Robot Karel pro výuku programování

Tomáš Lecián

Bakalářská práce
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Lecián**
Osobní číslo:	**A14119**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**prezenční**
Téma práce:	**Robot Karel pro výuku programování**
Téma anglicky:	**Karel the Robot for Introductory Programming Courses**

Zásady pro vypracování:

1. **Analyzujte stávající implementace programovacího prostředí Karel.**
2. **Prostudujte grafickou knihovnu SDL.**
3. **Navrhněte vlastní knihovnu s programovacím rozhraním robota Karla, která umožní studentům úvodních programovacích kurzů vytvářet programy pro Karla v běžném vývojovém prostředí v jazyce C.**
4. **Vytvořte demonstrační příklady, na kterých ukážete možnosti nové knihovny.**
5. **Vytvořte podrobnou dokumentaci všech funkcí knihovny ve formátu pro Doxygen.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **PATTIS, Richard E., Jim ROBERTS a Mark. STEHLIK. Karel the robot: a gentle introduction to the art of programming. 2nd ed. /. New York: Wiley, c1995. ISBN 0471597252.**
2. **PAZERA, Ernest. Focus on SDL. Cincinnati, Ohio: Premier Press, c2003. ISBN 1592000304.**
3. **PENTON, Ron. Data structures for game programmers. [Online-Ausg.]. Cincinnati, Ohio: Premier Press, 2003. ISBN 9781931841948.**
4. **MITCHELL, Shaun. SDL game development discover how to leverage the power of SDL 2.0 to create awesome games in C++. [Online-Ausg.]. Birmingham, UK: Packt Pub, 2013. ISBN 9781849696821.**
5. **KERNIGHAN, Brian W. a Dennis M. RITCHIE. The C programming language. 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, c1988. ISBN 0131103628.**

Vedoucí bakalářské práce: **Ing. Tomáš Dulík, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **24. února 2017**

Termín odevzdání bakalářské práce: **24. května 2017**

Ve Zlíně dne 24. února 2017

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl jsem seznámen s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 19. 5. 2017

.....
podpis diplomanta

ABSTRAKT

Cílem této bakalářské práce je vytvoření platformy programovacího jazyka Karel v jazyce C, která bude používána v úvodních kurzech programování na FAI UTB ve Zlíně. V první části je programovací jazyk Karel představen a popsán. Je zde uveden přehled významných celosvětových a místních implementací jazyka Karel. Dále práce obsahuje stručný přehled využití multimediální knihovny SDL. V druhé části je popsána vytvořená Knihovna robota Karla a její programové rozhraní. Závěrem je uvedeno několik ukázkových příkladů využití Knihovny robota Karla.

Klíčová slova: Robot Karel, výuka programování, jazyk C, knihovna SDL

ABSTRACT

The aim of this bachelor thesis is to create a platform of programming language Karel in language C, which will be used in introductory programming courses at FAI UTB in Zlín. In the first part, Karel programming language is introduced and described. Then an overview of important worldwide and local implementations of Karel are described. The thesis also contains a brief overview of using the multimedia library SDL. The second part describes the Robot Karel library and programming interface which were created. Finally, some examples of the Robot Karel library are shown.

Keywords: Robot Karel, Teaching of Programming, Language C, Library SDL

Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Tomášovi Dulíkovi, Ph.D. za věnovaný čas a věcné připomínky. Děkuji své nejbližší rodině za čas, který mi pro psaní této práce věnovali.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

V dnešní společnosti se enormně zvyšují nároky na chápání informačních technologií a softwarového vybavení nejrůznějších zařízení. Není proto výjimkou, že se již na základních či středních školách studenti setkávají se základy některého z programovacích jazyků. Úplné základy se u těch nejmladších studentů často vyučují v tzv. malých programovacích jazycích. Tyto jazyky jsou určeny pouze pro výukové potřeby a uzpůsobeny tak, aby byly jednoduché a intuitivní. Nevýhodou těchto jazyků je, že s jejich pomocí nelze vytvořit program či aplikaci, která by byla v praxi provozuschopná. Mezi malé programovací jazyky patří i jazyk Karel od Richarda E. Pattise. Vývojová prostředí malých programovacích jazyků mnohdy neposkytují uživateli nástroje pro snazší psaní kódu, které jsou v běžných moderních vývojových prostředích standardem. Těmito nástroji je myšleno např. debugger zdrojového kódu, našeptávání textu, vizualizace hodnot proměnných apod.

Knihovna robota Karla, o které pojednává tato práce, vychází z již zmiňovaného jazyka Karel a vznikla pro zvýšení atraktivity výuky úvodních kurzů programování na FAI UTB ve Zlíně. Umožňuje vytvářet jednoduché programy v jazyce C, a to v libovolném vývojovém prostředí. Knihovna nabízí jednoduché příkazy, kterými lze ovládat robota Karla v programovém rozhraní knihovny. Student tak může ovládat robota, aniž by musel znát složité konstrukce jazyka. S přibývajícími znalostmi jazyka C může student s využitím základních konstrukcí jazyka vytvářet složitější programy.

Knihovna robota Karla není jen pro úplné začátečníky. Mohou ji využít i studenti znalí syntaxe jazyka C. Mají možnost řešit složitější úkoly a učit tak robota Karla (i sebe) komplikovanějším algoritmům. Na uvedených příkladech je demonstrována práce s Knihovnou robota Karla a její možné využití.

I. TEORETICKÁ ČÁST

1 PROGRAMOVACÍ JAZYK KAREL

Karel je programovací jazyk, který byl vytvořen za účelem pochopení základů programování a osvojení si základních myšlenek v jednoduchém stejnojmenném programovacím prostředí.

Programovací jazyk Karel využívá podobných principů a konstrukcí jako jazyk Pascal. Umožňuje strukturované programování a princip rozkládání problému na jednodušší části. Předurčuje tedy k plynulému přechodu ke standardním programovacím jazykům, jako je Pascal, C a další.

1.1 Počátky programovacího jazyka Karel

S myšlenkou jednoduchého programovacího jazyka a prostředí bez složitostí přišel v roce 1970 student postgraduálního studia Stanfordské univerzity ve Spojených státech Richard E. Pattis. Navrhl programovací jazyk a prostředí, ve kterém studenti učí robota novým příkazům a učí jej řešit nejrůznější problémy.

Robota pojmenoval na počest českého spisovatele Karla Čapka, autora divadelní hry R. U. R. V tomto dramatu jako první na světě použil Čapek slovo robot [1] [3]. Jak později Čapek uvedl v článku pro etymologický slovník oxfordské angličtiny, nebyl to on, kdo s myšlenkou přišel. Slovo robot vymyslel jeho bratr, malíř a spisovatel Josef Čapek [2].

Robota Karla používal ve svých kurzech nejen Richard E. Pattis na Stanfordské univerzitě, který jej dodnes používá, ale i ostatní lektoři programování napříč celými Spojenými státy, zejména pak v úvodních hodinách.

1.2 Původní koncepce Karla

Původní koncepce jazyka Karla je popsána jeho zakladatelem Richardem E. Pattisem v knize *„Karel The Robot: A Gentle Introduction to the Art of Programming“* [4] z roku 1981.

1.2.1 Karlův svět

Karel je malý robot, který žije v nevelkém jednoduchém světě. Karlův svět je definován vodorovnými ulicemi (streets) vedoucími ze západu na východ a svislými bulváry (avenues) vedoucími ze severu na jih. Průsečík jednotlivých ulic a bulvárů je nazýván roh (corner).

V Karlově světě se mohou nacházet jen dva objekty – tzv. bzučák (beeper), což je plastový kužel, který vydává stálý pisklavý zvuk, a zeď (wall), přes kterou Karel nemůže projít [1].

1.2.2 Robot Karel

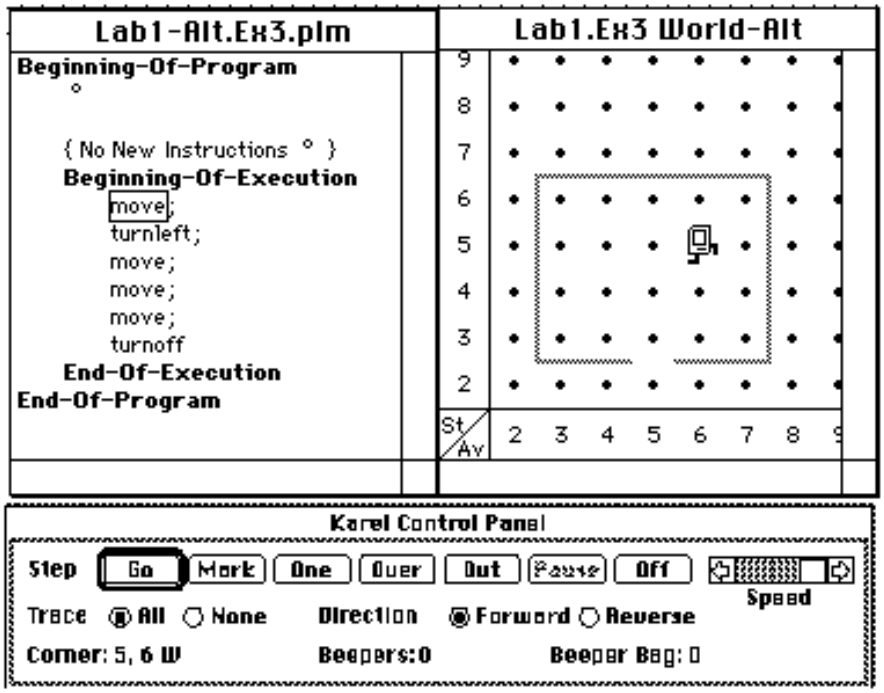
Robot Karel se může nacházet pouze na průsečících ulic a bulvárů, tj. na rozích, a může být natočen do jedné ze čtyř světových stran (sever, jih, východ, západ).

Karel zná 5 základních jednoduchých příkazů, díky kterým jej lze ovládat:

- move – robot Karel se posune o jeden blok vpřed ve směru, ve kterém je natočen,
- turnLeft – robot Karel se otočí o 90° vlevo,
- pickBeeper – robot Karel zvedne jeden bzučák na rohu, na kterém stojí, a vloží si jej do batohu,
- putBeeper – robot Karel vezme z batohu jeden beeper a položí jej na místo, na kterém právě stojí,
- turnOff – robot Karel se vypne.

Také může vyhodnocovat několik podmínek, které mu pomáhají získat informaci o tom, co se kolem něj nachází. Může testovat, zda je před ním či okolo něj volno a na kterou ze světových stran je natočen. Podmínky mu umožňují zjistit, jestli se na rohu, na kterém stojí, nachází bzučák a zda má nějaké bzučáky v batohu.

Pomocí těchto 5 základních příkazů a podmínek, které Karel umí vyhodnocovat, učí student Karla novým složitějším příkazům. Karel se díky nim může bezpečně pohybovat po svém světě a plnit úkoly, které mu student zadá [1]. Podoba prvního programovacího prostředí Karla je znázorněna na obrázku (Obr. 1).



Obr. 1. První programovací prostředí robota Karla [5].

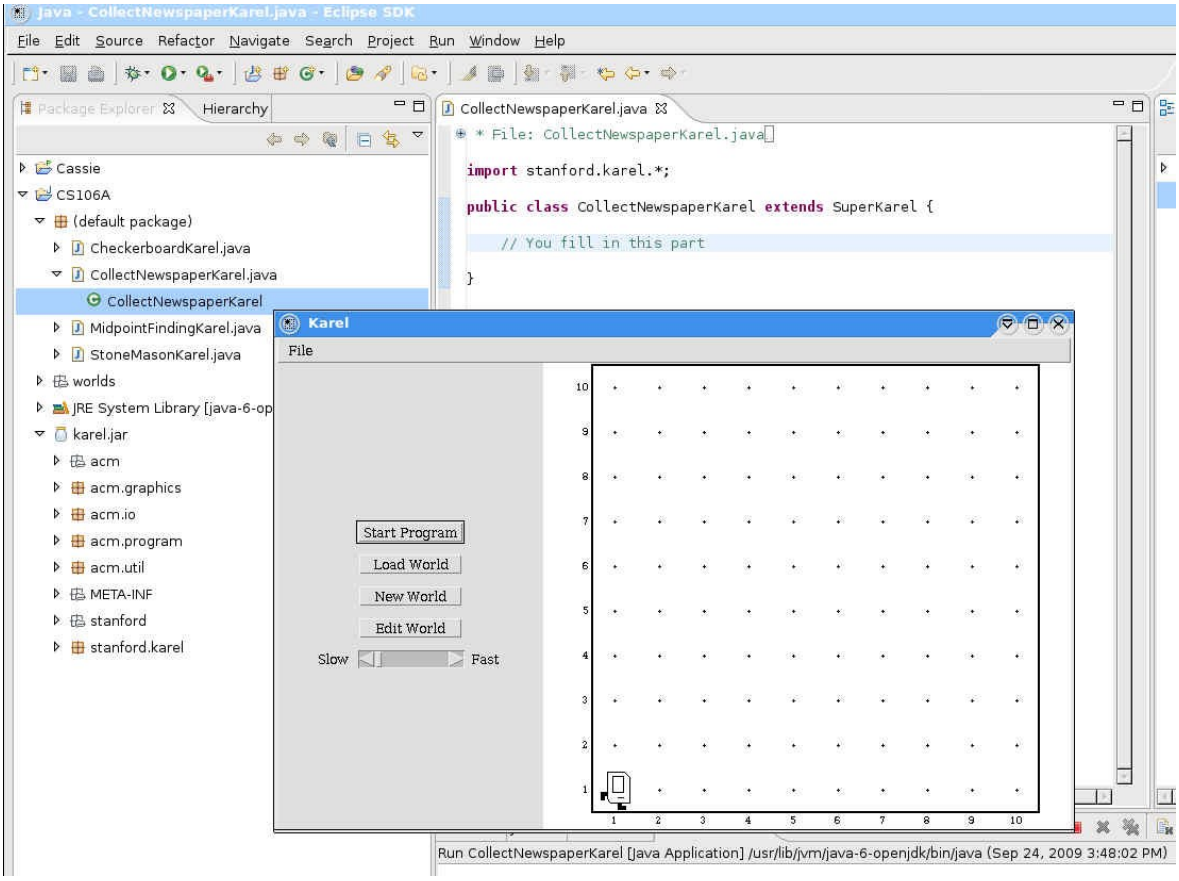
1.3 Vývoj a stávající implementace jazyka Karla

Jazyk Karel se stal inspirací mnoha lidem se zájmem o programování. Díky tomu vzniklo po celém světě několik modifikací jazyka robota Karla a jeho prostředí.

V 90. letech se začíná více prosazovat nové paradigma programování, tzv. objektové orientované programování (OOP). Jelikož původní koncepce Karla je typu strukturovaného programování, v roce 1996 Jozeph Bergin společně s Richardem E. Pattisem a dalšími autory vydávají knihu *„Karel++: A Gentle Introduction to the Art Of Object-oriented Programming“* [6]. V této knize představují objektově orientovanou koncepci jazyka Karla označenou Karel++. Tento jazyk vychází z původní koncepce jazyka Karla od Richarda E. Pattise, ale syntaxe je podobná programovacímu jazyku C++.

1.3.1 Robot Karel na Stanfordské univerzitě

S postupem času Stanfordská univerzita začala používat vývojové prostředí Eclipse. Jazyk Karel implementuje v podobě knihovny pro toto vývojové prostředí. Knihovnu je dovolené používat jen pro výukové účely v rámci Stanfordské univerzity [7]. Ukázku GUI knihovny ve vývojovém prostředí Eclipse můžeme vidět na obrázku (Obr. 2).

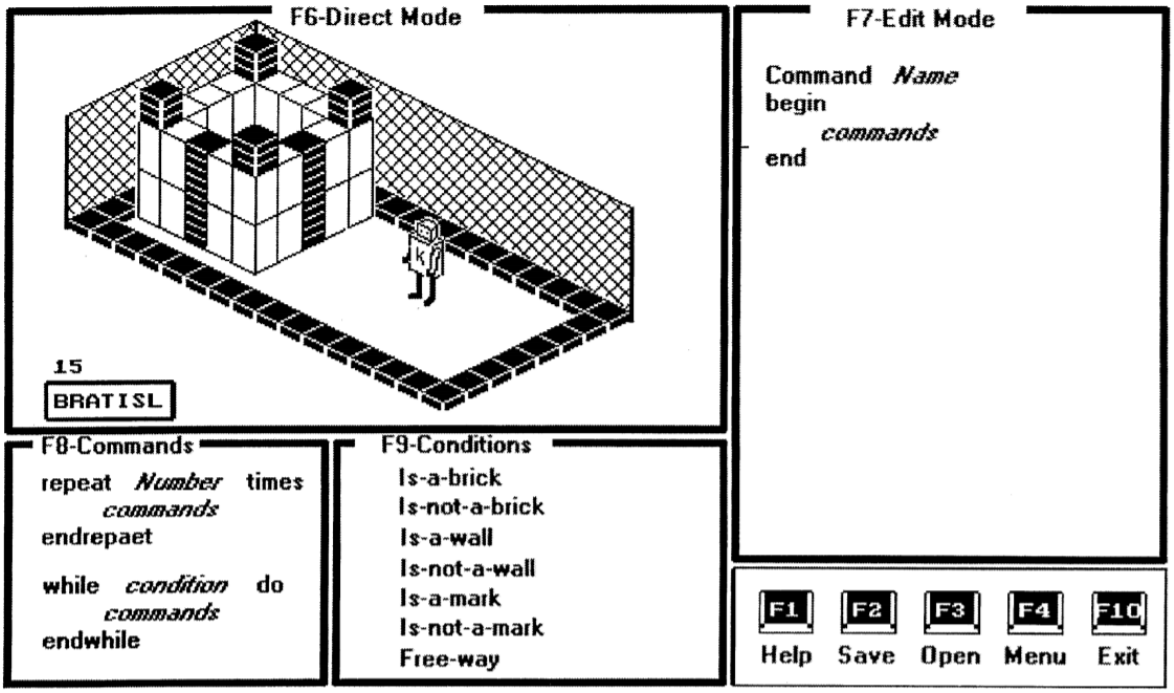


Obr. 2. Vývojové prostředí Eclipse s použitím knihovny Karel Stanfordské univerzity pro jazyk Java [15].

1.3.2 Vývoj robota Karla u nás

Karel 3D

Karel 3D vznikl zásluhou doc. Jozefa Hvoreckého 1992 v Bratislavě. Karel 3D je rozšířením původní myšlenky robota Karla a byl navrhnut pro studenty základních a středních škol. Ukázka programovacího prostředí Karla 3D je znázorněna na obrázku (Obr. 3).



Obr. 3. Programovací prostředí Karel 3D [5].

Prostředí, které je určeno pro systémy MS-DOS, je rozděleno do několika základních oken. V okně Přímého režimu se zobrazuje robot Karel ve svém světě, který se zde oproti koncepci Richarda E. Pattise pohybuje v trojrozměrném prostoru. Okno Editačního režimu umožňuje zadávání příkazů, jimiž se robot Karel ovládá, ale je jej možné ovládat i pomocí klávesnice. V oknech Příkazy a Podmínky se zobrazuje nápověda možných použitelných příkazů a podmínek.

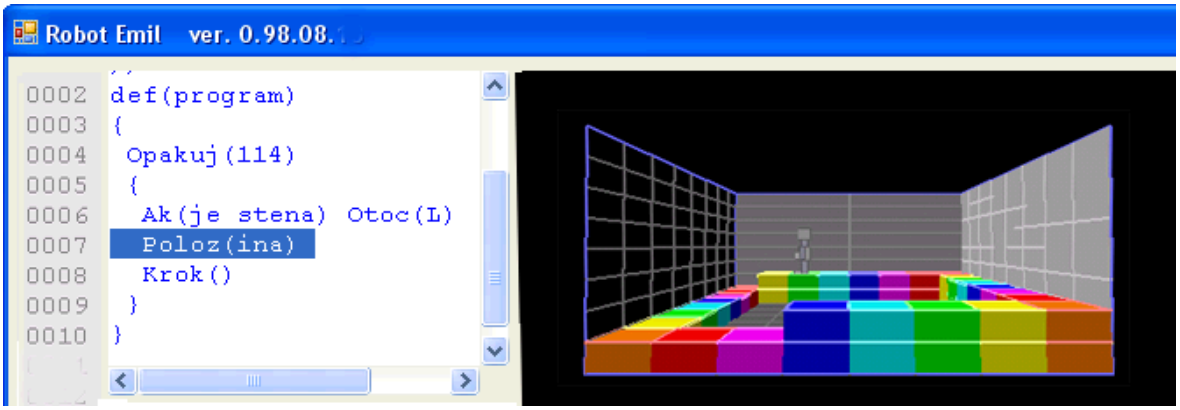
Karel ve svém světě umí pokládat nejen značky, ale také na sebe vrstvit kvádry a cihly, díky kterým vytváří libovolné objekty. Karel může na jednotlivých cihlách stát a tím se po nich pohybovat nahoru a dolů. Také má tu vlastnost, že pokud stojí na cihlách v nějaké výšce, může skočit dolů, aniž by se porouchal [5] [8].

Jozef Hvorecký ve spolupráci s Ľubou Gašparovičovou publikovali učebnici *„Kamaráti robota Karla“* [8], kde popisují, jak Karla ovládat.

Slovenský student Ľubomír Košút v rámci své diplomové práce v roce 1997 vytvořil verzi robota Karla 3D pro platformu Windows 3.11.

V roce 2008 Andrej Černík vylepšil stávající verzi robota Karla 3D. Umožnil robotovi pokládat cihly různých tvarů a barev. Značky může robot pokládat nejen na zem a na cihly, ale i na stěny. Na celý svět se může nahlížet pomocí kamery z různých směrů a vzdálenos-

tí. Vývojové prostředí bylo navrženo pro platformu Windows XP pod názvem Robot Emil 3D [9]. Ukázka programovacího prostředí Robot Emil je znázorněna na obrázku (Obr. 4).

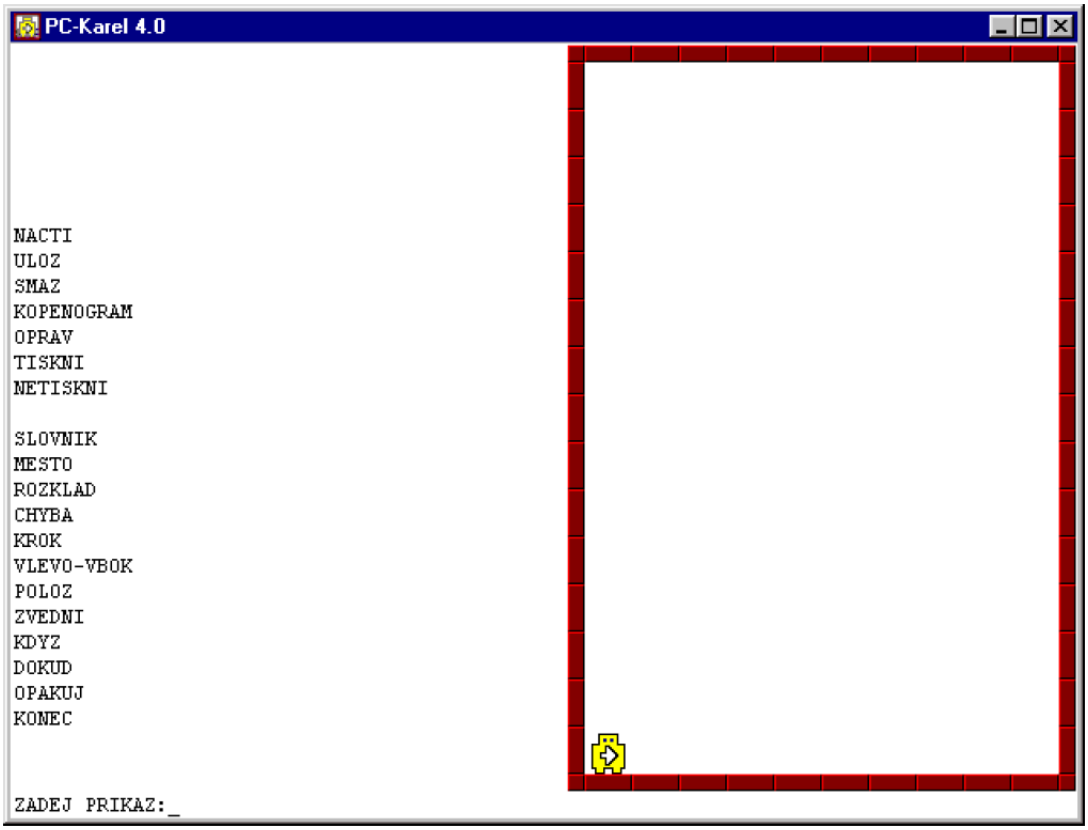


Obr. 4. Programovací prostředí Robot Emil [9].

PC-Karel

PC-Karel je interpret jazyka Karel [10] pro operační systémy MS-DOS a Windows, jehož autorem je Jiří Osoba. Robot Karel se pohybuje po čtvercové nebo obdélníkové síti nazývané dvoreček. Ve svém batůžku má nekonečně mnoho značek, které může pokládat. Všechny příkazy, které Karel umí, jsou uloženy ve slovníku pro pozdější použití.

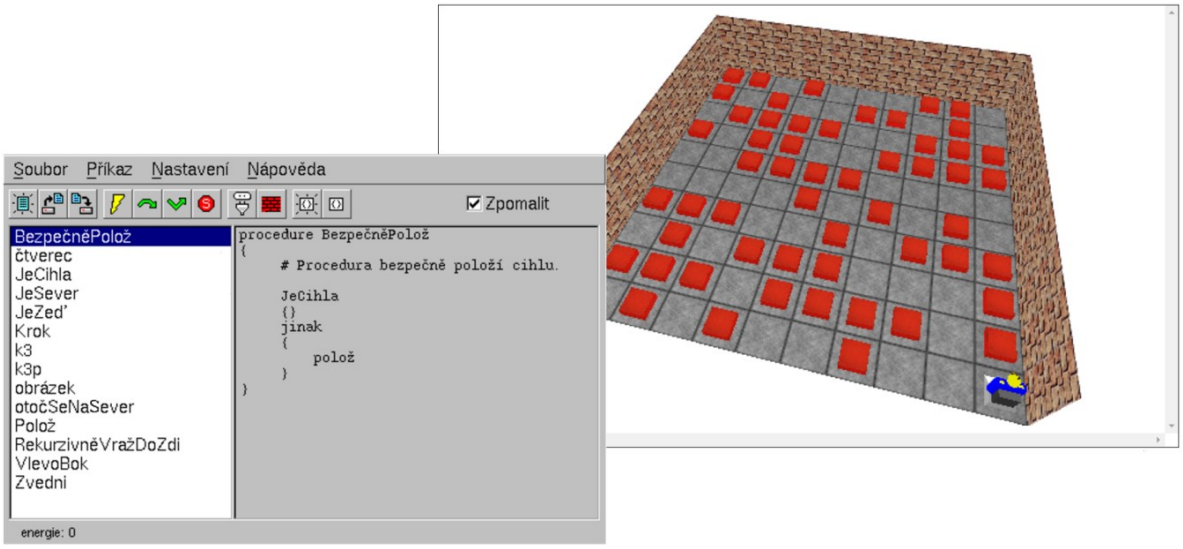
Dvojice Vejvoda, Rytíř vydala učebnici *„Programovací jazyk Karel“* [11], ve které popisují stručnou metodiku robota Karla pro výuku začátečníků jazyka PC-Karel. Ukázka programovacího prostředí PC-Karel ve Windows je na obrázku (Obr. 5).



Obr. 5. Programovací prostředí PC-Karel [11].

xKarel

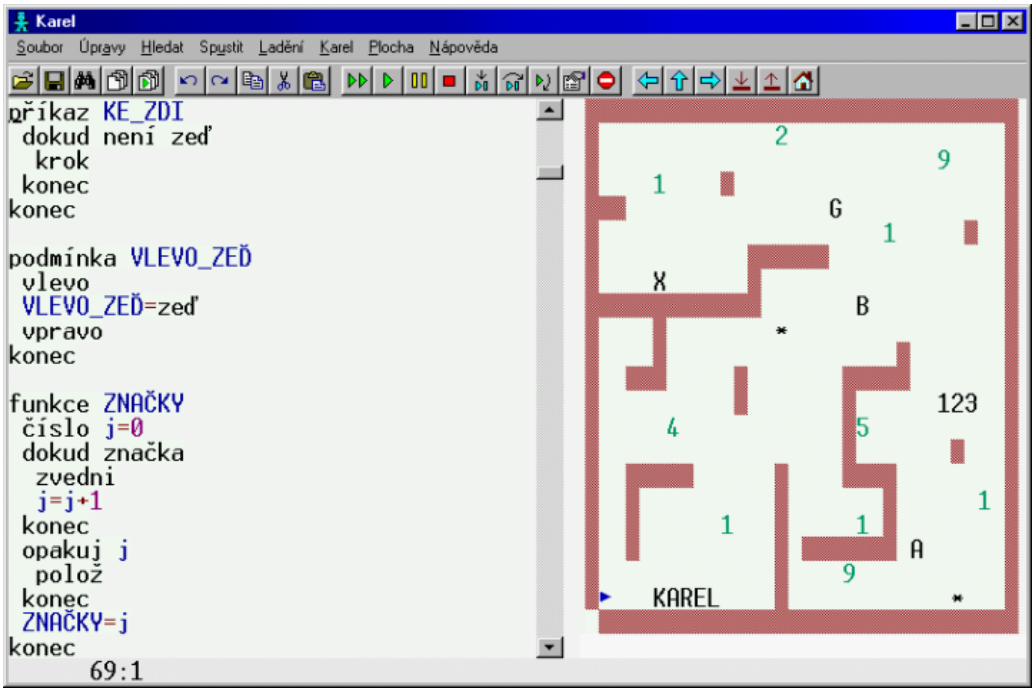
xKarel vznikl jako předmět seminární práce studentů Radima Dostála a Petra Abrahamczika na VŠB v Ostravě v roce 1997. xKarel je napsán v ANSI C++ s použitím knihovny FLTK. Je dostupných několik verzí xKarla, které umožňují spuštění na platformě Windows či Unixových systémech, a to v provedení 2D nebo 3D. Zdrojové kódy jednotlivých implementací je možné stáhnout z oficiálních stránek xKarel [12]. Ukázka programovacího prostředí xKarel je na obrázku (Obr. 6).



Obr. 6. Programovací prostředí xKarel [12].

Jazyk Karel Petra Laštovičky

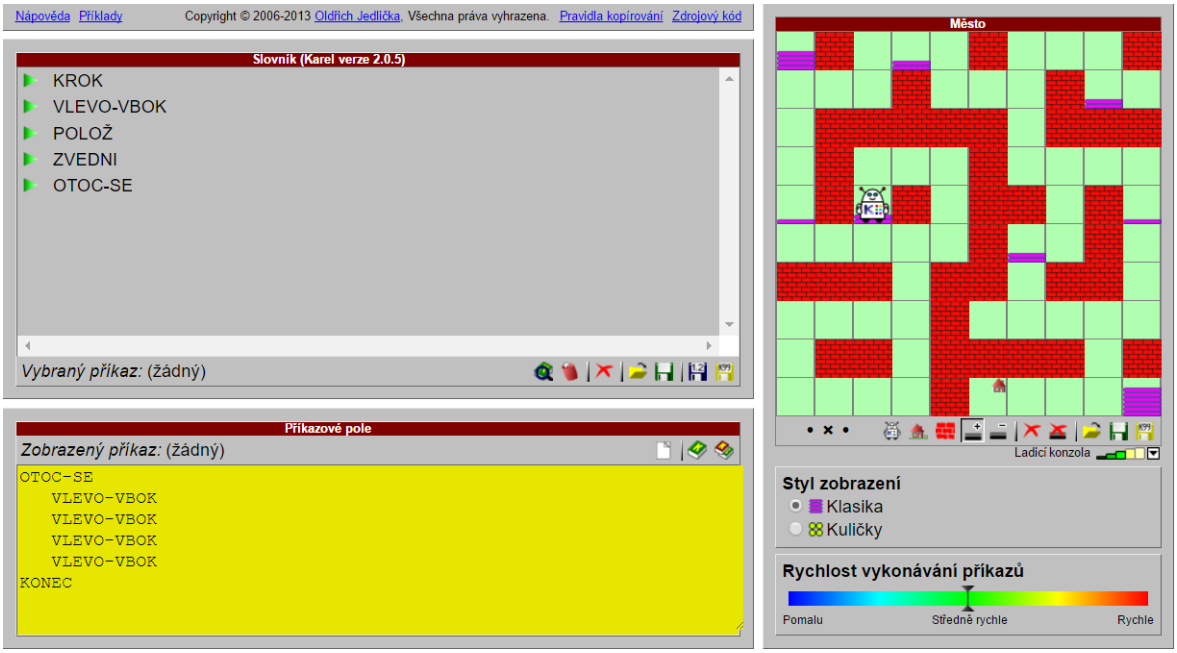
Petr Laštovička vytvořil programovací prostředí Karel v jazyce C++ pro platformy DOS i Windows. Uživatel při programování robota může využít nejen základních příkazů, které robot zná, ale i celočíselných proměnných, vícerozměrných polí, aritmetických a logických operací, procedur a funkcí s parametry. Na oficiálních webových stránkách Petra Laštovičky jsou dostupné nejen jednotlivé aplikace, ale i odkaz ke stažení zdrojových kódů [13]. Ukázka programovacího prostředí Karel od Petra Laštovičky je na obrázku (Obr. 7).



Obr. 7. Programovací prostředí Karel od Petra Laštovičky [16].

Jazyk Karel Oldřicha Jedličky

Oldřich Jedlička vytvořil v jazyce JavaScript programovací prostředí, které je přístupné online na internetu společně se zdrojovými kódy [14]. Robot se pohybuje po čtvercové síti s velikostí 10x10. Na jedno místo může položit maximálně 8 značek a má vlastní domeček. Uživatel ovládá robota 4 základními příkazy, které robot zná (KROK, VLEVO-VBOK, POLOZ, ZVEDNI), a základními programovacími podmínkami a cykly. Ukázka programovacího prostředí Karel od Oldřicha Jedličky je na obrázku (Obr. 8).



Obr. 8. Programovací prostředí Karel od Oldřicha Jedličky [14].

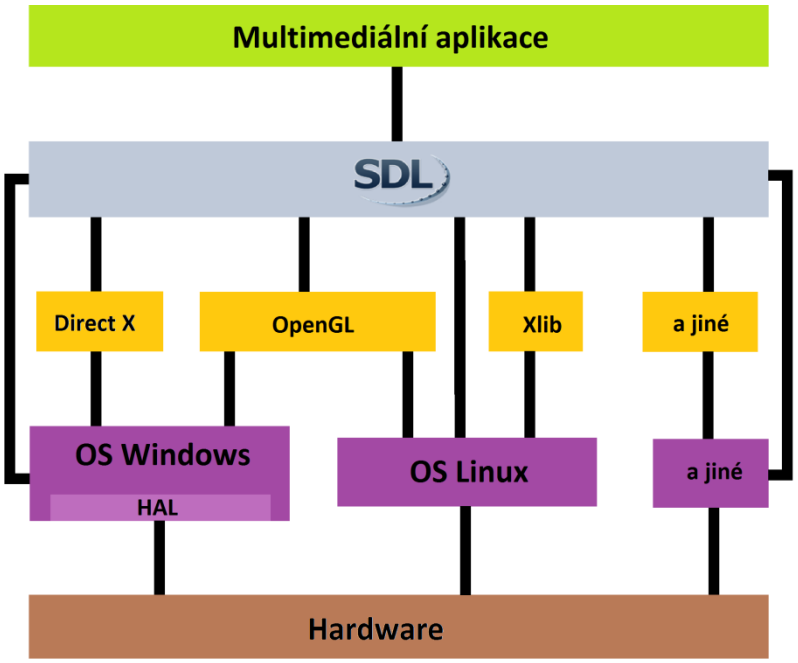
Další implementace jazyka Karel

Dalšími implementacemi robota Karla jsou např. robot Karel od Petra Plavjaníka, Karel od VS SOFT Computer servis a Karel 98 od Tomáše Holubce. Příklady implementací vytvořených v jazyce Java jsou KarelJ či Jarel. Příkladem v jazyce Python je pak pyKarel. Ondrej Krško vytvořil v rámci své diplomové práce na univerzitě Komenského v Bratislavě implementaci jazyka Karel pro platformu Microsoft .NET. Libor Boháč na Masarykově univerzitě v Brně pak vytvořil webového interpreta jazyka Karel.

2 SDL

Pro implementaci Knihovny robota Karla v jazyce C bylo potřeba vybrat grafickou knihovnu pro jazyk C, která by byla jednoduchá, malá a pokud možno multiplatformní. Těmto požadavkům vyhovuje jediná knihovna – libSDL.

SDL (SimpleDirectMediaLayer) je multimediální a zároveň multiplatformní vývojová knihovna poskytující nízkoúrovňový přístup ke vstupně-výstupním zařízením, tj. audio, klávesnice, joystick a grafický hardware přístupný přes OpenGL a Direct3D. Jinými slovy knihovna SDL zapouzdřuje přístup k jednotlivým specifickým funkcím operačního systému, které aplikace využívají. Poskytuje tak jednotné stanovisko k těmto funkcím pro více operačních systémů.



Obr. 9. Základní abstraktní vrstvy při využití SDL knihoven.

Knihovnu vytvořil v roce 1998 Sam Lantinga v rámci práce pro Loki Software, která nyní knihovnu SDL vyvíjí. Knihovna SDL oficiálně podporuje operační systémy Windows, Mac OS, Linux, iOS a Android. Podporu pro další platformy lze nalézt ve zdrojových kódech knihovny [17].

Společnost Loki Entertainment Software distribuuje SDL jako software (API) s otevřeným kódem. Verze SDL 1.2 a starší jsou chráněny licencí GNU LGPL. Verze SDL 2.0 a novější jsou chráněny licencí Zlib. SDL 2.0 je vylepšením SDL 1.2 a poskytuje vyšší a obecnější podporu vstupně-výstupních možností. Zpětná kompatibilita ale není zachována. Knihovna

je vytvořena v jazyce C, ale má k dispozici vazby na mnoho různých programovacích jazyků (tzv. SDL Language Bindings). Je tedy možné knihovnu použít v různých skriptovacích či speciálních programovacích jazycích.

Knihovna SDL je v základu relativně malá a jednoduchá. Poskytuje podporu pro 2D grafiku, zvuk, přístup k souborům, obsluhu událostí, časování, vlákna a další. 3D grafika je v SDL řešena voláním funkcí OpenGL.

2.1 Doplňkové knihovny SDL

Ke standardní SDL knihovně, která obsahuje pouze základní funkce, je k dispozici několik samostatných rozšiřujících knihoven: [18]

- SDL_image – je knihovna poskytující funkce pro nahrávání obrázků jiných formátů než pouze BMP, podporuje formáty BMP, GIF, JPEG, LBM, PCX, PNG, PNM, TGA, TIFF, WEBP, XCF, XPM a XV,
- SDL_mixer a SDL_sound – jsou knihovny poskytující rozšířené funkce pro práci s audiem a podporují další knihovny pracující se zvukem,
- SDL_net – je rozšiřující knihovna poskytující zjednodušené řízení síťových připojení a přenosu dat,
- SDL_rtf – knihovna podporující vykreslování souborů RTF, poslední verze SDL_rtf knihovny je vydána pro knihovnu SDL 1.2,
- SDL_ttf – je knihovna umožňující vykreslovat TrueType fonty v aplikacích využívající SDL knihovnu.

II. PRAKTICKÁ ČÁST

3 KNIHOVNA ROBOTA KARLA

Knihovna robota Karla vytvořená v rámci bakalářské práce umožňuje v běžném vývojovém prostředí jazyka C vytvořit vizuální podobu města pro robota Karla. Robot se v tomto městě ovládá základními funkcemi knihovny, které zprostředkovávají pohyb a poskytují robotovi informace o dění kolem něj.

Uživatel má tak názorné prostředí a představu o tom, jak se má robot ve svém městě pohybovat a chovat, aby splnil zadaný úkol. V jazyce C (jinému jazyku robot nerozumí) student učí robota, jak má daný úkol řešit.

Začátečníkům knihovna poskytne názornější představu o tom, jak se liší myšlení v reálném světě a ve světě softwaru. Podle obtížnosti úkolu, který má robot řešit, student uplatňuje odpovídající obtížnost konstrukcí jazyka C.

3.1 Koncepce knihovny

3.1.1 Robot Karel a jeho ovládání

Robot Karel se může volně pohybovat po svém městě. Musí ale dávat pozor, aby nenarazil do zdí, kterými je jeho město ohraničeno, případně členěno. Na pozici, na níž aktuálně stojí, může umístit zelenou kostičku jako značku. Těchto kostiček může na sebe naskládat maximálně 6. Ve městě má Karel také svůj vlastní domeček.

Funkce knihovny sloužící k ovládání robota Karla uživatelem se nazývají příkazy. Karel zná čtveřici příkazů k vykonávání pohybu. Karel umí vyhodnocovat 10 podmínek, které umožňují uživateli získat informace o tom, co se kolem Karla nachází. Karel zná také dva příkazy, díky nimž zjistí, v jakém městě se bude pohybovat a jak dané město případně opustit.

Vytvoření města

Deklarace funkce *postavitMesto*: `bool postavitMesto(char *nazevMapyMesta);`

Účelem této funkce je inicializace a vytvoření celého města. Robot Karel musí provést tento příkaz před prvním provedením jakýchkoli příkazů, které zná. Při neprovedení tohoto příkazu Karel nezná svou pozici, neví, kde se právě nachází a jak vypadá jeho město. Příkazy proto nemohou být správně vykonány.

Vstupním parametrem této funkce je cesta k umístění souboru TXT, ve kterém je definována podoba města. Cesta může být uvedena absolutně, např. C:\\Users\\Student\\VaseJmeno\\MapaMesta1.txt, nebo relativně z výstupního adresáře *Debug* ve vašem projektu, např. MapyMesta\\MapaMesta1.txt. Vstupní parametr funkce je řetězec. Proto je nutné pro zapsání zpětného lomítka v cestě k souboru v jazyce C použít Escape sekvenci v podobě dvou obrácených lomítek.

Dojde-li ke správné inicializaci města, je návratovou hodnotou funkce *true*. V opačném případě hodnota *false*.

Správné ukončení programu

Deklarace funkce *zbouratMesto*: `void zbouratMesto();`

Funkce se stará o správné uvolnění veškeré paměti, kterou si alokovaly vnitřní funkce knihovny. Vnitřní funkce knihovny se starají o správné vykonávání příkazů a podmínek, kterými uživatel robota Karla ovládá (vnitřní funkce nejsou uvedeny v hlavičkovém souboru knihovny). Příkaz *zbourtatMesto* také vypisuje konečný počet dvou typů chyb, jenž nastaly při běhu programu.

Chyby označené jako *Karlovy nezdary* jsou součtem nesprávně vykonaných Karlových příkazů. Příkladem nesprávně vykonaných příkazů je pak nepoužití příkazu *postavitMesto* před prvním příkazem ovládající robota Karla. Za chybu se taktéž považuje použití příkazu *krok* s následným nárazem do zdi. Pokus o zvednutí značky, která se na daném místě nenachází, nebo položení více značek na sebe, než je maximálně možné (tj. více než 6 značek), je taktéž bráno jako chyba. Nenulový počet těchto chyb ukazuje na nesprávnost uživatelova algoritmu.

Interní chyby jsou takové, které vzniknou např. nesprávným načtením dat či textur ze souboru, chybným vykreslováním při spuštění programu, nedostatkem paměti pro alokaci apod.

Krok vpřed

Deklarace funkce *krok*: `bool krok();`

Pomocí příkazu *krok* se robot Karel posune o jedno políčko vpřed ve směru, ve kterém je natočen.

Pokud Karel provede krok správně, návratová hodnota funkce je *true*. Stojí-li Karel čelem ke zdi a udělá krok vpřed, tak do ní narazí. Návratová hodnota je v takovém případě *false*.

Otočení vlevo

Deklarace funkce *vlevoVbok*: `bool vlevoVbok();`

Příkazem *vlevoVbok* se Karel otočí o 90° vlevo od směru, ve kterém je původně natočen.

Dojde-li k chybnému vykreslení nového směru Karlova natočení, je návratová hodnota funkce *false*. Při správném natočení je návratová hodnota *true*.

Položení značky

Deklarace funkce *polozZnacku*: `bool polozZnacku();`

Karel může příkazem *polozZnacku* položit jednu značku na pozici, na které aktuálně stojí. Na jedno místo ve svém městě může položit maximálně 6 značek.

Funkce ve své návratové hodnotě vrací hodnotu *true*, pokud položení značky proběhlo správně. Při nesprávném překreslení aktuálního počtu značek či v případě, že se Karel pokusí na jedno místo položit více než 6 značek, je vrácena hodnota *false*.

Zvednutí značky

Deklarace funkce *zvedniZnacku*: `bool zvedniZnacku();`

Příkaz *zvedniZnacku* Karlovi umožňuje zvednout jednu značku z místa, na kterém aktuálně stojí.

Funkce vrací hodnotu *false*, jestliže se Karel pokouší zvednout značku, která se na aktuální pozici nenachází. Návratová hodnota *true* označuje úspěšné zvednutí značky.

Zeď před robotem

Deklarace funkce *jePredemnouZed*: `bool jePredemnouZed();`

Příkazem *jePredemnouZed* Karel testuje, zda se na políčku před ním nachází zeď, kterou může být jeho město rozčleněno, nebo ohraničeno.

Funkce *jePredemnouZed* vrací hodnotu *true* v případě, že se na políčku před Karlem nachází zeď. Hodnota *false* je vrácena funkcí v případě, že se před Karlem zeď nenachází, a může tak např. provést jeden bezpečný krok, bez toho aniž by narazil do zdi.

Zeď nalevo od robota

Deklarace funkce *jeVlevoZed*: `bool jeVlevoZed();`

Karel si příkazem *jeVlevoZed* může ověřit, zda se vlevo od směru, ve kterém je natočen, nachází zeď.

Funkce vrací hodnotu *true*, detekuje-li robot Karel, že se vlevo od něj nachází zeď. V opačném případě je vrácena hodnota *false*.

Zeď napravo od robota

Deklarace funkce *jeVpravoZed*: `bool jeVpravoZed();`

Příkazem *jeVpravoZed* Karel ověřuje, zda se vpravo od směru, ve kterém je natočen, nachází zeď.

Funkce vrací hodnotu *true*, nachází-li se zeď napravo od robota Karla, jinak je vrácena hodnota *false*.

Detekce značky

Deklarace funkce *jeZdeZnacka*: `bool jeZdeZnacka();`

Tímto příkazem Karel zjistí, zda se na pozici, na které aktuálně stojí, nachází nějaká značka.

Návratová hodnota funkce je *true* v případě, že se Karel nachází na pozici, kde je nějaká značka. Jinak je vrácena hodnota *false*.

Zjištění počtu značek

Deklarace funkce *pocetZnacek*: `int pocetZnacek();`

Tento příkaz je rozšířením pro příkaz *jeZdeZnacka* a umožňuje Karlovi zjistit přesný počet značek nacházející se na místě, na kterém stojí.

Návratovou hodnotou je celé číslo označující počet značek.

Detekce domečku

Deklarace funkce *jeZdeDomecek*: `bool jeZdeDomecek();`

Karel příkazem *jeZdeDomecek* má možnost detekovat, zda se nachází ve svém domečku či nikoli.

Funkce *jeZdeDomecek* vrací hodnotu *true*, nachází-li se robot Karel ve svém domečku. Není-li Karel ve svém domečku, je vrácena hodnota *false*.

Zjištění směru natočení

Deklarace možných funkcí: `bool jeOtocenNaSever(); bool jeOtocenNaJih(); bool jeOtocenNaVychod(); bool jeOtocenNaZapad();`

Aby se Karel mohl určitým způsobem orientovat ve svém městě, může na základě následujících příkazů zjistit, na kterou ze světových stran je natočen – *jeOtocenNaSever*, *jeOtocenNaJih*, *jeOtocenNaVychod* a *jeOtocenNaZapad*.

Návratová hodnota jednotlivých funkcí je *true*, pokud je Karel natočen na příslušnou světovou stranu.

3.1.2 Karlovo město

Karlovo město je čtvercová nebo obdélníková síť, ve které se robot Karel pohybuje po jednotlivých políčkách (čtverečcích). Město může být různě členěno pomocí zdí, přes které Karel nemůže projít a musí se jim vyhýbat, jinak do nich narazí. Samotnou hranici města Karel vnímá jako zeď a nemůže ji nikdy překročit. Město je orientováno tak, že jeho severní hranice je identická s horní hranou monitoru počítače. Na všech políčkách města, na nichž není zeď, se mohou nacházet značky. Tyto značky má Karel možnost pokládat či sbírat. Maximální možný počet značek na jednom políčku je 6. Poslední objekt vyskytující se v Karlově městě je jeho domeček. Domeček musí být ve městě bezpodmínečně umístěn. Slouží jako označení místa ve městě, kde se Karel po spuštění programovacího rozhraní nachází, pokud není definována jeho jiná pozice. Domeček může také sloužit jako označení bodu ve městě, kam má Karel dojít.

Město po vykreslení nemůže být větší, než je nastavené rozlišení obrazovky počítače u operačních systémů Windows, a větší než rozlišení 1280 x 720 u Unixových systémů. Maximální počet políček města (velikost města), jež mohou být vykresleny, je dán jejich samotnou velikostí. Základní velikost políčka je nastavena na 40 pixelů. Minimální velikost města je jedno políčko.

Mapa města a její vytvoření

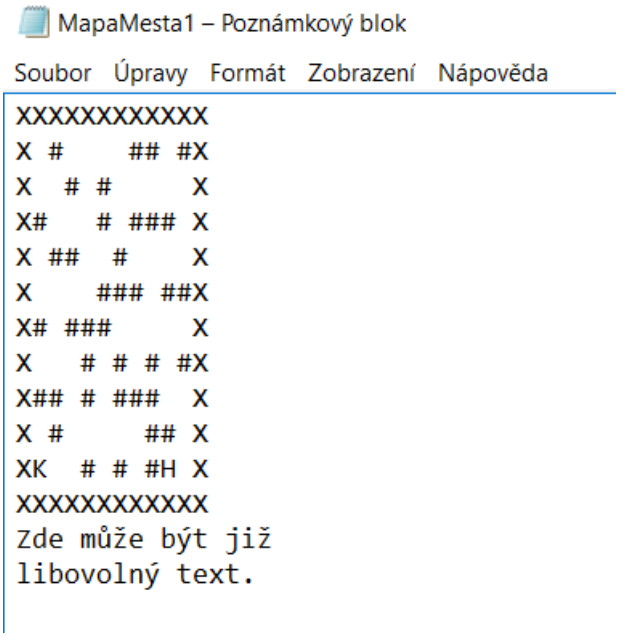
Pomocí tzv. mapy si uživatel může vytvořit vlastní podobu města, ve kterém se bude robot Karel pohybovat. Mapu města si uživatel dodržením určitých pravidel může vytvořit

v textovém souboru s příponou TXT. Aby se robot Karel v takto vytvořeném městě mohl pohybovat, je nutné v programu použít funkci knihovny *postavitMesto* a jako vstupní parametr uvést cestu k tomuto souboru.

Znaky zastupující jednotlivé prvky města jsou:

- **X** – Hranice města
- # – Zeď
- **K** – Karel
- **H** – Domeček
- **1 – 6** – Značky
- **Ostatní znaky** – Mezera

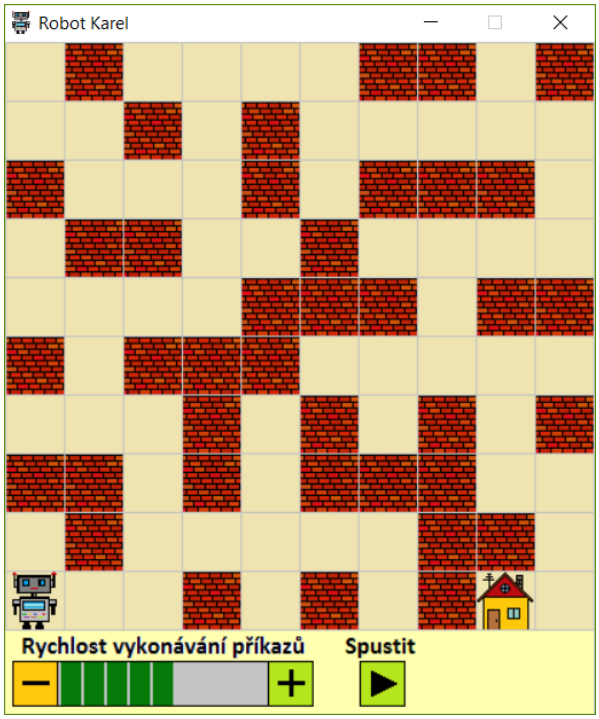
Levý horní roh hranice města musí být reprezentován prvním znakem na prvním řádku v souboru TXT. První řádek souboru (pouze znaky X) znázorňuje severní hranici města. Další řádky souboru znázorňují město samotné. Aby byla jasně stanovena západní a východní hranice města, musí tyto řádky začínat a končit znakem X. Poslední řádek pak reprezentuje jižní hranici města a je identický s prvním řádkem souboru (pouze znaky X). Všechny řádky znázorňující hranice města a město samotné musí mít stejný počet znaků. Na řádcích následujících po řádku jižní hranice města se již mohou nacházet libovolné textové řetězce. Tyto řádky se neskenují jako mapa města. Uvnitř města se znaky zastupující zeď, Karla, domeček a značky mohou použít libovolně. Povinným znakem tvořícím obsah města je znak H zastupující domeček.



Obr. 10. Správně vytvořená mapa Karlova města v souboru TXT.

3.1.3 Grafické rozhraní

Při správném načtení mapy města příkazem *postavitMesto* se zobrazí okno znázorňující samotné město i s robotem Karlem (Obr. 11).



Obr. 11. Ukázka grafického rozhraní Knihovny robota Karla.

Po spuštění uživatelského programu se programovací rozhraní nachází v pozastaveném režimu. V uživatelském kódu je běh programu pozastaven v místě volání funkce *postavitMesto*. Vykonávání uživatelského programu lze zahájit kliknutím na tlačítko Spustit nebo stisknutím klávesy Enter či Mezerník. Vykonávaný uživatelský program lze opět kdykoli pozastavit, a to kliknutím na tlačítko Zastavit nebo stisknutím klávesy Enter či Mezerník.

Rychlost vykonávání Karlových příkazů lze ovlivnit na panelu Rychlosti vykonávání příkazů. Změnu lze provést kliknutím na tlačítka Plus a Minus na uvedeném panelu nebo stisknutím kláves Plus či Minus. Okno programovacího rozhraní může uživatel kdykoli uzavřít kliknutím na křížek pro uzavření okna nebo stisknutím klávesy Escape. Při vykonávání příkazu *zbouratMesto*, který se stará o zrušení okna programovacího rozhraní, dojde před samotným zrušením okna k 1,5 sekundové prodlevě. Tato prodleva by měla uživateli umožnit vizuálně postřehnout finální polohu robota Karla.

3.2 Instalace knihoven

Aby bylo možné používat Knihovnu robota Karla, je nutno provést několik změn v nastavení projektu uživatele, ve kterém bude tato knihovna využívána. Uživatel může tuto knihovnu využívat v běžných vývojových prostředích. V každém vývojovém prostředí se umístění obsahu nastavení různí, ale potřebné body nastavení projektu pro jeho správný chod zůstávají stejné.

3.2.1 Prostředí CodeLite na platformě Windows

U projektu ve vývojovém prostředí CodeLite, který si uživatel vytvoří pro práci s robotem Karlem, je nutné nastavit zpracovávání zdrojových kódů tohoto projektu kompilátorem MinGW (mingw32) určeným pro Windows.

Grafická stránka věci programového rozhraní Knihovny robota Karla je řešena s využitím grafické knihovny SLD. Proto je třeba tuto knihovnu umístit v počítači a nastavit projekt tak, aby bylo možné knihovnu využívat. Poslední vydanou verzi knihovny SDL je možné stáhnout z oficiálních stránek knihovny [17]. Aby byla knihovna SDL kompatibilní s Knihovnou robota Karla, je nutné mít v počítači umístěnou vývojovou verzi SDL 2.0 a vyšší pro platformu Windows a kompilátor MinGW. Součástí přílohy bakalářské práce je i knihovna SDL ve verzi SDL2-2.2.4.

Pro správné nastavení projektu se nejprve musí definovat, kde má kompilátor hledat hlavičkové soubory knihovny SDL, jejichž obsah je třeba vložit (*„includovat“*) do zdrojového kódu Knihovny robota Karla. V Project Settings daného projektu je tedy nutno do položky Include Paths kompilátoru vložit absolutní cestu k těmto hlavičkovým souborům. Relativní cesta k těmto souborů v adresáři knihovny SDL je /x86_64-w64-mingw32/include/SDL2. Je-li třeba kompilátoru nadefinovat více cest, oddělují se tyto cesty středníkem.

Dále se musí nastavit linker k přilinkování SDL knihovny, tj. nastavit vyhledávací cestu a konkrétní knihovny. V Project Settings je třeba do položky Libraries Search Path linkeru vložit absolutní cestu k objektovým souborům. Relativní cesta k těmto souborům v adresáři knihovny SDL je /x86_64-w64-mingw32/lib. Do položky Libraries linkeru je nutno vložit tyto knihovny: SDL2main a SDL2. Jednotlivé knihovny se oddělují pouze středníkem.

Pro zajištění správné funkčnosti SDL knihovny je nutné připojit její dynamickou knihovnu SDL2.dll. Ta se taktéž nachází v adresáři grafické knihovny SDL a její relativní cesta je /x86_64-w64-mingw32/bin/SDL2.dll. Dynamickou knihovnu je nutno nakopírovat do výstupního adresáře uživatelského projektu. Ve vývojovém prostředí CodeLite je standardně výstupním adresářem adresář *Debug* v uživatelském projektu.

Aby uživatel mohl ve svém programu využívat Knihovnu robota Karla, musí vložit zdrojové kódy této knihovny, tj. soubory *KnihovnaRobotaKarla.c* a *KnihovnaRobotaKarla.h*, do svého projektu a použít direktivu *#include <KnihovnaRobotaKarla.h>*.

Knihovna robota Karla používá ve svém programovém prostředí textury. Poslední krok, který je třeba provést pro zajištění správné funkčnosti knihovny, je nakopírování adresáře *Textury* do výstupního adresáře *Debug* uživatelského projektu. Adresář Textury se rovněž nachází v příloze práce.

Chce-li uživatel používat některou z rozšiřujících knihoven SDL, je zapotřebí zopakovat obdobný postup popsaný výše pro danou knihovnu. Musí definovat cestu, kde má kompilátor hledat hlavičkové soubory rozšiřující knihovny, dále vložit cestu, odkud má linker přilinkovat dílčí knihovny a názvy těchto knihoven. Na závěr je nutno nakopírovat všechny dynamické knihovny do výstupního adresáře uživatelského projektu.

4 IMPLEMENTACE ŘEŠENÍ KNIHOVNY

Knihovna robota Karla vytvořená v rámci práce se skládá ze zdrojového souboru *KnihovnaRobotaKarla.c*, hlavičkového souboru *KnihovnaRobotaKarla.h* a adresáře *Textury*.

Ve zdrojovém souboru jsou obsaženy všechny dostupné funkce knihovny, které se podílí na správném fungování programového rozhraní knihovny. V hlavičkovém souboru jsou definovány tzv. uživatelské funkce, které může uživatel použít k ovládání robota Karla. Tento hlavičkový soubor musí být vložen do uživatelského programu direktivou *include*. Adresář *Textury* obsahuje textury, které jsou potřebné pro správné vykreslování programového rozhraní knihovny. Adresář musí být umístěn ve výstupní složce uživatelského projektu.

4.1 Popis konstant, výčtových a datových typů, struktur a globálních proměnných

4.1.1 Konstanty

Konstanta ***SIZE_OF_THE_PLAY_POINT*** datového typu Integer udává rozměry v pixelech jednoho čtvercového pole čtvercové či obdélníkové sítě města. Základní hodnota konstanty je 40. Změnou hodnoty této konstanty se úměrně změní velikost všech polí sítě města. Na tento nový rozměr se přizpůsobí i textury znázorňující objekty ve městě. To může způsobit vizuální změnu textur vykreslených v programovém rozhraní, protože původní textury mají velikost 40x40 pixelů.

Konstanty ***START_POINT_X*** a ***START_POINT_Y*** datového typu Integer určují počáteční bod v okně programového rozhraní, odkud se bude vykreslovat město. Základní hodnotou je 0. Město se tedy vykresluje v počátku souřadnicového systému, tj. v levém horním rohu okna programového rozhraní.

4.1.2 Výčtové typy enum

Výčtový typ ***TexturesOfKarelAndMap*** označuje význam jednotlivých textur a slouží pro názornější orientaci mezi indexy v poli *gTexturesOfKarelAndMap*

4.1.3 Nové datové typy

Datový typ ***significanceOfThePlayPoint*** je datovým typem dvourozměrného pole (dále jen „matice“) *gSemanticMatrix*. Datový typ udává, co se v jednotlivých polích čtvercové či obdélníkové sítě města nachází – zda je na daném místě zeď, značka apod.

direction je datový typ určující, kterým směrem je robot Karel natočen.

Datový typ ***status*** definuje, v jakých stavech se může nacházet programové rozhraní. Stav *PAUSE* znamená, že se čeká na podnět uživatele ke spuštění vykonávání příkazů. *PLAY* je stav indikující, že robot Karel vykonává příkazy. *NOT_INIT* znamená, že uživatel nepoužil příkaz *postavitMesto* a aktuální příkaz nemůže být vykonán.

4.1.4 Struktury

Struktura ***karelProperties*** reprezentuje vlastnosti robota Karla. Proměnné *rowMatrix* a *columnMatrix* této struktury jsou určeny pro indexy matice *gSemanticMatrix* a definují, na kterém prvku matice se robot aktuálně nachází. Proměnná *texture* je pointer na texturu, kterou je robot aktuálně vykreslován. *positionRectangle* je pointer na strukturu *SDL_Rect* knihovny *SDL*, která slouží pro definování velikosti a pozice oblasti při vykreslování textury robota. Proměnná *karelsDirection* definuje směr, kterým je robot aktuálně natočen. *houseRow* a *houseColumn* jsou indexy matice *gSemanticMatrix* a definují, v kterém prvku matice je umístěný Karlův domeček. Tato informace slouží k tomu, aby na jednom poli čtvercové sítě města mohly být vykresleny textury domečku, značky a robota Karla. Proměnná *speedLevel* je určena pro stupeň rychlosti vykonávání příkazů, kterou reprezentuje počet vykreslených obdélníků na panelu Rychlosti vykonávání příkazů programového rozhraní. Poslední proměnnou struktury je proměnná *speed* datového typu Float. Ta udává počet taktů procesoru pro pozastavení vykonávání programu tak, aby byla splněna rychlost vykonávání Karlových příkazů stanovená uživatelem na panelu Rychlosti vykonávání příkazů.

4.1.5 Globální proměnné

Globální proměnná ***gWindow*** obsahuje pointer na vytvořené okno programového rozhraní, do kterého se vykresluje požadovaný obsah. Pointer okna je získán pomocí funkce *SDL_CreateWindow* knihovny SDL.

gRenderer je globální proměnná obsahující pointer na funkci, která zprostředkovává 2D vykreslování do zvoleného okna programového rozhraní. Pointer renderu je získán funkcí *SDL_CreateRenderer* knihovny SDL.

gTexturesOfKarelAndMap je pole pointerů, jehož prvky odkazují na jednotlivé textury umístěné v paměti. Pointery na textury do pole vkládá funkce *loadingTexturesAndCreatingFavicon*.

gPlayGroundWidht a ***gPlayGroundHeight*** jsou globální proměnné, do kterých se ukládá zjištěná šířka a výška města, tj. počet čtvercových polí na šířku a na výšku.

Globální proměnná ***gSemanticMatrix*** reprezentuje dynamické dvojrozměrné pole označované jako matice. Hodnota proměnné je pointer na pointer. O vytvoření matice se stará funkce *createSemanticMatrix*, která na základě velikosti mapy města vytvoří identicky velkou matici. Tato matice je obrazem mapy města v souboru TXT a slouží pro určování vlastností jednotlivých míst ve městě.

V globální proměnné ***gStatus*** datového typu *status* je již při deklaraci nastavena hodnota *NOT_INIT*, která značí, že neproběhl proces inicializace pro správné fungování programového rozhraní. Hodnota proměnné *gStatus* je testována vždy na začátku každé uživatelské funkce.

gErrorsKarel je globální proměnná udávající počet chyb, které nastaly nesprávným použitím uživatelských funkcí. Proměnná je inkrementována v případě, použije-li uživatel nějaký Karlův příkaz, aniž by inicializoval město. Za chybu je taktéž považováno nesprávné použití příkazu *krok*, kdy je důsledkem náraz robota Karla do zdi. Pokusí-li se robot Karel zvednout značku, která se na daném místě nenachází nebo položí-li více jak 6 značek na jedno místo, je taktéž proměnná *gErrorsKarel* inkrementována.

Globální proměnná ***gErrorsInternal*** se inkrementuje, nastanou-li při vykonávání funkcí knihovny nějaké chyby. Tyto chyby mohou být způsobeny např. pokusem o otevření neexistujícího souboru, nedostatkem paměti pro alokaci či chybným vykreslením obrazu programového rozhraní.

Statické pole ***gTimeFormat*** je určeno pro funkci *getTime*, která vrací textový řetězec aktuálního času.

4.2 Struktura a popis funkcí knihovny

Ke Knihovně robota Karla je k dispozici podrobná dokumentace ve formátu Doxygen. Tato dokumentace je součástí přílohy práce. Z tohoto důvodu zde nejsou podrobně popsány vlastnosti všech funkcí, ale pouze důležité body, které je nutno provést pro správný běh programového prostředí.

Přestože knihovna poskytuje grafický výstup, nemůže být řešena jako většina grafických aplikací. V případě většiny grafických aplikací probíhá řešení tak, že se nejprve načte konfigurace a zinicializuje se vše potřebné pro správný běh aplikace. V nekonečné smyčce se následně provádí kontrola vstupů. Na jejich základě se provedou požadované změny, aktualizuje se obraz a na závěr se obraz vykreslí. Nekonečná smyčka však nemůže být v Knihovně robota Karla použita, protože by uživateli neumožnila vykonávat kód, který napsal ve vývojovém prostředí. Načtení konfigurace a inicializaci programového rozhraní uživatel provede použitím příkazu *postavitMesto*. Nekonečná smyčka je nahrazena jednotlivými příkazy, které robota Karla ovládají. Každý příkaz provede odpovídající změnu, aktualizuje a vykreslí obraz.

4.2.1 Načtení konfigurace a inicializace

K využívání funkcí knihovny SDL je nutno tuto knihovnu inicializovat pomocí funkce *SDL_Init*. Při inicializaci se definují tzv. subsystémy, které zpřístupňují prostředky (např. určité knihovny DirectX), jenž bude knihovna využívat. V Knihovně robota Karla se inicializuje pouze subsystém *SDL_INIT_VIDEO*, který se stará o práci s grafikou a automaticky inicializuje subsystém pro práci s událostmi.

Při vytváření okna programového prostředí pomocí funkce *SDL_CreateWindow* je nutno nejprve určit velikost tohoto okna. Velikost se určí z počtu znaků definujících město – jeho šířku a výšku v souboru TXT mapy města. Tyto hodnoty jsou uloženy v proměnných *gPlayGroundWidht* a *gPlayGroundHeight*. Pro vykreslování scény do vytvořeného okna se funkcí *SDL_CreateRenderer* definuje tzv. render, který bude vykreslování provádět.

Textury, které se mají v programovém prostředí používat, jsou obrázky ve formátu BMP s barevnou hloubkou 24 bitů. Jednotlivé obrázky se nahrají do operační paměti RAM pomocí funkce *SDL_LoadBMP*. Obrázky jsou uloženy v softwarové struktuře *SDL_Surface*, která umožňuje i jejich editaci. Protože formát BMP s 24 bitovou hloubkou neuchovává informaci o průhledném pozadí, nastaví se pomocí funkce *SDL_SetColorKey* bílá barva

pozadí textur jako průhledná. Funkce *SDL_CreateTextureFromSurface* převede strukturu *SDL_Surface* do hardwarové struktury *SDL_Texture*. Dojde k přemístění textury do paměti grafické karty, čímž se docílí rychlejšího zpracovávání textur.

Významová matice *gSemanticMatrix* je vyplněna odpovídajícími hodnotami během postupného procházením textového souboru s mapou města. Počáteční poloha robota je dána jeho umístěním v mapě města. Jeho natočení je vždy směrem na jih. O nastavení počátečních hodnot struktury *karelProperties* v proměnné *gKarel* se stará funkce *initKarel*.

4.2.2 Vytvoření a vykreslení obrazu

Po úspěšné inicializaci Knihovny robota Karla se vykresluje scéna do již vytvořeného okna programového rozhraní. Funkcí *createPlayGround* se vykreslí síť města, po které se robot Karel bude pohybovat. Jednotlivé objekty v síti města jsou vytvořeny postupným procházením matice a vykreslením odpovídajících textur na příslušná místa – funkce *fillPlayground*. Robot Karel se na odpovídající místo vykreslí na základě hodnot proměnné *gKarel*. Na závěr je funkcí *createToolBar* vykreslen ovládací panel programového rozhraní a celý obraz je zprostředkován na monitor pomocí renderu *gRenderer*.

Dojde-li ke změně scény vyvolané např. příkazem pro pohyb robota či uživatelským zásahem vyvolávajícím událost, nedochází k překreslení celé scény okna, ale pouze těch částí, kterých se změna týká. Např. při použití příkazu *krok* se v síti města překreslí pouze nová a stará pozice robota Karla.

4.2.3 Obsluha událostí

Obsluhou událostí se v Knihovně robota Karla řeší ukončení programového rozhraní, změna rychlosti vykonávání příkazů či samotné spuštění a pozastavení vykonávání příkazů.

Pohne-li uživatel myší, stiskne klávesu apod., je operačním systémem vytvořen objekt události s parametry, které charakterizují danou událost. Knihovna SDL reprezentuje objekt události unionem *SDL_Event*. Položky tohoto typu jsou struktury jednotlivých událostí. Struktury všech možných typů událostí, které SDL knihovna podporuje, jsou uvedeny v dokumentaci knihovny [19]. Události, které nastanou, knihovna SDL vkládá do fronty událostí. Z této fronty mohou být události postupně vytahovány.

Funkce *SDL_PollEven* umožňuje vložit pointer na union první události ve frontě do libovolné proměnné, se kterou je možno dále v programu pracovat. Každá struktura, která odpovídá jednotlivým typům událostí a je jedním z prvků typu *SDL_Event*, obsahuje prvek *type*. S jeho pomocí je možné v programu určit, o jaký typ událostí se jedná.

V Knihovně robota Karla se o obsluhu událostí stará funkce *checkEvents*. Ta je po zavolání vykonávána, dokud není fronta událostí knihovny SDL prázdná. Funkce je rozvětvena podle jednotlivých typů událostí, které Knihovna robota Karla využívá.

Funkce *checkEvents* je volána ve funkci *myDelay*. Tato funkce řeší zpoždění vykonávání příkazů. Vždy před uspáním vlákna, ve kterém bylo vytvořeno okno programového rozhraní, je ve funkci *SDL_Delay* volána funkce *checkEvents* pro vykonání obsluhy událostí, které mohly nastat.

5 PŘÍKLADY VYUŽITÍ KNIHOVNY ROBOTA KARLA

Součástí přílohy bakalářské práce je několik příkladů využití Knihovny robota Karla. Tyto příklady mají jednodušší a středně složitý charakter. Slouží jako ukázky řešení daných problémů při využití Knihovny robota Karla. Jsou koncipovány tak, aby měly nejen ukázkový, ale i inspirativní význam a vzbudily ve čtenáři nápad, jak daný příklad modifikovat či řešit elegantněji.

5.1 Průchod bludištěm – algoritmus Sledování zdí

Každý z nás někdy zkoušel najít správnou cestu napříč bludištěm. Ať už zahradním či bludištěm nakresleném v nějakém časopise. Typů bludišť existuje velké množství a můžeme je dělit podle topologie, dimenze, typu mozaiky a směrování.

Bludiště vytvořené jako mapa města pro robota Karla, umožňuje vytvořit pouze ortogonální 2D bludiště s topologií Euklidovského prostoru. Jinými slovy, bludiště ve dvourozměrné rovině tvořené čtvercovými buňkami, jehož cesty se kříží v pravých úhlech.

Pro jednoduchý příklad ukázky použití Knihovny robota Karla uvažujme průchod Karla perfektním bludištěm za pomocí sledování zdí. Perfektní bludiště je takové, ve kterém mezi startem a cílem existuje jen jedna správná cesta. Neobsahuje tedy žádné alternativní cesty ani oddělené nepřístupné části. Algoritmus sledování zdí vychází z toho, že při vstupu do bludiště je zvolená jedna ze zdí, která je po celou dobu průchodu bludiště sledována.

```
postavitMesto("mapyMesta\\Bludiste-perfektni.txt");
bool konec = false;
if(jeZdeDomecek()){
    printf("Jsem doma!");
    konec = true;
}
while(!konec){
    if(jeVlevoZed() && !jePredemnouZed()){
        krok();
    }else{
        if(!jeVlevoZed()){
            vlevoVbok();
            krok();
        }else{
            while(!jeVlevoZed() || jePredemnouZed()){
                vlevoVbok();
            }
            krok();
        }
    }
    if(jeZdeDomecek()){
        printf("Jsem doma!");
        konec = true;
    }
}
zbouratMesto();
```

Obr. 12. Ukázka Algoritmu, pomocí kterého robot Karel najde v bludišti svůj domeček.

Robot Karel se drží zdi po své levé straně a prochází bludištěm, dokud nenarazí na svůj domeček. Algoritmus je jednoduchý, ale nezaručuje nalezení cíle ve všech typech bludišť.

5.2 Pascalův trojúhelník

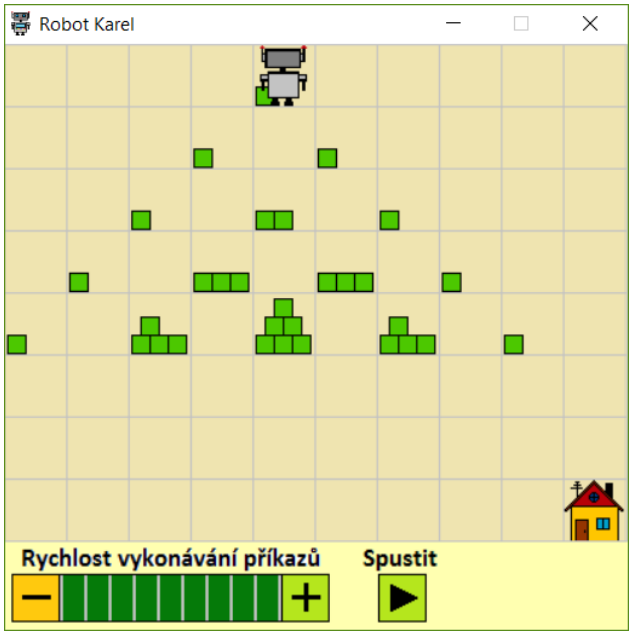
Jednoduchým a přitom názorným příkladem k pochopení či procvičení rekurze je vykreslení Pascalova trojúhelníku pomocí Karlových značek [11].

Robot Karel nejprve na místě, na němž stojí, položí značku. Následně udělá krok dolů a vlevo, kde položí další značku, a dva kroky vpravo, kde taktéž položí značku. Na místě, na kterém má robot Karel pokládat značku, se předchozí postup zopakuje (rekurze). Podmínkou pro ukončení rekurze je dosažení počtu řádků Pascalova trojúhelníku, které se mají vykreslit. Hodnoty Pascalova trojúhelníku jsou součty značek v jednotlivých polích města.

```
1  void pascaluvTrojuhelnik(int pocetRadku){
2      while(!jeOtocenNaJih()){
3          vlevoVbok();
4      }
5      polozZnacku();
6      if(pocetRadku > 1){
7          jitVlevo();
8          pascaluvTrojuhelnik(pocetRadku - 1);
9          jitVpravo();
10         pascaluvTrojuhelnik(pocetRadku - 1);
11         vratitSeZpet();
12     }
13 }
```

Obr. 13. Ukázka funkce pro vykreslení Pascalova trojúhelníku.

Ke správnému vykreslení Pascalova trojúhelníku nesmí Karlovi bránit v pohybu žádná zeď. Jelikož může robot Karel položit na jedno místo maximálně 6 značek, umožňuje tento algoritmus vytvořit pouze 5 řádků Pascalova trojúhelníku. Výsledek algoritmu je znázorněn na obrázku (Obr. 14).



Obr. 14. Pascalův trojúhelník vytvořený robotem Karlem pomocí značek.

5.3 Záplavový algoritmus

Robot Karel může jednoduše ilustrovat záplavový algoritmus. Tento algoritmus např. používá protokol OSPF při dynamickém směrování v počítačových sítích. Může být využíván

při vyplňování oblastí rastrové grafiky (semínkové záplavové vyplňování), hledání cesty z bludiště a jiné. Záplavový algoritmus je velice jednoduchý. Je charakteristický tím, že se šíří všemi směry, což může být někdy žádoucí, ale naopak to může způsobovat i neefektivnost daného řešení.

Záplavový algoritmus můžeme použít např. při řešení úkolu, kdy má robot Karel navštívit všechna dostupná místa ve svém městě a na každé z nich umístit jednu značku.

```
1  void vypln(smer iteracniSmer){
2      smer smerOdkuJsemPrisel = iteracniSmer;
3      if(!jeZdeZnacka()){
4          polozZnacku();
5          naVychod();
6          naSever();
7          naZapad();
8          naJih();
9          vratitSeOdkudJsemZdePrisel(smerOdkuJsemPrisel);
10     }
11 }
```

Obr. 15. Ukázka funkce charakterizující záplavový algoritmus.

Aby bylo patrné, v které části algoritmu se robot Karel nachází, je celá implementace Záplavového algoritmu doplněna o logování na výstupní konzole. Karel využívá již položených značek k tomu, aby zjistil, zda již na daném místě byl (funkce *naSever*, *naJih*, *naVýdchod* a *naZápad*). Nachází-li se ve městě nějaká značka, kterou sám Karel nepoložil, vyhodnotí po objevení této značky, že na daném místě již byl a nebude tímto směrem dále pokračovat.

5.4 Průchod bludištěm – algoritmus prohledávání do hloubky

Prohledávání do hloubky je grafový algoritmus, který umožňuje systematický průchod grafem, např. bludištěm. Ke splnění podmínky, že algoritmus projde všechny uzly grafu a žádný neprojde vícekrát než jednou, využívá algoritmus zásobník sloužící k dočasnému ukládání uzlů grafu, které v danou chvíli nemají být prohledávány [20].

Algoritmus prohledávání do hloubky je zde předveden na příkladu, kde má robot Karel posbírat všechny značky v bludišti, tj. musí projít všemi uzly grafu. Jelikož se jedná o složitější příklad, celý zdrojový kód implementace algoritmu prohledávání do hloubky je okomentován. V příslušném adresáři přílohy práce je k dispozici i dokumentace.

Algoritmus pracuje se zásobníkem *stag*, který je implementován pomocí statického pole o velikosti 100 prvků. Pole je současně prvkem struktury *zasobnik*, která dále obsahuje proměnnou *vrchol* označující vrchol zásobníku. Při uložení většího množství prvků na zásobník než je dovoleno, dojde k jeho přetečení a předčasnému ukončení programu. Vhodnějším řešením by byla implementace zásobníku pomocí lineárního seznamu, který by se v případě potřeby dynamicky zvětšoval. To ale není předmětem tohoto příkladu. Do zásobníku se budou ukládat křižovatky, které Karel již navštívil, ale ještě celé neprozkoumal. Jednotlivé prvky zásobníku jsou struktury *krizovatka*. Jedním z prvků této struktury je struktura *souradnice*, která označuje polohu dané křižovatky v bludišti. Aby bylo zřetelné, které odbočky dané křižovatky Karel již navštívil, jsou dalšími prvky struktury *krizovatka* 4 proměnné, které uchovávají informaci o statusu jednotlivých odboček křižovatky. Na vrchol zásobníku je možné přidat nový prvek pomocí funkce *vlozit*. Funkcí *vyjmout* lze pak prvek z vrcholu zásobníku také odebrat.

K identifikaci jednotlivých křižovatek je před startem algoritmu prohledávání bludiště vytvořen imaginární souřadnicový systém. Aktuální pozici, na které se robot Karel nachází, je přiřazena souřadnice [0;0]. Ostatní pozice bludiště mají odpovídající souřadnice vztažené k této startovací pozici.

Po spuštění algoritmu, tj. po spuštění funkce *algortimusProhledavani*, je nejprve provedena identifikace okolí Karla. Podle okolností Karel změní polohu či natočení tak, aby funkce *algortimusProhledavani* byla vykonávána správně.

Karel se následně ve směru, ve kterém je natočen, pohybuje tak dlouho (funkce *chuzeDokudNeniKrizovatkaNeboSlepaUlice*), dokud nenarazí na křižovatku, nebo slepou ulici (uzel grafu). Na základě hodnoty, kterou funkce vrátí, se program rozvětví.

Nachází-li se Karel na křižovatce, je ověřeno funkcí *znamTutoKrizovatku*, zda není v zásobníku uložena křižovatka se stejnými souřadnicemi.

Je-li návratová hodnota *NULL*, znamená to, že je Karel na této křižovatce poprvé. Křižovatka se uloží do zásobníku a Karel se vydá některou z možných cest křižovatky (rekurzivně se zavolá funkce *algortimusProhledavani*). Jsou-li již všechny cesty křižovatky navštíveny, vyjme se poslední křižovatka ze zásobníku, tj. ta, na které Karel aktuálně stojí, a vrátí se na nejbližší křižovatku, ze které na toto místo přišel. Na této křižovatce si označí cestu, odkud přišel, za prozkoumanou.

Zjistí-li Karel, že na křižovatce, na které se nachází, již někdy byl, označí pouze cestu, odkud na tuto křižovatku přišel, otočí se a vrátí se na nejbližší křižovatku. To zamezí Karlovi, aby se v bludišti zacyklil.

Stojí-li Karel ve slepé uličce bludiště, taktéž se vrátí na nejbližší křižovatku a označí danou cestu za prozkoumanou.

Karel při pohybu po bludišti kontroluje, zda se na daném místě nachází nějaké značky. V případě nálezu značky ji zvedne. Operace, která se má provádět v uzlech, případně hranách grafu, může být libovolná.

ZÁVĚR

Jedním z cílů práce je analyzovat stávající implementace prostředí robota Karla. Tímto se zabývá teoretická část, ze které vyplývá, že jazyk Karel byl ve své době velice populární. Bylo vytvořeno mnoho implementací prostředí robota Karla a v této práci zdaleka nejsou zmíněny všechny z nich. Jazyk Karel ale upadá částečně v zapomnění, protože většina implementací byla vytvořena před dlouhou dobou. Není tak k dispozici mnoho volně dostupných řešení pro moderní operační systémy, které by byly pro uživatele zajímavé a snadno dostupné.

V praktické části je vytvořena a popsána jednoduchá Knihovna robota Karla, která oživuje programovací jazyk Karel jakožto jednoduchý jazyk pro pochopení základů programování. Knihovna přenáší tuto myšlenku pomocí svých dostupných funkcí do známého a používaného jazyka C.

Důležitou součástí Knihovny robota Karla je knihovna SDL. Pomocí funkcí této knihovny je vytvářeno grafické programové rozhraní Knihovny robota Karla. Celá knihovna je vytvořena tak, aby uživateli poskytovala jednoduchou funkcionalitu, byla čitelná a mohla být názornou inspirací, že i v jazyce C mohou být jednoduše tvořeny zajímavé grafické aplikace.

Vytvořená dokumentace Knihovny robota Karla by měla sloužit pro snadnější orientaci a pochopení jednotlivých částí knihovny.

SEZNAM POUŽITÉ LITERATURY

[1] ROBERTS, Eric. *Karel the Robot Learns Java* [online]. Stanford University, 2005 [cit. 2017-03-14]. Dostupné z: https://web.stanford.edu/class/cs106a/textbook/karel-the-robot-learns-java.pdf

[2] Etymology Dictionary: Robot [online]. [cit. 2017-03-14]. Dostupné z: http://www.etymonline.com/index.php?allowed_in_frame=0&search=%C4%8Dapek&searchmode=none

[3] ROBERTS, Eric. *Karel the Robot* [online]. Stanford University, 2016 [cit. 2017-03-14]. Dostupné z: http://web.stanford.edu/class/cs208e/handouts/04-Karel.pdf

[4] PATTIS, Richard E., Jim ROBERTS a Mark. STEHLIK. Karel the robot: *A Gentle Introduction to the Art of Programming. 2nd ed*. /. New York: Wiley, c1995. ISBN 0471597252.

[5] BRUSILOVSKY, Peter, et al. *Mini-languages : A Way to Learn Programming Principles.* Education and Information Technologies. 1997, 2, 1, s. 70. ISSN 1360-2357.

[6] Joseph Bergin, Mark Stehlik, Jim Roberts, Richard E. Pattis. *Karel++: A GentleIntroduction to the Art of Object-Oriented Programming*. John Wiley & Sons, 1996. ISBN 0-471-13809-6.

[7] JOSEPH BERGIN .. [ET AL.]. *Karel J Robot: A Gentle Introduction to the Art of Object-oriented Programming in Java*. Prelim. ed. S.l.: Dream Songs Press, 2005. ISBN 9780970579515.

[8] GAŠPAROVIČOVÁ, Ľuba a Jozef HVORECKÝ. *Kamaráti robota Karla*. Ilustroval Koloman LEŠŠO. Bratislava: Mladé letá, 1991. ISBN 80-06-00421-8.

[9] Robot Emil [online]. [cit. 2017-03-23]. Dostupné z: http://www.emil.input.sk//index.html

[10] PC-KAREL: *Interpret jazyka Karel pro PC* [online]. [cit. 2017-03-25]. Dostupné z: http://pckarel.zde.cz/

[11] VEJVODA, Michal a Miroslav RYTÍŘ. *Programovací jazyk KAREL* [online]. 2001 [cit. 2017-03-25]. ISBN 99972-03-09-7.

[12] XKarel: Domovská stránka programu xKarel [online]. [cit. 2017-03-25]. Dostupné z: http://xkarel.sourceforge.net/cz/index.php

[13] Petr Laštovička [online]. [cit. 2017-03-25]. Dostupné z: http://petr.lastovicka.sweb.cz/

[14] Robot Karel: Vývojové prostředí [online]. [cit. 2017-03-25]. Dostupné z: http://karel.oldium.net/

[15] HANCOCK, Terry. *GettingStanford's "Karel the Robot" to Run in Debian'sEclipse* [online]. 2009 [cit. 2017-03-26]. Dostupné z: http://freesoftwaremagazine.com/articles/getting_stanfords_karel_robot_run_debians_eclipse/

[16] TIŠNOVSKÝ, Pavel. Root: *Programovací jazyky určené pro výuku programování* [online]. [cit. 2017-03-27]. Dostupné z: https://www.root.cz/clanky/programovaci-jazyky-urcene-pro-vyuku-programovani/

[17] SDL: Simple Direct Media Layer [online]. [cit. 2017-03-28]. Dostupné z: https://www.libsdl.org

[18] SDL: Simple Direct Media Layer [online]. [cit. 2017-03-28]. Dostupné z: https://www.libsdl.org/projects/

[19] SDL: Simple Direct Media Layer [online]. [cit. 2017-04-29]. Dostupné z: https://wiki.libsdl.org/SDL_Event

[20] VEČERKA, Arnošt. *Grafy a grafové algoritmy* [online]. Olomouc, 2007 [cit. 2017-04-30]. Dostupné z: https://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ANSI	American National Standards Institute
API	Application Programming Interface
BMP	Bitmap Image File
DirectX	Sada knihoven pro přímé ovládání hardwaru
FLTK	The Fast LightToolkit
GNU	GNU's Not Unix
GNU LGPL	GNU Lesser General Public License
GUI	Graphical User Interface
MinGW	Minimalist GNU for Windows
MS-DOS	Microsoft Disk Operating System
OSPF	Open Shortest Path First
RAM	Random Access Memory
RTF	Rich Text Format
TXT	Text file
VŠB	Vysoká škola Báňská v Ostravě

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘENOSNÉ MÉDIUM CD-ROM

Přenosné médium obsahuje následující soubory a adresáře:

- Dokumentace
  - Adresář obsahující dokumentaci Knihovny robota Karla.
- fulltext.pdf
  - Bakalářská práce ve formátu PDF.
- MapyMesta
  - Adresář obsahující mapy měst použité v ukázkových příkladech využití Knihovny robota Karla.
- PrikladyVyuzitiKnihovnyRobotaKarla
  - Adresář obsahující workspace vývojového prostředí CodeLite s ukázkovými příklady Knihovny robota Karla.
- SDL2-2.0.4
  - Adresář obsahující zdrojové kódy knihovny SDL ve verzi 2.0.4.
- Textury
  - Adresář obsahující textury potřebné pro správné fungování programového rozhraní Knihovny robota Karla.
- ZdrojoveKodyKnihovnyRobotaKarla
  - Adresář obsahující zdrojový a hlavičkový soubor Knihovny robota Karla.