu, která je následující:

- `value` – hodnota, která byla přečtena z editoru.
- `column` – sloupec začátku slova v editoru.
- `row` – řádek, kde se slovo v editoru nachází.
- `dictKey` – klíč do jazykového slovníku aplikace. V každém jazyce mají slova stejného významu stejný klíč.
- `meaning` – sémantický význam slova.

Toto rozdělení na tokeny vykonává funkce `nativeCodeTokenizer(editor, forInterpretation)` dostupná ve třídě `interpret`. Při tokenizování kódu se výsledné tokeny ukládají do struktury `codeArray`, která je reprezentována polem polí. Vnitřní Pole ukládají posloupnosti tokenů pro každý příkaz, podmínku či globální definici proměnné. Pokud je výsledek míněný k interpretaci, tokeny komentářů jsou zahozeny a globální definice jsou přesunuty na začátek struktury `codeArray`. Nakonec funkce ještě celou vytvořenou reprezentaci projde a rozdělí spojené matematické výrazy, které funkce `TokenIterator` knihovny ACE vytváří (místo tokenu pro více znaků je pro každý znak vytvořen jeho token).

Následně jsou provedeny první kontrolní průchody kódem. Při nich se kontrolují identifikátory nejprve příkazů a podmínek, poté globálních a nakonec lokálních proměnných. Kontrola prověřuje, zda-li jsou všechny identifikátory v kódu definované, a zda-li nedochází k redefinicím, které nejsou v jazyce povolené. Popsanou kontrolu realizuje funkce `preCheckRepair(codeArray, errors)`, které se předává výsledek tokenizující funkce v parametru `codeArray`. Pokud je objevena nějaká chyba, je zapsána do pole `errors`.

Následuje kontrola syntaxe podle LL(1) tabulky pro prediktivní syntaktickou analýzu, doplněná o precedenční analýzu matematických výrazů. Nejprve bude popsána kontrola syntaxe jazyka. Ta využívá zásobníku a LL(1) tabulky. Při kontrole se algoritmus snaží na zásobníku pomocí pravidel vygenerovat stejnou syntaxi, kterou zadal uživatel, čímž se kontroluje, zda-li je správná. Zásobník může obsahovat terminály, i neterminály. Terminály jsou koncová slova jazyka Karel, kdežto neterminály je třeba dále rozgenerovat pomocí pravidel. Na začátku je na vrcholu zásobníku neterminál `<start>`. Algoritmus má přístupnou tabulku, kde se na řádky označují neterminály a ve sloupečcích jsou terminály. Neterminály se mohou nacházet na vrcholu zásobníku, terminály mohou být na začátku řetězce tokenů, které jsou kontrolovány. Příslušné políčko v tabulce, které sedí do popsaného páru obsahuje číslo pravidla, které se má použít. Pravidlo popisuje, čím se má neterminál na vrcholu zásobníku nahradit, aby se mohlo v kontrole pokročit dál. Pokud se v buňce tabulky nenachází validní číslo pravidla, je detekována chyba. Pravidla jsou vidět v seznamu A.2 a tabulka A.1 zobrazuje jejich použití.

Pokud se na vrcholu zásobníku nachází terminál, musí se shodovat s terminálem, který je na začátku zatím nezpracované části vstupního pole tokenů, který vznikl z analyzovaného zdrojového kódu. Pokud se neshodují, je detekována chyba. Pokud se terminál na zásobníku shoduje s terminálem ve vstupním poli, jsou z obou míst odebrány. Terminál ze vstupního řetězce (neboli token) je vložen do vnitřní reprezentace kódu, která se ukládá v instanci třídy `command`, a to takovým způsobem, že každý příkaz či podmínka má svoji posloupnost tokenů, které je definují (v JavaScriptu je tento fakt uložen pomocí slovníku odkazujícího na pole).

Pokud se při kontrole kódu narazí na terminál matematického výrazu, předává se kontrola prediktivní syntaktické analýze pro matematické výrazy, která je implementovaná ve třídě `math`. Při této kontrole je nejprve načten celý výraz (jsou brány tokeny, dokud mají sémantický výraz `expression`), všechny jeho tokeny jsou ze vstupního řetězce vyřazeny a nazpět se vloží jediný token reprezentující celý matematický výraz. Následuje kontrola, při které se využívá precedenční tabulka 5.1 a pravidla 5.2.

Nejprve je na zásobník umístěn terminál `$` značící konec matematického výrazu. Stejný terminál je vložen na konec načteného výrazu (ten je poznán podle toho, že následující slovo není možné počítat mezi výraz). Následně je spuštěna samotná precedenční analýza. Ta podle terminálu na vrcholu zásobníku a příchozího terminálu z načteného výrazu vyhledá buňku v tabulce 5.1. Pokud je v buňce znak `<`, vloží jej na zásobník, ten se nepočítá jako terminál, ale jako neterminál. Dále se na zásobník přesune příchozí terminál z načte-

ného výrazu. Když je v buňce znak `>`, načtou se všechny terminály, které jsou v zásobníku do znaku `<` a výsledný řetězec se vyhledá v pravidlech 5.2. Pokud je nalezeno pravidlo, nahradí se jeho pravá strana na zásobníku za neterminál `E` (levou stranu pravidla). Pokud není pravidlo nalezeno, je nahlášena chyba. Kontrola je úspěšná, pokud z načteného matematického výrazu nic nezbude a na zásobníku je uzavírací symbol `$`. Během kontroly se vytváří derivační strom tohoto výrazu, který je poté uložen do token reprezentujícího celý zkontrolovaný výraz. Ten po kontrole v zápisu kódu zbude. Strom se využívá při interpretaci pro vypočítání hodnoty výrazu.

Pokud je během kontroly kódu nalezena nějaká chyba, je uložen její text společně s tokenem do pole chyb. Na konci kontroly se seznam chyb projde, a pokud jsou přítomny nějaké záznamy, jsou zobrazeny v textovém editoru podtrhnutím tokenu a ikonkou v levém okraji editoru na řádku tokenu. Tato ikona zobrazuje i text chyby. Chyba je hledána vždy pouze jedna v jednom příkazu či podmínce. Pokud žádné chyby v celém kódu nalezeny nejsou, je pokročeno k interpretaci. Pro více informací o těchto metodách a o kontrole syntaxe je možné prostudovat knihu Alexandra Meduny [8] nebo nahlédnout do opory pro předmět Formální jazyky a překladače [9] vyučovaný na Fakultě informačních technologií Vysokého učení technického v Brně.

Token na vrcholu zásobníku / Vstupní token	*, /, %	+, -	<, <=, >, >=, ==, !=	číslo, proměnná	(	)	$
*, /, %	>	>	>	<	<	>	>
+, -	<	>	>	<	<	>	>
<, <=, >, >=, ==, !=	<	<		<	<	>	>
číslo, proměnná	>	>	>			>	>
(	<	<	<	<	<	=	
)	>	>	>			>	>
$	<	<	<	<	<		

Tabulka 5.1: Tabulka precedenční syntaktické analýzy

Pravidla	
E → číslo, proměnná	E → E + E
E → E - E	E → E * E
E → E / E	E → E % E
E → E > E	E → E >= E
E → E < E	E → E <= E
E → E == E	E → E != E
E → (E)	

Tabulka 5.2: Pravidla precedenční analýzy

5.5 Interpretace kódu

Interpretace kódu se provádí, pokud proběhla jeho úspěšná kontrola. Prvním krokem interpretace je nalezení, který příkaz či podmínka má být spuštěna. To je určeno pozicí kurzoru v textovém editoru. Pokud se kurzor nachází v definiční části některého z příkazů či podmínek, tento příkaz či podmínka je spuštěna. Když se kurzor nenachází v žádném spustitelném kódu, je nahlášena chyba a interpretace končí.

Jestliže je určení začátku provádění programu úspěšné, textový editor je přepnut do režimu pouze pro čtení a interpret do režimu běhu. Ten zabraňuje opětovnému spuštění další interpretace, dokud nedoběhne aktuálně vykonávaný kód. Vytvoří se struktura `codePointer`, která určuje aktuální vykonávaný příkaz v kódu. Obsahuje řetězcovou položku `functionName`, která obsahuje název aktuálně vykonávaného příkazu či podmínky a číselnou položku `tokenPointer`, která obsahuje index do pole tokenů vykonávaného příkazu či podmínky. Ty jsou uložené ve vnitřní reprezentaci vytvořené při kontrole kódu. Dále interpret pro vykonávání kódu obsahuje zásobník ukládající pozice, odkud bylo v kódu odskočeno, který se jmenuje `programQueue`. Pokud je volán v příkazu či podmínce jiný příkaz či podmínka, ukládá se místo volání do této struktury a odskakuje se na začátek volané rutina.

Interpretace je vykonávána v krocích. Nejprve se načte aktuální token, na který je odkazováno strukturou `codePointer`. Podle významu tohoto tokenu se vyvolá příslušná akce, kterou poskytuje buď třída `interpret`, `command`, a nebo `math`. Jedná-li se o některý ze základních příkazů jazyka, provede jej třída `command`, která je k tomu vybavena funkcemi. Ty jsou popsány v sekci 5.1. Na další tokeny reaguje interpret následovně:

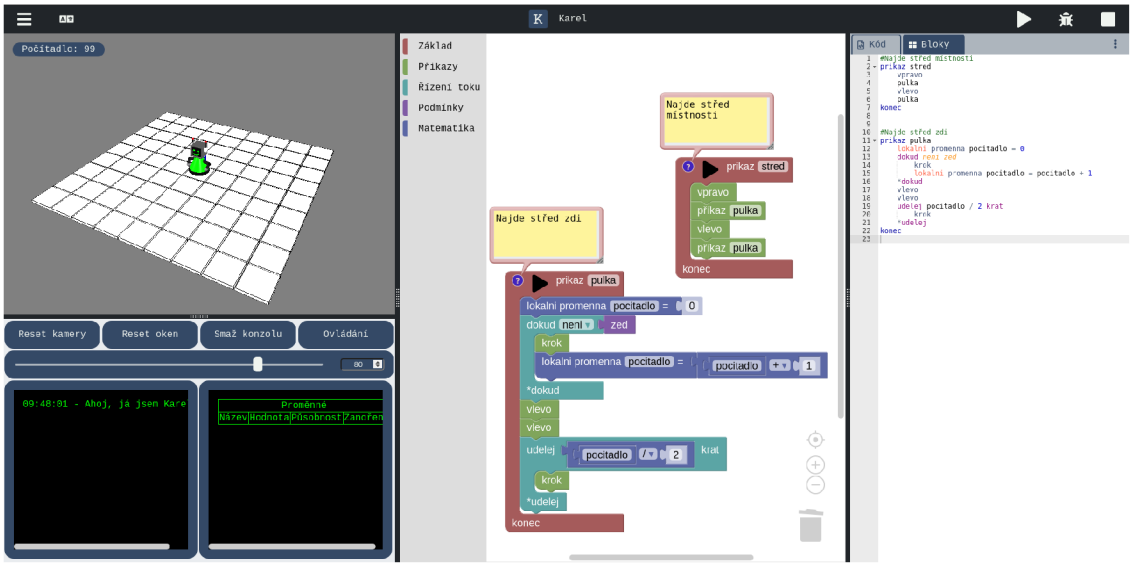
- `prikaz` nebo `podminka` – uvození příkazu nebo podmínky. Na začátku každého příkazu či podmínky je vytvořeno nové pole působnosti pro lokální proměnné, a to včetně opětovného volání přes rekurzi.
- `konec` – ukončení příkazu nebo podmínky. Interpret se podívá na svůj zásobník odskočení. Pokud je prázdný, interpretace končí. Pokud je neprázdný, nastaví se ukazatel do kódu podle vrcholu tohoto zásobníku a položka je ze zásobníku odstraněna.
- `udelej` – uvození struktury `udelej [N] krat`. Po rozpoznání této struktury se vyhodnotí počet opakování, které může být díky rozšíření matematická rovnice s proměnnými. Třída `math` vypočte počet opakování. Pokud je hodnota menší než jedna, přeskakuje se na korespondující uzávěru této struktury. Pokud je vypočtená hodnota větší než jedna, uloží se na vrchol zásobníku, který ukládá počty opakování této struktury, a pokračuje se dále v interpretaci.
- `*udelej` – ukončení struktury `udelej [N] krat`. Pro nalezení ukončení struktury interpret nahlédne na vrchol zásobníku ukládajícího počty opakování struktury `udelej`. Tuto hodnotu dekrementuje, a pokud je stále větší než nula, skočí ukazatel interpretu na začátek této programové konstrukce a opakuje cyklus. Pokud je hodnota rovna nule, je ze zásobníku tato hodnota odstraněna a pokračuje se v interpretaci dále.
- `dokud` – uvození struktury `dokud je/neni [podminka]`. Provede se vyhodnocení podmínky včetně přiloženého prefixu. Pokud je výsledek podmínky pravda, pokračuje se v interpretaci, pokud nepravda, přeskakuje se ukazatelem interpretace na uzavření této struktury.

- `*dokud` – ukončení struktury `dokud je/neni [podminka]`. Interpret vrátí ukazatel na začátek korespondující struktury, kde se v dalším kroku interpretace znovu provede vyhodnocení podmínky.
- `kdyz` – uvození struktury `kdyz je/neni [podminka]`. Nejprve se provede vyhodnocení podmínky včetně jejího prefixu. Pokud je výsledek pravda, pokračuje se dále v interpretaci, pokud je nepravda, přeskakuje se buď na korespondující klíčové slovo `jinak`, nebo na uzavření této struktury. Jazyk Karel podporuje jak verzi s větví `jinak`, tak bez ní.
- `jinak` – druhá větev struktury `kdyz je/neni [pdominka]`. Pokud interpret narazí na tento token, přeskakuje se na konec uvození struktury `kdyz`.
- `globalni` nebo `lokalni` – práce s proměnnou. Předá se třídě `math`, která vypočte přiložený matematický výraz a uloží jej do dané proměnné.

Podmínky jsou vyhodnocovány společně se svým prefixem. Pokud je podmínka jedna ze základních, je přímo určena. Jestliže je na místě podmínky výraz, vyhodnotí jej třída `math`. Když je výsledek roven nule, vyhodnotí se samotná podmínka bez prefixu jako nepravda, pokud je výsledek různý od nuly, je vyhodnocena jako pravda. Výsledek je poté modifikován prefixem. Dále může být přítomna uživatelem definovaná podmínka. Jestliže je nalezena taková podmínka, je nastaven ukazatel do kódu na začátek vykonávání dané podmínky. Výsledek se uloží a při navrácení do původního kódu se výsledek využije k rozhodnutí a smaže.

Pokud při běhu programu nastane některá z běhových chyb, například dělení nulou, nebo čtení nedefinované proměnné, interpretace se zastaví a chyba je oznámena v konzoli.

5.6 Uživatelské funkce a uživatelské rozhraní



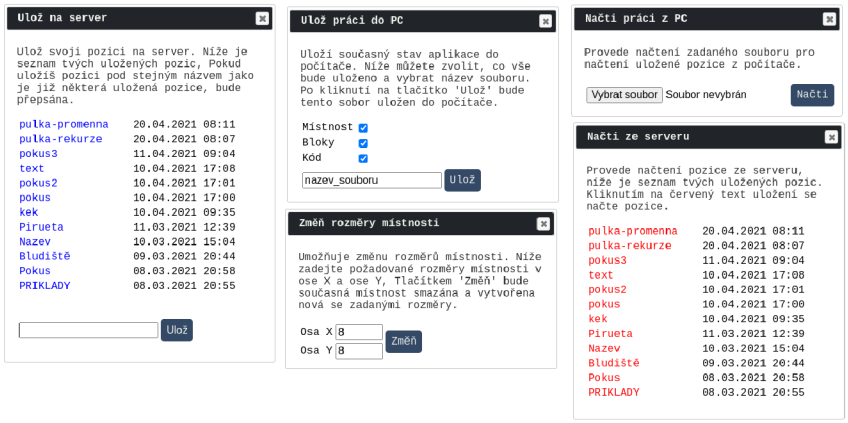
Obrázek 5.2: Implementované uživatelské prostředí nové verze

V této kapitole bude popsána implementace a rozložení vytvořeného uživatelského prostředí aplikace. Myšlenka pro něj byla nastíněna v kapitole 4.6. Výsledné implementované prostředí je znázorněno na obrázku 5.2. Nejdříve bude popsána horní lišta aplikace. Na té se zleva doprava nachází několik prvků. Prvním prvkem je hlavní menu aplikace, jehož položky se zobrazí po najetí kurzoru na ikonu. Mezi dostupné funkce v tomto menu patří:

- Změň místnost – Otevře dialog, ve kterém je možno nastavit rozměry místnosti. Vypsané hodnoty jsou aktuální nastavené. Po zvolení rozměrů a potvrzení se vymaže aktuální místnost s robotem a vygeneruje se vyplněná obdélníková místnost se zadanými rozměry. Robot je přesunut na výchozí pozici.
- Ulož do PC – Otevře dialog, ve kterém je možné vytvořit soubor, který ukládá současný stav aplikace. Je možné nastavit, které položky se mají do souboru uložit, přičemž ve výchozím stavu je nastaveno kompletní uložení aplikace. Pod výběrem je možné zvolit název ukládaného souboru. Kliknutím na tlačítko `Ulož` se požadovaný soubor vygeneruje a stáhne do počítače. Přípona ukládaných souborů je `.karel`.
- Načti z PC – Otevře dialog, který umožňuje načtení stavu aplikace ze souboru v počítači. Dostupné je tlačítko pro zvolení souboru, přičemž vyhledávání v počítači je omezeno pouze na soubory s koncovkou `.karel`. Po kliknutí na tlačítko `Načti` se povede načtení stavu ze zvoleného souboru, podle položek, které obsahuje.
- Nastav okna – Nastaví výchozí pozice oken aplikace podle jejich aktuálního rozložení. O své činnosti informuje do konzole.
- Ulož na server – Položka je zobrazena pouze, pokud je uživatel přihlášený. Otevírá dialog, pomocí kterého je možné uložit aktuální stav aplikace na server. Dialog obsahuje seznam již uložených stavů, ke kterým je přiložen čas jejich poslední modifikace.

Pokud uživatel klikne na název stavu, vyplní se do textového pole pod tímto seznamem. Toto pole určuje pojmenování uložené pozice. Jestliže je zvoleno jméno, které je již použito, bude tato pozice přepsána novými daty. Tlačítkem ulož se provede uložení kompletního stavu aplikace pod zadaným jménem.

- Načti ze serveru – Položka je zobrazena pouze, pokud je uživatel přihlášený. Otevře dialog pro načtení uložené pozice ze serveru. Dialog obsahuje seznam uložených pozic reprezentovaných červeným jménem a časem poslední modifikace daného záznamu. Pokud uživatel klikne na některý z červených názvů, je načtena odpovídající uložená pozice celé aplikace.



Obrázek 5.3: Dialogy v nové verzi prostředí jazyka Karel

Další položkou horního panelu je volba jazyků. Při najetí na položku se zobrazí možnosti. V případě jazyků to jsou čeština a angličtina. Pokud je jazyk přepnut, celá aplikace včetně zadaných kódů je přeložena do daného jazyka. Jazyk si aplikace ukládá do paměti počítače a po zavření a opětovném spuštění bude nastaven naposledy zvolený jazyk. Následuje tlačítko `Karel` s ikonkou aplikace, která po kliknutí přesune uživatele na hlavní stranu informačního systému aplikace. Poslední tři tlačítka ovládají interpret v oblasti textového editoru a mají následující funkcionalitu:

- `Spusť` – Spustí interpretaci kódu. Provede se takový program, mezi jehož uvozovacím slovem `prikaz` nebo `podminka` a ukončovacím slovem `konec` se aktuálně nachází kurzor editoru.
- `Krokuj` – Spustí interpretaci kódu podobně jako předchozí varianta. Rozdílem je, že interpretace není provedena vcelku. Pokud jsou nastaveny breakpointy, interpretace je pozastavena na nich, a další její postup musí být potvrzen tímto tlačítkem. Pokud breakpointy nastaveny nejsou, interpretace je zastavována po každém kroku a čeká na potvrzení tímto tlačítkem.
- `Zastav` – Pokud je zrovna prováděna interpretace, tímto tlačítkem je možné ji zastavit. Platí jak pro normální běh interpretace, tak pro krokovací režim.

To jsou všechny prvky dostupné z hlavního panelu aplikace a následují samotná okna. Před popisem jejich samotných obsahů je nutné podotknout, že jsou uživatelsky nastavitelná. Rozdělují je tmavé kraje, která jsou zároveň táhly. Ty lze posouvat podél jejich kratší strany zakliknutím myši a tažení. Prvky lze tímto způsobem různě zmenšovat, zvětšovat, případně úplně skrývat.

Okna budou nyní popisována zleva doprava. Prvním oknem je tedy dvojice podoken, které obsahují místnost s robotem a rychlý přehled aplikace. Pouze v těchto dvou položkách lze okna nastavovat vertikálně. Okno místnosti s robotem graficky vyobrazuje Robota ve svém prostředí. Kameru pohledu lze po kliknutí na prvek ovládat myší a šipkami na klávesnici. Toto okno mimo grafického pohledu poskytuje i upozorňující bubliny, které se zobrazují v levém horním rohu okna. Mezi ty patří následující:

- Počítadlo – Pokud je spuštěna interpretace, zobrazí se bublina počítadla, které ukazuje, kolik bylo doposud potřeba provést iterací interpretru. Počítadlo se dá vynulovat kliknutím na tuto bublinu.
- Ovládáš – Pokud je okno vizualizace aktivní, zobrazí se tato bublina. Pokud je bublina vizualizována, je možné ovládat robota přímo pomocí klávesnice. Pro zrušení ovládáni je možné kliknout na bublinu.
- Běží – Pokud je spuštěna interpretace, zobrazí se tato bublina. Pokud je zobrazena, zablokuje se přímé ovládáni robota. Po ukončení interpretace je bublina odstraněna. Kliknutím na bublinu se právě probíhající interpretace zastaví, má tedy stejnou funkcionalitu jako tlačítko `Stop`.

Obrázek 5.4: Příklad upozorňujících bublin

Pod místností se nachází prostor rychlého přehledu stavu aplikace. Zde jsou prvky, kterými je možné ovládat aplikaci a sledovat její vnitřní stav. Tento element obsahuje následující prvky:

- tlačítko „Reset kamery“ – Vrátí kameru pohledu místnosti do základní pozice.
- tlačítko „Reset oken“ – Vrátí rozložení oken do základní pozice.
- tlačítko „Smaž konzolu“ – Vymaže obsah konzole.
- tlačítko „Ovládání“ – Zobrazí ovládání přes tlačítka, což může být nápomocné pro dotyková zařízení. Ta se objeví na prvním řádku tohoto přehledu.
- posuvník rychlosti – Nastaví rychlost interpretace robota v procentech. Vedle posuvníku je číselná reprezentace jeho hodnoty opět v procentech. Posuvník je aktualizován i pomocí klíčových slov jazyka `rychle` a `pomalu`.
- konzola – Do ní jsou zapisovány informace od aplikace pro uživatele. Většinou jsou to různá chybová hlášení nebo potvrzení nastavení.

- přehled proměnných – Zde jsou přehledně v tabulce vypsány proměnné a jejich hodnoty. Tabulka obsahuje název proměnné, její hodnotu, její působnost v rámci příkazů a podmínek a zanoření rekurze, ve které byla proměnná v daném příkazu či podmínce vytvořena.

Na tuto dvojici prvků navazuje po pravé straně okno blokového editoru. Ten je podrobně popsaný v kapitole 5.3. Blokový editor je po pravé straně následován oknem textového editoru. Ten má na své horní straně k sobě přiřazené ovládací prvky. Nejprve to jsou záložky jednotlivých editorů. Dostupný je editor `Kód`, kde je možno textově editovat programy pro robota, a `Bloky`, což je editor v režimu pouze pro čtení a zobrazuje textový přepis obsahu blokového editoru. Kliknutím na záložku je okno přepnuto do daného editoru. Na levém okraji této lišty se nachází menu funkcí přiřazených k textovému editoru. Menu obsahuje následující položky:

- Vytvoř bloky – Pokud je uživatel v textovém editoru kódu, může označit část napsaného kódu. Po kliknutí na tuto položku je kód vygenerován v blokové podobě v blokovém editoru. Ze zobrazení přepisu blokového editoru nelze generovat bloky.
- Vymaž breakpointy – V aktuálně zobrazeném editoru vymaže všechny zadané breakpointy.
- Napovídání slov – Povoluje nebo zakazuje napovídání během psaní kódu. Obsahuje vizualizaci, zda-li je tato funkce povolena či zakázána.
- Automatické odsazování – Povoluje nebo zakazuje funkci automatického odsazování, které pomáhá k vizuálně přehlednému kódu. Je znázorněno pomocí zaškrtávacího pole zda-li je funkce aktivní.
- Ukazuj pozici – Povoluje nebo zakazuje znázorňování aktuální pozice interpretu v kódu pomocí kurzoru. Pokud je aktivní, při běhu interpretace je aktuálně zpracovávaný příkaz znázorněn kurzorem. Zaškrtávacím polem je znázorněno, jestli je funkce aktivní nebo ne.

Samotný textový editor poté nabízí již zmíněnou funkcionalitu obarvování kódu a základních klávesových zkratek pro editaci kódu, jako jsou funkce zpět pomocí `ctrl+z`, znovu pomocí `ctrl+shift+z` nebo vyhledej pomocí `ctrl+f`.

5.7 Postup práce

Práce je rozdělena do průběhu jednoho ročníku. Nejprve bylo zapotřebí vytvořit reprezentaci místnosti a naprogramovat základní chování robota v místnosti společně s objekty jako jsou cihla a značka. Pro tyto účely byla využita knihovna THREE.js, o které se hovořilo v kapitole 4.3. Díky této knihovně se podařilo vytvořit prozatímní interaktivní 3D rozhraní, kde nebyly dostupné finální 3D modely. Místo nich byly použity pouze zástupné jednoduché objekty, které jsou knihovnou nabízeny.

Následně bylo přidáno textové rozhraní a knihovna ACE popsaná v kapitole 4.3. Byl vytvořen jednoduchý interpret bez kontroly pro jazyk Karel s prvními příkazy a strukturami. Poté byl zprovozněn blokový editor s možností blokového programováni. Byla tak přidána knihovna Blockly, jež byla popsána v kapitole 4.3. Do ní byly nejprve vytvořeny

bloky pomocí dostupného nástroje od vývojářů knihovny a nakonfigurován generátor textového kódu pro jazyk Karel. Vygenerovaný kód je možné vykonávat již hotovým interpretem kódu.

Dále byl vývoj zaměřen tak, aby projekt poskytoval stejnou funkcionalitu jako předloha kvůli zpětné kompatibilitě. Byly doplněny všechny instrukce dostupné v předloze, včetně příkazu vydávajícího zvuk a příkazů měnících rychlost robota při vykonávání instrukcí. Rozměry místnosti lze měnit a byla přidána možnost podobná sloupu, která umožňuje odebírat políčka místnosti v editačním režimu, čímž je možné vytvořit místnosti různých tvarů.

Následovalo zprovoznění ukládání stavu aplikace. Lze uložit nejen textový zápis programu, ale také blokový zápis programu a aktuální stav místnosti. V této implementaci je možné volit, co z těchto položek má být uloženo. Soubor s uloženými hodnotami se stáhne uživateli do zařízení a obsahuje řetězec popisující formát JSON. Při načítání probíhá kontrola, zda-li soubor odpovídá formátem a strukturou, a poté je načten. Zároveň byl vytvořen převod textové podoby zápisu programů jazyka Karel na blokovou formu. Tato funkcionalita je potřeba k vytváření a načítání uložených stavů aplikace, kdy je obsah blokového editoru uložen v textové podobě, tedy ukládání blokového rozhraní využívá přepisu na textové a poté zpět z textového na blokové.

Byla přidána také možnost jednoduchého krokování, kdy lze program spustit v režimu, kdy uživatel potvrzuje každý krok programu. Přidány byly i „Breakpointy“ v kódu, ovšem zatím byly implementované pouze graficky a aplikace na ně prozatím nereagovala. Přidáno bylo také přepínání mezi českým a anglickým jazykem, které ovlivňuje blokové rozhraní a textové rozhraní, přičemž jsou kompletně změněny klíčová slova programovacího jazyka do jiného světového jazyka. Není realizován překlad kódu mezi světovými jazyky v textovém rozhraní, avšak blokové rozhraní se přizpůsobí.

Před koncem zkouškového období prvního semestru, kdy byla práce zpracovávána, byl vytvořen první návrh uživatelského rozhraní pro aplikaci, kde byly přidány knihovny Split.js a jQuery UI popsané v sekci 4.3. Zároveň byl do místnosti přidán model robota Karla, který byl autorem vymodelován v programu Blender, a do místnosti načten pomocí GLTFLoaderu poskytnutého knihovnou Three.js.

Dalším přídavkem v druhém semestru vypracovávání projektu bylo vytvoření LL1 kontroly popsané v kapitole 5.4 a rozšíření jazyka o proměnné a matematické výrazy. Pro ně bylo potřeba vytvořit podporu v kontrole, interpretaci, ale i vytvořit nové bloky pro blokový editor kódu. Byla snaha i o implementaci neustálé kontroly kódu, kterou podporuje knihovna ACE, ovšem díky horší dokumentaci této knihovny se nepodařilo tuto funkcionalitu zařídit. Současně pokračovala práce na uživatelském rozhraní. Byl vytvořen prostor rychlého přístupu s jeho obsahem, ovládací prvky textového editoru a zpřístupněny funkce prostředí pro textový editor.

Posledním krokem bylo vytvoření informačního systému, příkladů pro prostředí a zpřístupnění ukládání a načítání ze serveru.

5.8 Demonstrace

V této kapitole bude demonstrováno několik příkladů v nové verzi prostředí jazyka Karel navrženého touto prací.

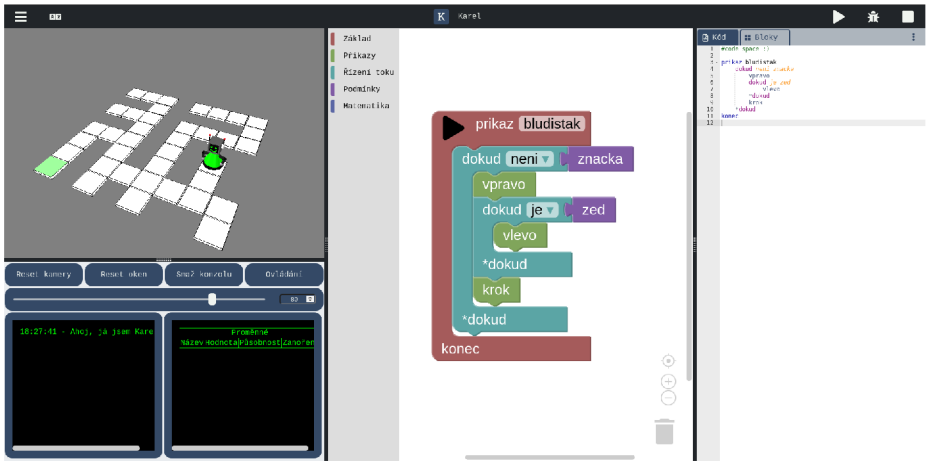
Bludiště

Již několikrát byl použit příklad pro jazyk Karel, kde je úkolem průchod bludištěm na danou pozici. Příklad je detailněji popsaný v kapitole 2.2 včetně algoritmu, který tuto úlohu řeší. Na obrázku 5.5 je znázorněno řešení vytvořené v novém prostředí včetně znázornění bludiště v místnosti a blokového i textového zápisu algoritmu. Řešení je možné vyzkoušet i na tomto odkazu. Textový zápis algoritmu je velmi podobný zápisu v Karlovi pro MS-DOS, jelikož byl dán důraz na kompatibilitu. Oba dva kódy vedle sebe jsou znázorněny níže:

```
# zápis algoritmu v~nové verzi
prikaz bludistak
    dokud neni znacka
        vpravo
        dokud je zed
            vlevo
        *dokud
        krok
    *dokud
konec
```

```
# zápis algoritmu ve verzi na MS-DOS
Příkaz BLUDISTE
začátek
    dokud NENÍ ZNAČKA dělej
        VPRAVO-VBOK
        dokud JE ZEĎ dělej
            VLEVO-VBOK
        *dokud
        KROK
    *dokud
konec
```

Jak je na příkladech vidět, zápis v nové verzi je funkcionálně identický, ovšem kratší, což by mělo uživatelům zpříjemnit práci při vytváření svých kódů. Další rozdíl je v reprezentaci místnosti. Díky tomu, že je v nové verzi možné odebírat políčka z místnosti, je na celý prostor vidět lépe, než kdyby byly implementovány sloupy jako v MS-DOS verzi. Bludiště v MS-DOS verzi je znázorněno na obrázku 2.4.



Obrázek 5.5: Příklad bludiště v nové verzi prostředí robota Karla 3D

Střed místnosti - rekurze

Dalším demonstračním příkladem pro prostředí robota je nalezení středu místnosti. Verze s rekurzí funguje i v prostředí na MS-DOS. Aby byl příklad zřetelnější, bude pro jeho vykonávání nastavena místnost o rozměrech devět × devět, která má středové políčko. Pro vyřešení se využije funkce, která je schopná najít střed zdi, tedy jednoho rozměru místnosti. Ta nejprve udělá dva kroky ve směru zdi a poté se zeptá, zda-li je před robotem zeď, pokud ano, otočí se dvakrát vpravo, pokud ne, je funkce rekurzivně opět spuštěna. Zakončení funkce provede jeden krok, tedy při vynořování z rekurzí se udělá polovina kroků oproti zanořování. Tato funkce se nejprve spustí pro jednu zeď a poté pro druhou. Výsledek je znázorněný na obrázku 5.6 a lze si jej v aplikaci otevřít na tomto odkazu[2]. Kód zapsaný jak v nové, tak staré verzi je k nahlédnutí níže.

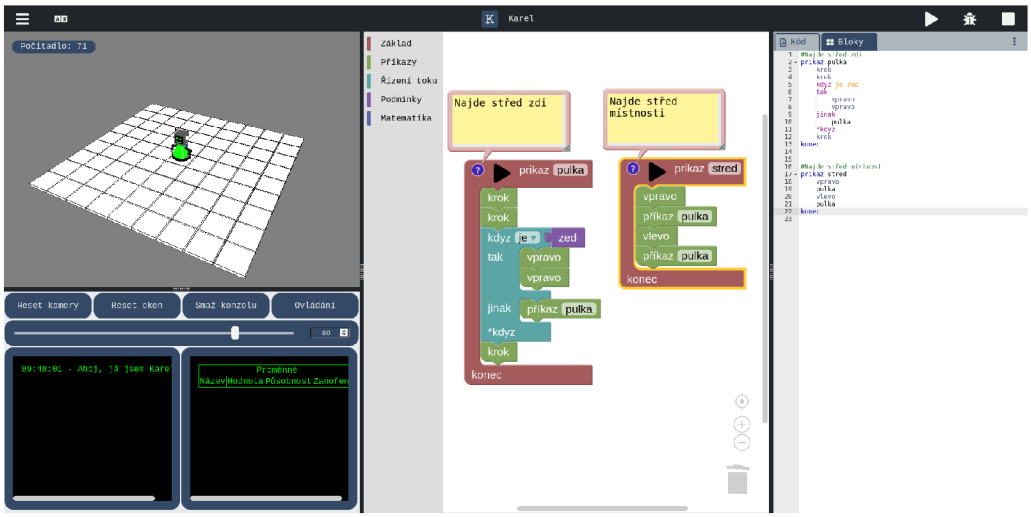
```
# zápis algoritmu v~nové verzi
prikaz pulka
    krok
    krok
    kdyz je zed
    tak
        vpravo
        vpravo
    jinak
        pulka
    *kdyz
    krok
konec

prikaz stred
    vpravo
    pulka
    vlevo
    pulka
konec
```

```
# zápis algoritmu ve verzi na MS-DOS
Příkaz PULKA
začátek
    KROK
    KROK
    když JE ZEĎ
        tak VLEVO-VBOK
            VLEVO_VBOK
        jinak PULKA
    *když
    KROK
konec

Příkaz STRED
začátek
    VLEVO-VBOK
    VLEVO-VBOK
    PULKA
    VLEVO-VBOK
    PULKA
konec
```

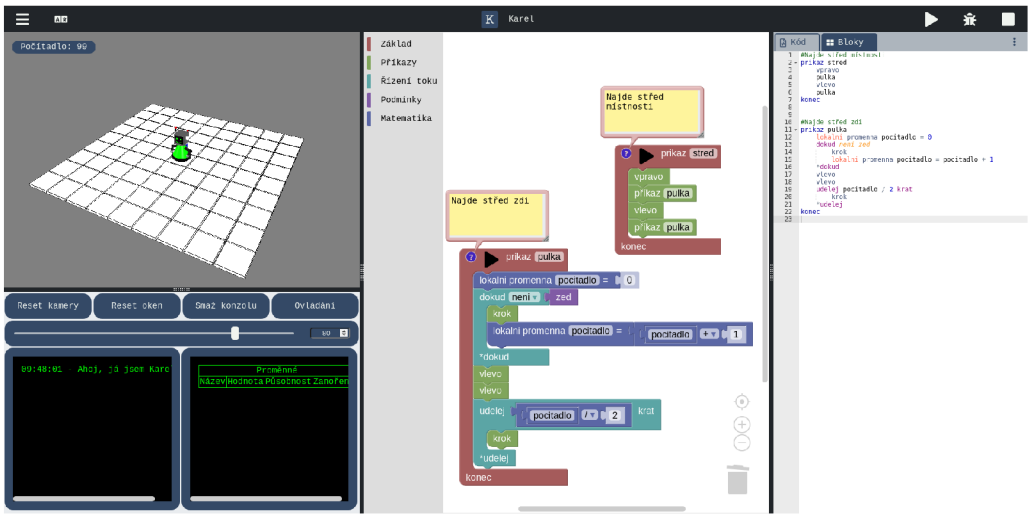
[2] http://karelrobot.cz/karel.html?StredMistnostiRekurze



Obrázek 5.6: Příklad nalezení středu pomocí rekurze v novém prostředí robota Karla 3D

Střed místnosti - proměnné

Díky rozšíření o proměnné je možné předchozí úlohu řešit i jiným způsobem bez rekurzivního zanořování. Tento algoritmus hledání středu zdi vytváří lokální proměnnou, kterou s každým krokem inkrementuje, dokud nenarazí na zeď. Poté se robot otočí a udělá přesně polovinu kroků zpět. Pro nalezení středu místnosti se hledání poloviny zdi spustí ještě pro druhý rozměr. Výsledek je znázorněn na obrázku 5.7 a lze si jej v aplikaci otevřít na tomto odkazu[3]. Při spouštění algoritmu jsou v tabulce proměnných vypisovány aktuální informace o proměnné `pocitadlo`. Zápis algoritmu je k nahlédnutí níže. Nyní již není přiložená k porovnání verze zápisu z MS-DOS, neboť v ní nelze tento algoritmus použít.



Obrázek 5.7: Příklad nalezení středu pomocí proměnných v novém prostředí robota Karla 3D

[3] http://karelrobot.cz/karel.html?StredMistnosi

```
#Najde střed místnosti
prikaz stred
    vpravo
    pulka
    vlevo
    pulka
konec

#Najde střed zdi
prikaz pulka
    lokalni promenna pocitadlo = 0
    dokud neni zed
        krok
        lokalni promenna pocitadlo = pocitadlo + 1
    *dokud
    vlevo
    vlevo
    udelej pocitadlo / 2 krat
        krok
    *udelej
konec
```

Toto byl přehled různých příkladů pokrývajících různé prvky, které prostředí nabízí. Další příklady je možné nalézt v systému aplikace na hlavní straně v sekci příklady[4].

[4] http://karelrobot.cz/#karelExercises

Kapitola 6

Testování

Po implementaci popsaných funkcí a uživatelského prostředí byla aplikace otestována. Pro testování vnitřní části byla využita sada příkladů, pomocí které se určovalo, zda-li robot správně reaguje na zapsaný algoritmus a všechny ostatní funkcionality pracují, jak mají. Následovalo testování uživatelského rozhraní, kdy byl k počítači s otevřenou aplikací posazen uživatel zapadající do cílové skupiny uživatelů a byla mu dána sada úkolů, které měl v aplikaci a systému splnit. Úkoly a průběh experimentu je sepsán níže.

1. Přihlášení do systému – Prvním úkolem pro uživatele bylo přihlášení do systému pomocí předem vytvořeného účtu. Uživatel ihned v systému nalezl požadovaný přihlašovací formulář a bez problémů se přihlásil.

2. Nalezení příkladů – Následoval úkol, ve kterém měl uživatel nalézt knihovnu příkladů v systému. Uživatel otevřel menu na levé straně navigačního panelu a přes položku „Příklady“ se přesunul ke knihovně.

3. Vypracování prvních třech příkladů z první kapitoly příkladů – Uživatel dále dostal za úkol vypracovat několik příkladů. V prvním příkladu měl problém vytvořit zadaný příkaz, neboť neměl vůbec žádné instrukce k aplikaci. Po malé nápovědě byl schopný příkaz vytvořit. Uživatel byl schopný robota ovládat v přímém režimu, kde využil ovládání skrze tlačítka na obrazovce. Byl schopný příkazy spouštět i zastavovat. Problém nastal, když se chtěl vrátit do systému. Pro návrat nakonec použil tlačítko „zpět“ ve webovém prohlížeči. Po druhém úkolu již použil pro návrat do systému tlačítko uprostřed navigačního panelu, které je k tomu určené. Uživateli se povedlo s malou dopomocí dokončit všechny tři příklady.

4. Ukládání a načítání – Dále uživatel dostal za úkol uložit stav aplikace do počítače, změnit stav aplikace a uložený stav opět načíst. Při ukládání neměl uživatel problém s nalezením správného nástroje a podařilo se mu vytvořit kompletní uložení aplikace. Při načítání měl uživatel problém s nalezením uloženého souboru v úložišti počítače, nicméně po radě byl schopný stav načíst.

5. Spouštění z textového rozhraní – Uživatel celou dobu k programování robota využíval blokového rozhraní. Proto dostal doplňující úkol, ve kterém načetl vyřešený příklad a měl spustit řešení. S tímto úkolem uživatel neměl problém.

Testování bylo provedeno na třech uživatelích různého věku a s rozdílnými schopnostmi práce s počítačem. Z testování je vidět, že uživatelské rozhraní by bylo vhodné ještě nějak upravit. Hlavně tlačítko systému je pro uživatele hůře objevitelné. Také se ukazuje, že pro pochopení jazyka Karel je třeba buď výkladu učitele, nebo přečtení si návodů, které jsou dostupné na hlavní stránce systému. Typický uživatel ale nečte tyto texty, proto by bylo potřeba je udělat přitažlivějšími, například přidáním obrázků, videí a podobně. Dále také při testování nastal problém při přihlašování, kdy selhalo připojení na web. Konzola není v základní pozici webu viditelná, což znamená, že se k uživateli nedostávají důležité informace o aplikaci. Po načtení byli někteří uživatelé zmatení z toho, že jsou bloky zabalené, a bylo potřeba jim ukázat, jak je rozbalit. Zbylé problémy jsou způsobeny nižší zběhlostí uživatelů při obecné práci s počítačem, což by vyřešil dohled učitele nebo vhodně vytvořený návod.

Bohužel testování nebylo věnováno tolik času, jak bylo původně plánováno. Místo toho byla provedena implementace sytému okolo aplikace. Nestihlo se tak otestovat na větším vzorku relevantních uživatelů, a proto je testování zařazeno do budoucího plánu projektu.

Kapitola 7

Závěr

Cílem práce je prozkoumat existující pedagogické nástroje pro vyučování programování a navrhnout vlastní novou verzi prostředí podporujícího jazyk Karel. Díky tomu, že to není můj první pokus o vytvoření takového prostředí, mohu porovnat své schopnosti při předchozím a tomto novějším pokusu. Myslím, že je při porovnání vidět, že jsem se mnohé za tu dobu naučil a i při realizaci aplikace jsem se snažil využít široké spektrum znalostí, které nám byly při studiu předávány. Vytvořená aplikace plní velkou část vizí, se kterými jsem do projektu vstupoval a poskytuje funkční a dle mého příjemné prostředí pro pedagogický jazyk Karel. Prostředí je obohaceno o nové technologie, které jsou nyní standardem v podobných jiných prostředích, která tato práce shrnuje a prozkoumává.

Na druhou stranu se najde se i pár problémů, které během vývoje nastaly. První problém vznikl, když jsem se pokusil implementovat kontrolu kódu, která by probíhala přímo při psaní kódu, což je funkce nabízená knihovnou ACE. Bohužel jsem nebyl schopný nalézt dostatečnou dokumentaci této funkcionality. Na druhou stranu jsem díky této slepé cestě prozkoumal detailně knihovnu a povedlo se mi využít některé její další funkce. Původně bylo také plánováno testování výsledku na žácích ve škole při reálné výuce na mém bývalém gymnáziu. Bohužel současná situace a špatné časové rozložení práce způsobilo, že se toto testování nekonalo. Testování celkově bohužel nebylo realizováno tolik, kolik by si aplikace zasloužila, ovšem o to více času bylo věnováno vývoji. Myslím, že aplikace je díky tomu ve stavu, kdy by mohla být nasazena při výuce a věřím, že by mohla na některé škole pomoci při výuce programování. Díky kontaktu s mojí bývalou školou se mi podařilo sehnat několik příkladů a vytvořit z nich knihovnu, která může posloužit k inspiraci, jak s aplikací pracovat.

Aplikace dospěla do stavu vývoje, ve kterém je již použitelná při výuce, avšak stále jsou cesty, kterými by šla vylepšit. Hlavním a dle mého nejvíce obohacujícím směrem je rozšíření stávající knihovny příkladů, které by žákům poskytlo plynulé a hravé pochopení programování, například s využitím zdroje [14]. Co se týče vylepšení aplikace, je možné vytvořit další rozšíření, podobné implementovanému rozšíření o matematické výrazy a proměnné. Tématem takového rozšíření by mohla být podpora více vláken, podpora návratových hodnot příkazů, podpora parametrů příkazů, a podobně. Při jejich vytváření je ovšem nutné klást důraz na jednoduchost a pedagogickou věcnost. Systém je také možno dále rozšiřovat, neboť v aktuální implementaci má pouze základní funkcionalitu. Podpora propojování účtů, vytváření třídních skupin, ve kterých by mohl pedagog žákům zadávat úkoly a kontrolovat je, by prostředí dále obohatilo. Nakonec by bylo prospěšné otestovat jak funkční část, tak přívětivost uživatelského rozhraní aplikace, opravit chyby a přizpůsobit prostředí lépe potřebám uživatelů.

Literatura

[1] Ajax.org B. V.. *Advanced Code Editor* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://ace.c9.io/.

[2] Boháč, L. *Tvorba webového interpretu jazyka Karel*. Brno, CZ, 2011. Magisterská práce. Masarykova univerzita, Fakulta informatiky. Dostupné z: https://is.muni.cz/th/j86kf/.

[3] Cahill, N. *Split.js* [online]. 2021 [cit. 2019-04-27]. Dostupné z: https://split.js.org/.

[4] Dirksen, J. *Learning Three.js - the JavaScript 3D Library for WebGL*. 2. vyd. Packt, 2015. ISBN 9781784392215.

[5] Fraser, N., Neutron, Q., Spertus, E. a Friedman, M. *Blockly* [online]. 2021 [cit. 2019-04-27]. Dostupné z: https://developers.google.com/blockly.

[6] Jedlička, O. *Robot Karel: vývojové prostředí* [online]. 2006 [cit. 2021-03-30]. Dostupné z: http://karel.oldium.net/.

[7] Klíma, K. *Karel* [online]. 2007 [cit. 2021-01-09]. Dostupné z: http://karel.webz.cz/.

[8] Meduna, A. *Automata and Languages*. 1. vyd. Springer-Verlag London, 2000. ISBN 978-1-85233-074-3.

[9] Meduna, A. a Lukáš, R. *Formální jazyky a překladače – IFJ – Studijní opora* [online]. 2015 [cit. 2019-04-27]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=115409.

[10] Pattis, R. E. *Karel The Robot: A Gentle Introduction to the Art of Programming*. 1. vyd. Wiley, 1981. ISBN 978-0471089285.

[11] Resig, J. *JQuery* [online]. 2021 [cit. 2019-04-27]. Dostupné z: https://jquery.com/.

[12] Resnick, M. *Scratch* [online]. 2005 [cit. 2021-03-30]. Dostupné z: https://scratch.mit.edu.

[13] SGP Systems. *SGP Systems - Baltík* [online]. 1996 [cit. 2021-04-21]. Dostupné z: https://www.sgpsys.com/cz/Download_B3.asp.

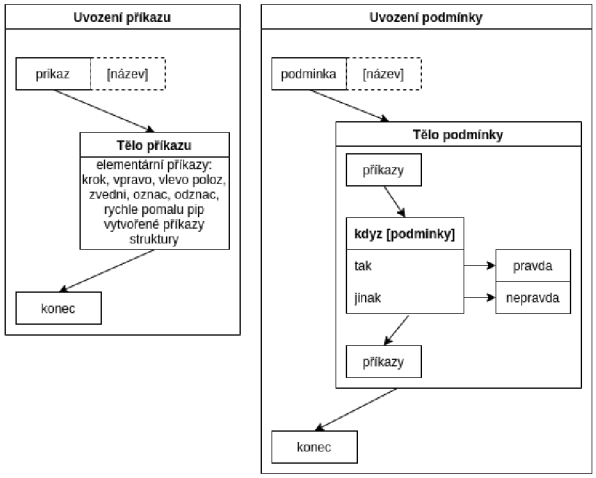
[14] Synovcová, M. *Martina si hraje s počítačem : 107 programů pro robota Karla : pro děti od 8 let*. 1. vyd. Albatros, 1989. ISBN 2-0957.146.

[15] Zakas, N. C. *Understanding ECMAScript 6*. 1. vyd. No Starch Press, 2016. ISBN 978-1593277574.

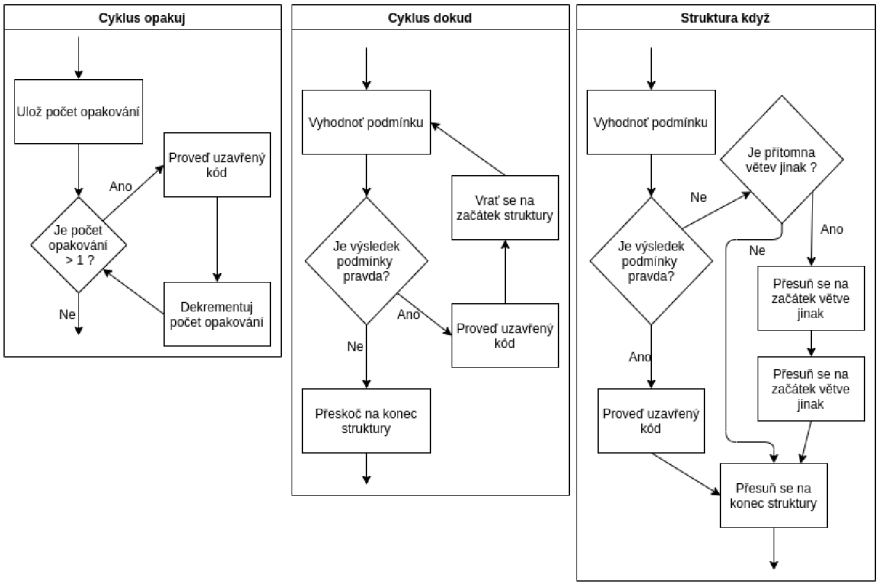
[16] Zaytsev, J. *ECMAScript 6 compatibility table* [online]. 2021 [cit. 2021-04-27]. Dostupné z: https://kangax.github.io/compat-table/es6/.

Příloha A

Přílohy



Obrázek A.1: Diagram syntaxe definice příkazů a podmínek v jazyce Karel



Obrázek A.2: Diagram syntaxe struktur v jazyce Karel

Neterminály na vrcholu zásobníku \ Terminály na vstupním řetězci	function-def	condition-def	end	user-command	user-condition	command	condition	do-start	times	do-end	while-start	while-end	if-start	then	else	if-end	cond-prefix	expression	output	global	local	[nic]
<start>	0	1																		13		9
<command-block>			9	3		2		8		9	7	9	4		9	9			12	14	15	
<end-if>															6	5						
<condition-core>					10		11											16				

Tabulka A.1: Tabulka pro aplikování pravidel pro LL1 kontrolu

č.p.	neterminál	prvky na rozgenerování neterminálu
0.	<start>	function-def, user-command, <command-block>, end
1.	<start>	condition-def, user-condition, <commadn-block>, output, <command-block>, output, <command-block>, end
2.	<command-block>	command, <command-block>
3.	<command-block>	user-command, <command-block>
4.	<command-block>	if-start, condition-prefix, <condition-core>, then, <command-block>, <end-if>
5.	<end-if>	if-end, <command-block>
6.	<end-if>	else, <command-block>, if-end, <command-block>
7.	<command-block>	while-start, condition-prefix, <condition-core>, <command-block>, while-end, <command-block>
8.	<command-block>	do-start, expression, times, <command-block> do-end, <command-block>
9.	<command-block>	
10.	<condition-core>	user-condition
11.	<condition-core>	condition
12.	<command-block>	global, expression, <start>
13.	<start>	
14.	<command-block>	global, expression, <command-block>
15.	<command-block>	local, expression, <command-block>
16.	<condition-core>	expression

Tabulka A.2: Pravidla pro LL1 kontrolu

Příloha B

Obsah přiloženého paměťového média

- `Paměťové médium`
  - `css` – stylové soubory aplikace
  - `doc` – veškeré dokumentační texty
    - `Excel@FIT` – materiály pro prezentaci na Excel@FIT
    - `summer` – text bakalářské práce
    - `winter` – materiály pro prezentaci k předmětu ITT
  - `favicon.ico` – ikona aplikace
  - `img` – použité obrázky v aplikaci
  - `index.php` – implementace serverové části informačního systému
  - `is` – data informačního systému
    - `clanky` – texty článků zobrazujících se v sekci blog
    - `data` – SQLite databáze systému
    - `examples` – knihovna příkladů pro jazyk Karel
    - `img` – obrázky informačního systému
    - `info.php` – testovací skript pro zjištění verze PHP na serveru
    - `ini.php` – nastavení databáze pro informační systém
    - `introductionTexts` – texty na úvodní straně informačního systéému
    - `lib` – knihovny pro informační systém
    - `sql.php` – skript pro operaci s databází
    - `vendor` – skripty pro vizualizace Markdown souborů
  - `js` – hlavní skripty aplikace
    - `ace` – soubory knihovny ACE
    - `blockly` – soubory knihovny Blockly
    - `jquery` – soubory knihovny jQuery
    - `main.js` – hlavní spouštěcí skript
    - `source` – komponenty aplikace
    - `split` – soubory knihovny split.js
    - `three` – soubory knihovny three.js
  - `karel.html` – popis prvků uživatelského rozhraní
  - `objects` – model robota Karla
  - `README.md` – návod
  - `sounds` – zvuky aplikace