Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Výukové prostredie založené na jazyku Karel

Bakalárska práca

2017
Andrej Zbín

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Výukové prostredie založené na jazyku Karel

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	prof. RNDr. Rastislav Kráľovič, PhD.

Bratislava, 2017
Andrej Zbín

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Andrej Zbín
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Výukové prostredie založené na jazyku Karel.
Web-based implementation of Karel-like language.

Cieľ: Cieľom je implementovať webové prostredia na výuku programovania pre žiakov základných škôl.
Výsledkom má byť použiteľný produkt, ktorý bez nutnosti inštalácie umožní riešiť úlohy v
(modifikovanom) jazyku Karel.

Hlavné požiadavky:
- navrhnúť vhodnú verziu jazyka Karel s dôrazom na vyváženosť jednoduchosti a výrazovej sily
- implementovať interpreter v jazyku JavaScript
- implementovať grafický výstup (animovaný robot) pomocou WebGL
- implementovať webstránku s možnosťou riešiť vybrané úlohy

Vedúci: prof. RNDr. Rastislav Kráľovič, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 29.10.2016

Dátum schválenia: 04.11.2016 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Abstrakt

Cieľom práce je navrhnúť a vytvoriť aplikáciu určenú na výučbu programovania žiakov na základných školách. Výsledkom je produkt, ktorý umožní riešiť úlohy v modifikovanom jazyku Karel bez nutnosti inštalácie. V práci predstavíme jazyk Karel a taktiež už existujúci softvér určený na výučbu programovania. Ďalej popíšeme návrh, použité technológie a samotnú implementáciu našej aplikácie.

Kľúčové slová: Karel, webová aplikácia, výučba programovania, edukačný sofvér

Abstract

The aim of the thesis is to design and implement an application for pupils at elementary school designed to teach programming. The result is a product that allows pupils to solve tasks in the a modified Karel programming language without the need of an installation. In our work we introduce the Karel programming language and the existing software used for teaching programming. Next, we describe the design, the technologies used and the implementation of our application.

Keywords: Karel, web application, programming teaching, educational software

Obsah

Zoznam obrázkov

Úvod

V súčasnosti existuje veľký počet softvéru určeného na výučbu programovania pre začínajúcich programátorov na základných školách. Používanie tohto softvéru zväčša prebieha bezproblémovo pod dozorom učiteľa na hodinách informatiky. Ak však žiak chce riešiť úlohy aj doma, môžu sa vyskytnúť problémy spojené so získavaním a inštaláciou daného softvéru.

Cieľom našej práce je navrhnúť a vytvoriť aplikáciu, ktorá sa bude dať spustiť priamo vo webovom prehliadači bez nutnosti inštalácie. V aplikácii by malo byť možné riešiť predpripravenú sadu úloh, ktorá má žiakovi umožniť venovať sa programovaniu aj vo svojom voľnom čase. Aplikácia by mala byť jednoduchá na používanie, zároveň však musí poskytovať dostatočnú výrazovú silu, ktorá umožní riešenie širokej škály úloh. Motivovať žiaka v riešení úloh chceme dosiahnuť interaktívnym zobrazením robota, ktorého žiak ovláda písaním programov v upravenom a rozšírenom jazyku Karel.

V prvej kapitole popíšeme pôvodnú verziu jazyka Karel, ktorým sa inšpirujeme. V ďalšej kapitole predstavíme už existujúci softvér, ktorý sa používa alebo je určený na výučbu programovania na základných školách. Medzi tento softvér patrí Karel++, Panák, Imagine a Scratch. Predstava a návrh našej aplikácie sa dá nájsť v tretej kapitole práce. Popíšeme vzhľad aplikácie a odôvodnime jednotlivé vlastnosti programovacieho jazyka, ktorý sme navrhli a v ktorom sa budú riešiť dané úlohy. Štvrtá kapitola je venovaná krátkemu popisu technológií využitých pri tvorbe aplikácie. Aplikácia je implementovaná v programovacom jazyku Javascript. Spomenieme knižnice využité pri implementácii, editor Ace, program Blender, v ktorom sme vytvorili trojrozmerné modely zobrazené v aplikácii. Posledná kapitola popisuje samotnú implementáciu aplikácie. V tejto kapitole podrobne charakterizujeme jednotlivé súbory programu a objekty a funkcie implementované v týchto súboroch.

Kapitola 1

Programovací jazyk Karel

V tejto kapitole si povieme niečo o interpretačnom programovacom jazyku Karel, ktorým sa náš programovací jazyk inšpiruje. Existuje množsvto rôznych implementácií, ktoré sa od seba líšia, či už prostredím, v ktorom sa robot pohybuje, alebo samotnými vlastnosťami a schopnosťami robota. My popíšeme verziu jazyka Karel, ktorú navrhol jeho tvorca Richard E. Pattis v knihe Karel the robot: a gentle introduction to the art of programming [12]. Programátor pomocou tohto jazyka ovláda virtuálneho robota s menom Karel.

1.1 Prostredie

Robot Karel sa pohybuje v jednoduchom dvojrozmernom prostredí po hranách jednotlivých políčok mriežky. Táto mriežka je z každej strany ohraničená a Karel sa cez tieto hranice nedokáže dostať. Na jednotlivých políčkach mriežky sa môžu nachádzať steny. Karel sa nedokáže pohybovať po spoločnej hrane dvoch susediacich políčok, pokiaľ na oboch týchto políčkach je stena. Inak povedané, vie sa pohybovať len horizontálne a vertikálne po hrane políčka mriežky, pokiaľ táto hrana nie je spoločnou hranou dvoch susediacich políčok so stenou. V rohoch políčok mriežky sa taktiež môžu nachádzať takzvané bzučiaky, ktoré dokáže robot detegovať, zbierať a ukladať.

1.2 Syntax

1.2.1 Základné inštrukcie

Programovací jazyk Karel obsahuje len malé množstvo inštrukcií, pomocou ktorých dokáže programátor ovládať robota.

- **move** prikáže robotovi pohnúť sa po jednej hrane políčka v smere, v ktorom je Karel otočený. Robot sa vždy premiestňuje z rohu políčka do iného rohu. Pokiaľ nie je

možné túto akciu vykonať, robot sa vypne a program skončí s chybou.

- **turnLeft** otočí robota o 90 stupňov vľavo. Táto akcia sa dá vykonať vždy.
- **turnOff** vypne robota. Po zavolaní tejto inštrukcie už Karel nevykoná žiadne ďalšie inštrukcie.
- **pickBeeper** povie robotovi, aby sa pokúsil zdvihnúť bzučiak z rohu, v ktorom sa práve nachádza. Podobne ako pri move, ak sa inštrukcia nedá vykonať z dôvodu, že sa v danom rohu žiaden bzučiak nenachádza, robot sa vypne a program skončí s chybou.
- **putBeeper** opak inštrukcie pickBeeper, robot sa pokúsi položiť bzučiak do rohu, v ktorom sa práve nachádza. Pokiaľ už pri sebe nemá bzučiaky, robot vypne sa a program skončí s chybou

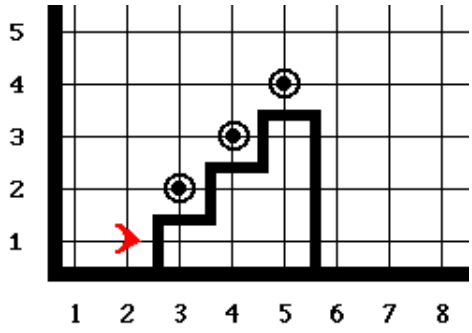
Menšie množstvo inštrukcií umožní programátorovi si tieto inštrukcie rýchlo zapamätať a osvojiť si ich používanie. Ani jedna zo základných inštrukcií nie je nahraditeľná inou základnou inštrukciu. Môžeme si všimnúť, že nám chýba napríklad inštrukcia **turnBack**, ktorá by otočila robota o 180 stupňov. Táto inštrukcia by však nepridala žiadnu funkcionalitu a je ju možné jednoducho nahradiť použitím inštrukcie **turnLeft** dvakrát.

1.2.2 Ďalšie príkazy

Malý počet inštrukcií spôsobuje, že na vykonanie určitej akcie musíme vykonať jednu inštrukciu viackrát. Napríklad ak chceme, aby sa robot otočil doprava, musíme použiť inštrukciu **turnLeft** až trikrát. Našťastie existujú v jazyku aj cykly, ktoré nám to uľahčia.

- **iterate** *počet opakovaní* **times** *príkaz* zopakuje zadaný *príkaz* zadaný počet krát.
- **while** *podmienka* **do** *príkaz* zopakuje zadaný *príkaz* pokiaľ je *podmienka* splnená. Viacej o podmienkach si povieme nižšie.
- **if** *podmienka* **do** *príkaz* vykoná *príkaz* len pokiaľ je *podmienka* splnená.

Taktiež je možné *príkaz* nahradiť skupinou, blokom, viacerých príkazov. Blok príkazov vieme vytvoriť pomocou **begin** a **end**, kde **begin** je začiatok bloku a **end** koniec.



Obr. 1.1: Ukážka prostredia Karel. Karel je znázornený červenou šípkou, hrubé čierne čiary sú steny, krúžky sú bzučiaky.

1.2.3 Orientácia v prostredí, logické výrazy

Robot Karel nepočúva len rozkazy, ale vie aj odpovedať programátorovi na otázky týkajúce sa prostredia odpoveďami áno alebo nie. Ako už bolo spomenuté, niektoré inštrukcie sa dajú vykonať len za predpokladu, že je splnená istá podmienka. Napríklad robot sa nevie pohnúť, pokiaľ sa pred ním nachádza stena. Preto existujú nasledujúce logické výrazy.

- **frontIsClear** je splnené, pokiaľ sa priamo pred robotom nenachádza stena.
- **facingNorth** je splnené, pokiaľ je robot otočený na sever, respektíve pri pohľade na mriežku zhora musí byť otočený smerom hore.
- **facingSouth** je splnené, pokiaľ je robot otočený na juh, respektíve pri pohľade na mriežku zhora musí byť otočený smerom dole.
- **facingWest** je splnené, pokiaľ je robot otočený na západ, respektíve pri pohľade na mriežku zhora musí byť otočený smerom doľava.
- **facingEast** je splnené, pokiaľ je robot otočený na východ, respektíve pri pohľade na mriežku zhora musí byť otočený smerom doprava.
- **nextToABeeper** je splnené, pokiaľ sa v rohu, v ktorom sa nachádza robot, taktiež nachádza aspoň jeden bzučiak, ktorý môže Karel zdvihnúť.
- **anyBeepersInBeeperBag** je splnené, pokiaľ má Karel vo svojom vreci aspoň jeden bzučiak, ktorý môže položiť.
- **not** *výraz* je splnené, pokiaľ *výraz* nie je splnený.

1.2.4 Vlastné príkazy

V programovacom jazyku Karel je taktiež možné vytvoriť si vlastné príkazy. Otáčať robota dozadu zakaždým pomocou cyklu alebo viacnásobným použitím **turnLeft** by bolo zbytočné. Práve absencia tejto a mnohých ďalších užitočných inštrukcií núti programátora zamýšľať sa nad vytvorením vlastných príkazov. Syntax je veľmi jednoduchá.
defineNewInstruction *meno inštrukcie* **as** *blok príkazov*

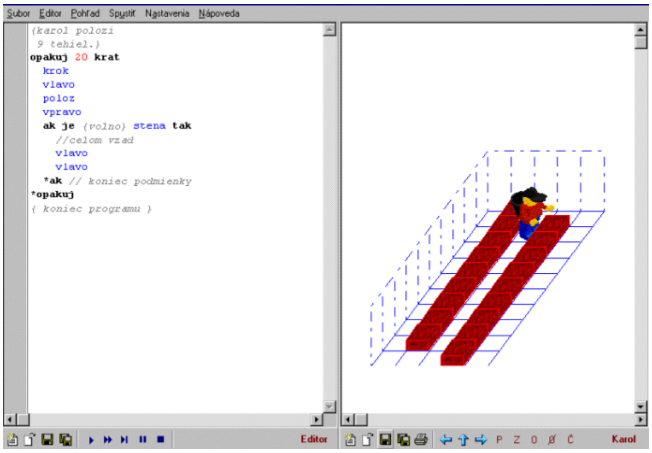
Kapitola 2

Existujúci softvér určený na výučbu programovania

V tejto kapitole v krátkosti popíšeme niektoré aplikácie, ktoré sa využívajú alebo sú určené na výučbu programovania žiakov na základných školách a preskúmame ich možné problémy. Budeme sa venovať len aplikáciám, ktoré sú lokalizované do slovenčiny.

2.1 Robot Karol++

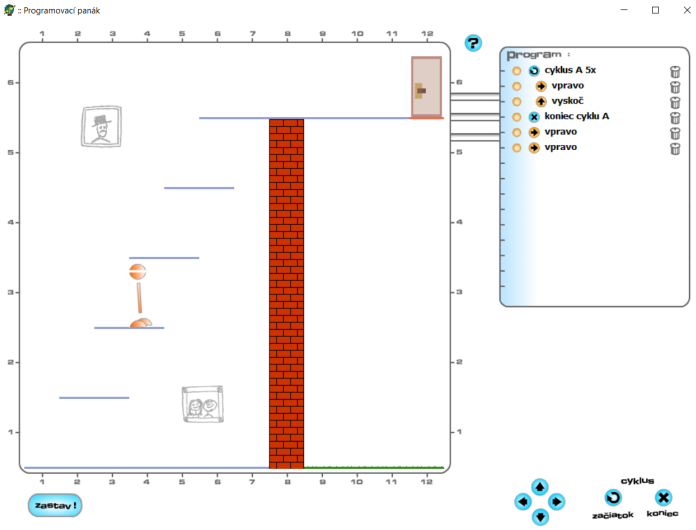
Robot Karol++ sa inšpiruje jazykom Karel. Obsahuje preddefinované inštrukcie, podmienky, cykly a podobne. Programovacie prostredie umožňuje krokovať program, zvýrazňuje syntax a podporuje automatické doplňovanie slov. Neobsahuje zabudovanú sadu úloh na riešenie priamo v programe. Autor však k programu prikladá ukážkové úlohy aj s riešeniami[8].



Obr. 2.1: Ukážka prostredia Robot Karol++.

2.2 Panák

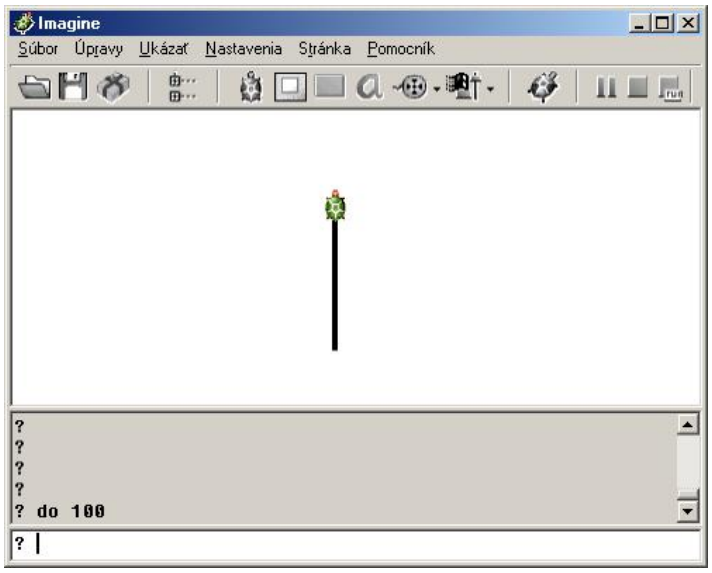
Panák je jednoduchý program, v ktorom programátor ovláda panáka pomocou štyroch základných inštrukcií pre pohyb a cyklu opakuj. Panák prechádza cez miestnosti a prekonáva prekážky. Cieľom je dostať panáka k dverám, čo spôsobí prechod do ďalšej miestnosti. Náročnosť úloh sa stupňuje. Programátor príkazy nezadáva na klávesnici, vyberá ich pomocou myši. Program existuje len vo verzii pre operačný systém Windows. Nízky počet inštrukcií a spôsob ich zadávania umožňuje programátorovi si tento program rýchlo osvojiť, no taktiež obmedzuje možnosť vytvoriť rôznorodé úlohy. Pekný grafický výstup umožňuje programátorovi dobre vidieť beh programu a prípadné chyby v programe je ľahké odhaliť a opraviť.



Obr. 2.2: Ukážka prostredia Panák.

2.3 Imagine

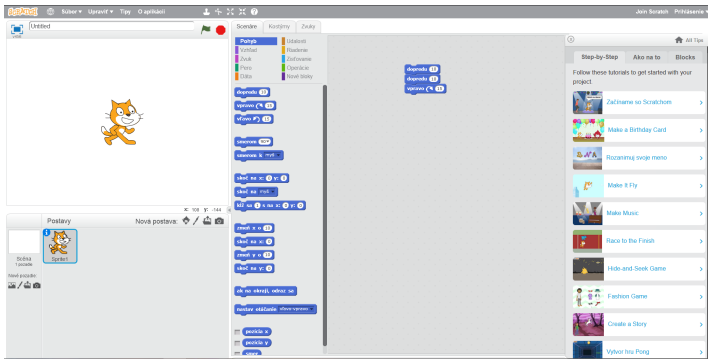
Imagine je program, v ktorom programátor ovláda korytnačku pomocou programovacieho jazyka Logo. Korytnačka sa dokáže hýbať dopredu, otáčať, kresliť, reagovať na udalosti. Programátor môže vytvárať vlastné príkazy, pomocou ktorých vie korytnačka napríklad vykresliť istý geometrický útvar. Možnosť naučiť korytnačku nové veci spojená s grafickým výstupom zvyšujú motiváciu programátora objavovať ďalšie možnosti jazyka Logo. Imagine neobsahuje úlohy na riešenie priamo v programe, avšak v učebnici 1. zošit z programovania, ktorá slúži ako návod pre Imagine, sa dajú nájsť aj úlohy na riešenie[11]. Súčasťou Imagine sú hotové ukážkové programy, ako je napríklad hra obesenec.



Obr. 2.3: Ukážka prostredia Imagine.

2.4 Scratch

Scratch umožňuje jednoduchú tvorbu interaktívnych príbehov, hier a animácií. Jedná sa o vizuálny programovací jazyk podobne ako v prípade Panáka, programátor príkazy nepíše, ale vyberá a spája pomocou myši. Scratch nie je nutné inštalovať. Prostredie je dostupné na oficiálnej stránke[7]. Vytvárať programy alebo upravovať už existujúce projekty je možné zadarmo a bez registrácie priamo vo webovom prehliadači. Existuje však aj verzia pre viaceré operačné systémy určená pre používateľov bez internetového pripojenia.



Obr. 2.4: Ukážka prostredia Scratch.

Kapitola 3

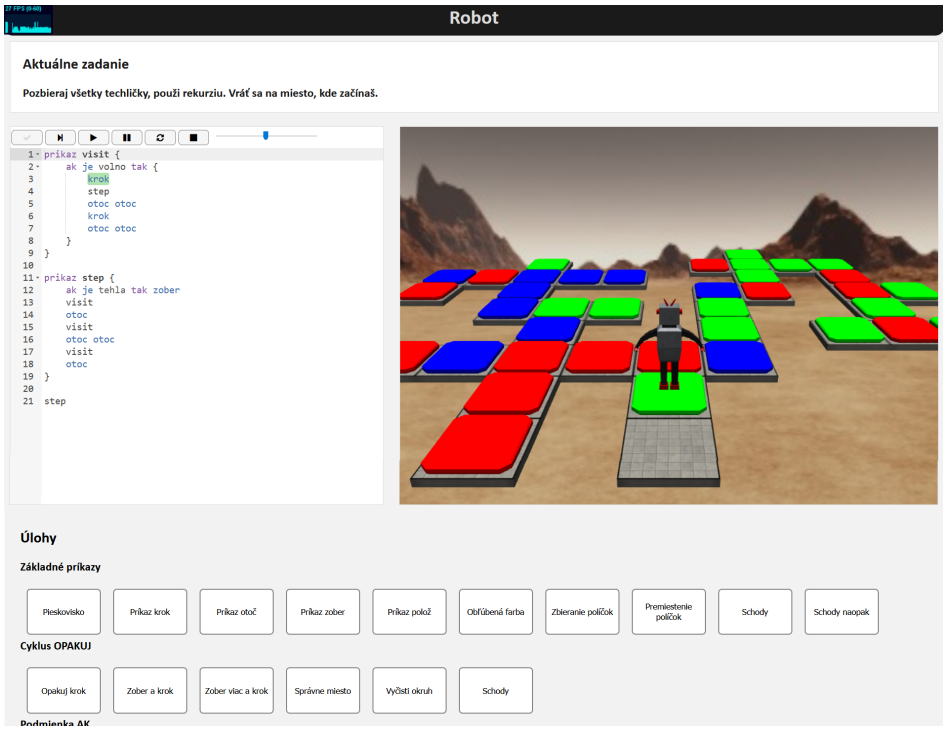
Návrh aplikácie

V tejto kapitole popíšeme našu predstavu o aplikácií, čo je jej cieľom a ako vyzerá. Naša aplikácia je určená na výučbu programovania žiakov na základných školách. Programátor v našej aplikácii ovláda virtuálneho robota jednoduchými a intuitívnymi inštrukciami. Sústredíme sa na vyváženosť jednoduchosti a výrazovej sily. Aplikácia sa dá spustiť bez nutnosti inštalácie priamo vo webovom prehliadači. Cieľom našej aplikácie je pomôcť porozumieť žiakom základným princípom programovania interaktívnou formou, kde žiak ovláda svojho virtuálna robota, ktorý plní sadu predpripravených úloh. Fungovanie aplikácie vo webovom prehliadači a sady predpripravných úloh umožňujú žiakom riešiť úlohy aj vo voľnom čase doma. Navyše uľahčí prácu učiteľom, ktorí nebudú musieť vymýšľať úlohy na riešenie na každú hodinu.

3.1 Používateľské rozhranie

Používateľské rozhranie môžeme rozdeliť na štyri hlavné časti: editor kódu, zobrazenie sveta a robota v ňom, výber úloh na riešenie a zobrazenie aktuálnej úlohy. Zadanie aktuálnej úlohy zobrazujeme na úplnom vrchu, aby bolo viditeľné hneď pri navštívení stránky. Ďalej chceme, aby mal programátor možnosť vidieť, ktorá inštrukcia sa aktuálne vykonáva a súčasne aj pohyb robota vo svete. Preto je editor kódu umiestnený hneď vedľa zobrazenia sveta. Aktuálne vykonávaná inštrukcia je zvýraznená priamo v editore. Tieto dva prvky sa nachádzajú pod zadaním úlohy, pretože programátor bude pri jej riešení viackrát prechádzať medzi editorom kódu a textom zadania. Editor kódu obsahuje tlačidlá slúžiace na spracovanie kódu a vytvorenie programu, spúšťanie programu, krokovanie programu, pozastavenie programu, opakovanie programu a zastavenie programu a taktiež posuvník, ktorým sa ovláda rýchlosť behu programu. Všetky tieto prvky sú umiestnené vo vrchnej časti editoru. Úlohy sa nachádzajú pod všetkými ostatnými časťami. Sú rozdelené do viacerých kategórii podľa typu úloh. Úloha sa dá vybrať kliknutím ľavého tlačidla myši. Výberom úlohy sa posunie stránka hore, aby

bolo viditeľné jej zadanie.



Obr. 3.1: Ukážka aplikácie.

3.2 Prostredie

Podobne ako v originálnej verzii robota Karla sa robot pohybuje po obdĺžnikovej mriežke. Rozdielom však je, že v našej aplikácii sa robot nepohybuje po hranách jednotlivých políčok ale po políčkach samotných. Na každom políčku je buď podlaha alebo je toto políčko diera. Diera má rovnakú funkciu ako stena v pôvodnej implementácii. Robot sa vie dostať na susedné políčka, ktoré majú podlahu, pohybom vpred, avšak len ak sú splnené podmienky, ktoré popisujeme ďalej v texte. Ďalším rozdielom je, že robot sa vie pohybovať aj v treťom rozmere, a teda do výšky. Namiesto bzučiakov robot ukladá a zbiera farebné tehličky, na ktoré vie vyskočiť, poprípade vie z nich zoskočiť. Tehlička má vždy jednu z troch farieb: červenú, zelenú alebo modrú. Tehličky fungujú rovnako ako bzučiaky ale s tým rozdielom, že pokiaľ sa na jednom políčku nachádza určitý väčší alebo menší počet tehličiek ako na susednom políčku, kde práve stojí robot, tak na toto políčko nevie robot prejsť. Cieľom farebných tehličiek je spestrenie sveta a interakcie s ním a taktiež možnosť vytvoriť širšie spektrum náročnosti a jedinečnosti úloh.

3.3 Robot

V starších implementáciách je robot zväčša reprezentovaný dvojrozmerným obrázkom robota. V našej aplikácii je jeho model trojrozmerný. Myslíme si, že model robota dokáže žiakov zaujať viac ako obyčajný obrázok. V súčasnej verzii model robota nie je animovaný, animácie sú však podporované.

3.4 Jazyk

Ako sme už spomenuli, sústredíme sa na vyváženosť jednoduchosti a výrazovej sily. Inštrukcie musia byť jednoduché, intuitívne a ľahko zapamätateľné, aby si ich začínajúci programátor rýchlo osvojil. To mu umožní sústrediť sa na samotné riešenie úlohy bez rozmýšľania nad syntaxou jazyka. Aj z tohto dôvodu sú všetky inštrukcie v slovenskom jazyku. Je však dôležité, aby výrazová sila jazyka nebola príliš obmedzená jeho jednoduchosťou. Väčšia výrazová sila umožňuje vytvoriť rozdielne typy úloh, v ktorých sa programátor oboznámi s rôznymi princípmi programovania využívanými aj v iných programovacích jazykoch. Programátor zadáva inštrukcie pomocou klávesnice. Inštrukcie nie je potrebné písať do rôznych riadkov ani oddeľovať špeciálnymi znakmi, stačí medzera.

3.4.1 Základné inštrukcie

Základné inštrukcie, ktorými programátor ovláda robota, sú veľmi podobné tým v originálnej verzii. Sú to inštrukcie pre presun robota dopredu o jedno políčko (krok), otočenie robota (otoc), položenie tehličky danej farby (poloz farba) a zdvihnutie tehličky zo zeme (zober). Robot môže zdvihnúť len najvrchnejšiu tehličku z políčka, na ktorom stojí. Rovnako môže položiť tehličku len na vrch ostatných tehličiek položených na políčku, kde stojí, prípadne na podlahu, pokiaľ sa na tomto políčku žiadne tehličky nenachádzajú, položí ju na podlahu.

3.4.2 Podmienky

Podmienky a možnosť vetviť program sú dôležitou súčasťou takmer každého programovacieho jazyka. Vetvenie sme využili pri tvorbe úloh, v ktorých robot nejakým spôsobom reaguje na prostredie. Je dôležité, aby vedel programátor zistiť, či robot stojí na nejakej zafarbenej tehličke. Ak áno, chceme, aby bolo možné zistiť, akej farby táto tehlička je. To umožní robotovi zbierať tehličky len istej farby a podobne. Tehličky, ktoré robot pozbiera, môže znova položiť. Preto je nutné, aby existoval spôsob, ako zistiť, či robot má alebo nemá tehličku nejakej farby. Taktiež sme sa rozhodli umožniť programátorovi zistiť, do akej strany je robot otočený. Táto schopnosť môže byť

užitočná pri riešení úloh, v ktorých programátor robota naviguje po svete. Bez nej by programátor nevedel otočiť robota do konkrétnej strany, keďže robot sa otáča vždy len doľava. Pri pohybe robota po svete je taktiež nutné overiť, či sa robot vie dostať na nasledujúce políčko. Pred robotom môže byť diera, jama (na políčku je položený malý počet tehličiek a robot naň nevie zoskočiť) a stena (na políčku je položený veľký počet tehličiek a robot naň nevie vyskočiť). Tieto logické výrazy je možné spájať pomocou logických spojok a, alebo.

3.4.3 Cykly

V úlohách sa často opakujú niektoré činnosti robota. V našom programovacom jazyku umožňujeme vytvárať dva typy cyklov: cyklus s pevným počtom opakovaní (opakuj) a cyklus s podmienkou na začiatku (pokial). Jednoduchých príkladom, kedy je využitie cyklu s pevným počtom opakovaní takmer nevyhnutné, je presun robota o viacero políčok dopredu. Cyklus s podmienkou na začiatku sa dá tiež, okrem iného, využiť, ak chceme, aby sa robot hýbal, kým sa vie dostať na nasledujúce políčko. Vďaka cyklom vieme vytvoriť úlohy, ktoré vie programátor riešiť aj bez toho, aby presne poznal, ako vyzerá svet, v ktorom sa robot nachádza a písal presný počet príkazov, ktoré je nutné vykonať na vyriešenie úlohy. Cykly je možné prerušiť pomocou inštrukcie preruš.

3.4.4 Premenné

V našom jazyku umožňujeme ukladať dáta do premenných. Premenné môžu byť lokálne aj globálne. Globálne premenné sú určené pomocou kľúčového slova vsade. Názvy premenných určujú ich typ. Premenné môžu byť štyroch typov.

číslo Názvy premenných musia začínať reťazcom 'cislo-', obsahujú celočíselnú hodnotu. Hodnota neinicializovanej premennej je 0.

farba Názvy premenných musia začínať reťazcom 'farba-', nadobúdajú tri hodnoty: cervena, zelena a modra. Hodnota neinicializovanej premennej je cervena.

pravdivosť Názvy premenných musia začínať reťazcom 'pravdivosť-', nadobúdajú dve hodnoty: pravda a nepravda. Hodnota neinicializovanej premennej je nepravda.

smer Názvy premenných musia začínať reťazcom 'smer-', nadobúdajú štyri hodnoty: vpravo, hore, vlavo a dole. Hodnota neinicializovanej premennej je vpravo.

3.4.5 Iné

Okrem už spomenutých vecí je možné používať základné aritmetické operácie na číslach, prípadne číselných premenných. Premenné a konštanty toho istého typu sa dajú

porovnávať.

3.4.6 Vlastné funkcie

Programátor si môže vytvoriť vlastné funkcie. Funkcie je možné vytvoriť a volať aj s parametrami. Keďže jazyk obsahuje len malý počet základných inštrukcií, niektoré časté činnosti robota, ktoré vyžadujú väčší počet inštrukcií, si programátor može definovať ako vlastnú funkciu. Príkladom takejto pomocnej funkcie môže byť už predtým spomínaný príkaz na otočenie robota doprava, poprípade do smeru, ktorý funkcia dostane ako parameter. Medzi ďalšie pomocné funkcie môžeme zaradiť funkciu, ktorá pozbiera všetky tehličky pod robotom. Taktiež je jednoduchšie vytvoriť sériu úloh, kde nasledujúca úloha je rozšírením tej predchádzajúcej. Funkcie definované v predchádzajúcej úlohe sa dajú znovu použiť, kód je prehľadnejší a ľahko upraviteľný.

3.5 Úlohy

Úlohy sú zoradené do viacerých kategórií. Každá kategória ma za úlohu vysvetliť a precvičiť niektorý z vyššie popísaných prvkov nášho programovacieho jazyka. Prvé úlohy oboznámia programátora so základnými inštrukciami na triviálnych problémoch. Náročnosť úloh sa postupne zvyšuje. Pri návrhu úloh bolo dôležité, aby žiadna úloha nevyžadovala znalosť vecí, ktoré ešte neboli vysvetlené a zároveň, aby každá úloha prinášala len toľko nových informácií, koľko dokáže programátor pri riešení jednej úlohy pochopiť. Nadobudnuté vedomosti potom programátor využíva v ďalších úlohách, takže si ich neustále opakuje. Programátor sa môže stretnúť pri riešení úloh s niektorými úlohami viackrát. Jedná sa o úlohy, ktoré sa dajú riešiť viacerými spôsobmi. Jednou z týchto úloh je úloha schody. Ako názov napovedá, cieľom v tejto úlohe je postaviť schody z tehličiek. Pri prvom stretnutí s touto úlohou programátor ovláda len základné príkazy. Keď na úlohu narazí druhý raz, ovláda už aj cykly. To, že úlohu už predtým riešil, minimalizuje čas strávený snahou pochopiť zadanie. Taktiež poukáže na výhody používania novonaučenej veci, v tomto prípade cyklu. Programátor už nemusí pre každú jednu tehličku na napísať príkaz pre jej položenie, stačí napísať príkaz opakuj. Tretíkrát vyskúša túto úlohu vyriešiť pri učení premenných. Zistí, že nie je potrebné písať samostatný cyklus pre každé políčko mapy, ale dajú sa použiť dva cykly, pričom v premennej mám uložené, koľko tehličiek mám na každom políčku položiť.

Kapitola 4

Použité technológie

Táto kapitola je venovaná popisu technológií a knižníc využitých pri tvorbe aplikácie.

4.1 HTML5, CSS3, Javascript

Jednou z požiadaviek na našu aplikáciu je, aby bola spustiteľná bez nutnosti inštalácie. Webová aplikácia je preto najlepším riešením. Technológie HTML5, CSS3 a Javascript sú v súčasnosti plne podporované všetkými modernejšími webovými prehliadačmi bez nutnosti inštalácie ďalších doplnkov. [10]

4.2 Three.js

Three.js je názov javascriptovej knižnice umožňujúcej zobrazovanie trojrozmernej počítačovej grafiky vo webovom prehliadači. Three.js umožňuje vykresľovanie pomocou technológie WebGL a taktiež pomocou HTML prvku <canvas>, čo zaručuje širokú podporu prehliadačmi. Zdrojové kódy knižnice je možné nájsť v oficiálnom repozitári na GitHube. [6] Túto knižnicu využívame na vykresľovanie prostredia, v ktorom sa robot nachádza a taktiež na zobrazenie samotného robota. Three.js podporuje načítanie modelov vytvorených v programe Blender.

4.3 Ace

Ace je editor kódu naprogramovaný v jazyku Javascript, ktorý je možné vložiť na stránku. Editor podporuje zvýraznenie syntaxe viacerých programovacích jazykov. Taktiež je možné vytvoriť vlastné pravidlá pre zvýrazňovanie. Medzi jeho ďalšie užitočné schopnosti môžeme zaradiť automatické doplňovanie kľúčových slov, párovanie zátvoriek, možnosť kopírovať, vystrihnúť a vložiť text a podobne. Ace je dostupný na oficiálnej stránke projektu[4].

4.4 Blender

Blender je voľne dostupný softvér na tvorbu trojrozmernej počítačovej grafiky. Je možné ho získať priamo z oficiálnej stránky[2]. V tomto programe sme vytvorili model robota a taktiež modely tehličiek nachádzajúcich sa vo svete. Modely sme exportovali vo formáte JSON pomocou doplnku z knižnice Three.js pre Blender. Na modely je priamo aplikovaná aj textúra. Export animácií je tiež podporovaný[6].

4.5 Jison

Kód, ktorý programátor napíše, je nutné pred jeho vykonaním spracovať parserom. Jison sme použili na vytvorenie tohto parsera. Vstupom pre Jison je bezkontextová gramatika a výstupom program v javascripte, ktorý dokáže spracovať jazyk popísaný touto gramatikou. Vďaka tomuto vytvorenému programu vieme overiť, či programátorom napísaný kód je syntakticky správny a v prípade, že áno, vieme tento kód vykonať. [3] Na vytvorenie parsera sme použili webové rozhranie priamo na stránke Jisonu.

4.6 Ďalšie knižnice

Medzi ďalšie knižnice, ktoré sme využili v menšej miere, patrí jQuery. jQuery uľahčuje programátorovi spojenie HTML s javascriptom. Knižnicu je možné voľne získať z oficiálneho repozitára na GitHube[1]. Font Awesome je framework pre CSS. Poskytuje vektorové ikonky použiteľné na webstránke [5]. Tieto ikonky sme použili pre tlačidlá v našej aplikácii.

Kapitola 5

Návrh implementácie a implementácia

V tejto kapitole podrobne popíšeme implementačné detaily našej aplikácie a problémy, ktoré sme museli riešiť.

5.1 Gramatika jazyka

Ako sme už spomenuli v prechádzajúcej kapitole, programovací jazyk, v ktorom programátor rieši úlohy v našej aplikácii, je popísaný bezkontextovou gramatikou. Gramatika sa nachádza v súbore language.yy. Pri tvorbe gramatiky bolo náročné vyhnúť sa konfliktom. Konflikty môžu byť dvoch typov: shift/reduce a reduce/reduce[9].

Prvý spomenutý konflikt nastal, keď sme neurčili správne priority a asociatívnosť aritmetických a logických operátorov. Tento problém sa tiež vyskytol pri podmienke AK TAK INAK (IF THEN ELSE). Problém nastane, ak programátor napíše dve podmienky za sebou, pričom použije len jednu vetvu INAK. V tomto prípade nevieme povedať, ku ktorej podmienke vetva INAK patrí. Možností, ako riešiť tento problém je viacero. Jednou z možností je priradiť vetvu INAK k najbližšej podmienke. Iné riešenie je špeciálne kľúčové slovo, ktoré by sa písalo na koniec štruktúry AK a tým ju uzatvorilo. My sme sa rozhodli pre prvé spomenuté riešenie. Nastavenie väčšej priority pre INAK ako pre AK spôsobí, že vetva pre INAK bude priradená k najbližšiemu AK.

Druhý spomenutý konflikt nastáva pri zlom návrhu gramatiky, kedy sa dajú uplatniť dve alebo viaceré pravidlá gramatiky. Na tento problém sme narazili, keď sme sa pokúšali o možnosť pretypovania konštánt a premenných na iný typ. Konkrétne typ číslo na pravdivosť a naopak. Rozhodli sme sa, že možnosť pretypovania nie je v našom jazyku vôbec potrebná a tak tento problém nebolo nutné ďalej riešiť. Neskôr sme sa snažili, aby bolo možné dať premenným ľubovoľné meno a typ premennej sa odvodil pri vykonávaní programu podľa kontextu, v ktorom je použitá. Problém nastal pri logickej operácii porovnania. Jedným riešením bolo použitie špeciálnych znakov alebo kľúčových slov na získanie hodnoty premennej, čo by však výrazne skomplikovalo prog-

ramovací jazyk. Problém sa však dal vyriešiť aj úpravou gramatiky, zmeny by ju však veľmi skomplikovali. Riešenie, kde premenné rôznych typov majú vyhradené mená, tento problém rieši a podľa nášho názoru aj sprehľadňuje a zjednodušuje pochopenie napísaného zdrojového kóde za cenu písania zopár znakov navyše.

5.2 Parser

Parser sa nachádza v súbore language.js. V predchádzajúcej kapitole sme spomenuli, že parser je vygenerovaný pomocou Jisonu. Pri parsovaní zdrojového kódu je však možné vykonávať nejaké akcie. Po sparsovaní zdrojového kódu vráti parser javascriptový objekt (Object). Tento objekt sa vytváral počas parsovania pri uplatnení každého pravidla gramatiky. Objekt obsahuje dve hlavné properties. Property __main__, ktorej hodnotou je objekt obsahujúci všetky napísané príkazy mimo definícií vlastných funkcií (príkazov) uložené v poli. Property __globalvars__, ktorej hodnotou sú príkazy na definované globálnych premenných uložených v poli.

Ďalšie property sú mená vlastne definovaných funkcií, ich hodnotou sú objekty, ktoré obsahujú príkazy v týchto funkciách uložené v poli, mená týchto funkcií a ich parametre. Každý príkaz je objekt, ktorý má jednu povinnú property, a to je command. Command určuje, o aký príkaz ide. Ďalšia nepovinná property je source, ktorá hovorí, kde v zdrojovom kóde sa tento príkaz nachádza. Ostatné príkazy môžu mať vlastné špecifické property.

Príkaz na pridelenie hodnoty do premennej obsahuje informácie o tom, aké je meno tejto premennej, či je globálna a hodnotu, ktorú má do tejto premennej priradiť. Informáciu o tom, o aký typ premennej ide, neobsahuje, pretože každý typ premennej má iný príkaz na pridelenie hodnoty do nej (z pohľadu používateľa aplikácie je rovnaký). Akú hodnotu je potrebné premennej priradiť je v skutočnosti funkcia, ktorá túto hodnotu vypočíta, pretože ju chceme zistiť, až keď sa inštrukcia priradenia vykonáva. To platí aj pri ostatných príkazoch, kde sa ukladá nejaká hodnota do property.

Pri podmienke AK je nutné si pamätať, kde začína vetva INAK, prípadne kde končí vetva TAK, pokiaľ vetva INAK neexistuje. Túto informáciu potrebujeme, aby sme pri interpretovaní kódu vedeli, kam sa máme premiestniť, pokiaľ nie je podmienka splnená, pretože príkazy vo vetvách sa nachádzajú v tom istom poli, ako príkaz AK. Túto hodnotu zistíme súčtom pozície v poli príkazu AK a počtom príkazov nachádzajúcich sa vo vetve TAK plus jedna.

Pri cykloch potrebujeme zaručiť, že sa na konci cyklu vieme dostať na jeho začiatok. Dosiahneme to pridaním príkazu jump na koniec cyklu. Jump obsahuje property, ktorá hovorí, na aký index v poli sa treba premiestniť po vykonaní tejto inštrukcie. V tomto prípade je to začiatok cyklu. Túto hodnotu zistíme podobným spôsobom ako pri

podmienke AK. Vykonávanie cyklu však vieme aj prerušiť. Mohli by sme použiť príkaz jump, tu by však mohli nastať problémy. Ak by sme skočili na začiatok cyklu, nevedeli by sme rozoznať, či máme v cykle pokračovať, pretože sme sa dostali na koniec alebo máme cyklus prerušiť. Pri skoku na koniec zas nemôžme vykonať potrebné operácie v interpreteri. Napríklad pri cykle opakuj si pamätáme, koľkokrát sa má cyklus ešte vykonať. V prípade vnorených cyklov je túto hodnotu pri prerušení cyklu potrebné obnoviť. Preto na prerušenie cyklu musíme použiť iný príkaz. Taktiež musíme prejsť cez všetky príkazy, ktoré sú súčasťou daného cyklu a hľadať príkaz na prerušenie cyklu, aby sme mu mohli nastaviť, ktorý cyklus má prerušiť. Nastavujeme však len také príkazy na prerušenie, ktoré ešte neboli nastavené, pretože pri vnorených cykloch sa najskôr dokončí spracovanie toho vnútorného.

```
| WHILE condition THEN command
{
    $$ = [{
        'command' : 'while',
        'value' : $2,
        'end' : $4.length+2,
        'source' : {
            'first_line'   : @1.first_line,
            'first_column' : @1.first_column,
            'last_line'    : @3.last_line,
            'last_column' : @3.last_column
        }
    }].concat($4);
    for (var i=0; i<$4.length; i++) if (($4[i].command=='break')&&($4[i].value==0))
        $4[i].value=-i-1;
    $$.push({'command' : 'jump', 'value': -$4.length-1});
}
```

Obr. 5.1: Spracovanie príkazu pokial. Najskôr sa vytvorí pole, ktoré obsahuje len príkaz while. Potom toto pole spojí s polom obsahujúcim všetky príkazy, ktoré sa vykonávajú, len ak je podmienka splnená. Nakoniec sa nastavia všetky ešte nenastavené príkazy prerus a pridá sa príkaz jump

5.3 Robot

Objekt robot zo súboru robot.js reprezentuje robota. Sú v ňom uložené informácie o pozícii a rotácii. Ako už bolo spomenuté, robot zbiera tehličky. Informácie o tom, aké tehličky má pri sebe, sú uložené v objekte bag implementovanom v súbore bag.js. Základný objekt bag funguje ako klasický batoh. Robot do neho vie uložiť hocijakú tehličku a vie z neho vybrať hocijakú tehličku, ktorá sa v batohu nachádza.
Metódy objektu bag:

- **hasBrick()** Zistí, či má robot pri sebe aspoň jednu tehličku ľubovoľnej farby.
- **hasColor(color)** Zistí, či má robot pri sebe aspoň jednu tehličku farby, ktorá je argumentom metódy.

- **takeBrick(color)** Vymaže z batohu tehličku farby, ktorá je argumentom metódy.
- **putBrick(color)** Vloží do batohu tehličku farby, ktorá je argumentom metódy.

Implementovali sme aj iné typy batohov, ktorá sa dajú využiť v špecifických úlohách. Nekonečný batoh obsahuje nekonečne veľa tehličiek každej farby. Zo zásobníkového batohu je možné vyberať len poslednú vloženú tehličku. Na otázku, či má tehličku istej farby odpovedá pravdivo len ak bola tehlička tejto farby vložená do batohu ako posledná. Niektoré metódy objektu robot:

build(x,y,z,o,bag,maxjumpup,maxjumpdown) Vytvorí robota na súradniciach x a y vo svete a položí ho do výšky z, kde o je smer robota, bag batoh s tehličkami a maxjumpup a maxjumpdown určujú, akú výškovú vzdialenosť vie robot prekonať pri kroku dopredu, ak musí vyskočiť hore, respektíve skočiť dole.

nextPos() Vráti pozíciu robota, ak by spravil krok dopredu.

currentPos() Vráti aktuálnu pozíciu robota.

step() Posunie robota o políčko dopredu v smere, v ktorom je otočený.

turn() Otočí robota doľava.

createMesh (scene) Vytvorí robota a pridá ho do scény z argumentu, aby mohol byť zobrazený. Taktiež vytvorí animácie.

playAnimation(anim, speed) Spustí animáciu modelu robota. Anim je meno animácie, speed je rýchlosť prehrávania animácie.

stopAnimation(anim) Zastaví animáciu modelu robota. Anim je meno animácie, ktorá sa má zastaviť.

5.4 Mapa

Objekt world zo súboru world.js reprezentuje svet, v ktorom sa robot nachádza a pohybuje. Sú v ňom uložené informácie o polohe tehličiek, a tiež o tom, kde sa robot môže pohnúť a naopak, kde vo svete sa nachádzajú diery. Informácie sú uložené v trojrozmernom poli map. Niektoré metódy objektu mapa:

- **build(x,y)** Vytvorí svet s rozmermi x a y. V tomto svete sa nebudú nachádzať žiadne tehly. Každé políčko však bude mať tehličku a robot sa naň vie dostať.
- **isHole(x,y)** Zistí, či sa vo svete na políčku so súradnicami x a y nachádza diera.
- **isBrick(x,y)** Zistí, či sa vo svete na danom políčku nachádza tehla.

- **isColor(x,y,color)** Zistí, či sa vo svete na danom políčku nachádza tehla danej farby. Ak sa na políčku nachádza viacej tehiel, do úvahy berieme len tehlu navrchu.
- **height(x, y)** Vráti počet tehličiek na danom políčku. Ak je toto políčko jama, vráti -1.
- **makeHoleAt(x, y)** Vymaže všetky tehličky z daného políčko a vytvorí na tomto políčku dieru.
- **putAt(x,y,item)** Na dané políčko pridá tehličku. Item je objekt s property type, ktorá má hodnoty 'brick' a property color, ktorá určuje farbu tehličky.
- **takeFrom(x,y)** Zoberie vrchnú tehličku z daného políčka a vymaže ju.
- **setScene(scene)** Nastaví scénu, v ktorej sa má svet zobraziť.
- **syncScene()** Synchronizuje scénu a svet. Do scény pridá všetky tehličky a podlahu.

5.5 Interpeter

Objekt Interpeter, nachádzajúci sa v súbore interpeter.js, slúži na interpretovanie kódu, ktorý programátor napísal. Na svoju prácu potrebuje objekty robota a mapy. Interpreter spúšťa kód vo vykonávacích rámcoch (execution frames). Tieto rámce si udržiava v zásobníku. Rámec je objekt, ktorý obsahuje časť kódu, ktorá sa práve vykonáva (v našom prípade pole príkazov), lokálne premenné a ďalšie informácie potrebné na vykonávanie programu. Po sparsovaní kódu parser vráti objekt, ktorý budeme volať program. Z property __globalvars__ inicializuje globálne premenné. Program vždy obsahuje property __main__. Pre __main__ sa vytvorí nový rámec a uloží sa do zásobníku. Pri volaní programátorom definovanej funkcie sa pre túto funkciu vytvorí nový rámec a vloží sa na vrch zásobníku. Po vykonaní všetkých príkazov v rámci sa tento rámec odstráni zo zásobníka. V každom rámci si pamätáme index príkazu z poľa príkazov, na ktorom sa práve nachádzame. Tiež je nudné mať uložené počítadlá pre cyklus opakuj. Tieto počítadlá sú uložené v objekte, kde property je pozícia cyklu v poli príkazov a hodnota tejto property určuje, koľkokrát sa má cyklus ešte vykonať.
Popis dôležitých metód objektu interpreter:

- **loadProgram(code)** Parametrom je zdrojový kód programu. Metóda sparsuje kód pomocou parseru a výsledný objekt uloží. Potom zavolá metódu restart.
- **restart()** Umožní spustenie programu od začiatku. Vymaže zásobník s rámcami a vloží doň nový rámec pre __main__.

- **next()** Vykoná jeden príkaz a prejde na nasledujúci. Zo zásobníka vyberie (bez vymazania) vrchný rámec a zistí, ako akom príkaze sa nachádza. Podľa typu príkazu vykoná, čo je potrebné (ale iba v rámci interpreteru, robota a mapu nemení) a presunie sa na ďalšiu príkaz. Metóda vráti vykonaný príkaz a informácie o prípadných chybách.

Medzi ďalšie metódy patria metóda na zistenie hodnoty premennej, orientácie robota, informácie o mape a podobne.

```
case 'repeat' :
    //ak sme tu prvykrat, resp. ak uz bol tento repeat predtyp dokonceny
    if (!frame.repeats.hasOwnProperty(frame.i) || frame.repeats[frame.i]==-2) {
        if (command.infinite) frame.repeats[frame.i]=-1; //nekonecne repeat, pocitadlo bude -1
        else frame.repeats[frame.i]=Math.max(0, command.value(this)); //zaporne zmenime na 0
    }
    if (frame.repeats[frame.i]>0) { //repeat pokracuje
        frame.repeats[frame.i]--;
        frame.i++;
    }
    else if (frame.repeats[frame.i]==-1) frame.i++; //infinite loop
    else {
        //repeat je dokonceny, prejdime na koniec
        frame.repeats[frame.i]=-2;
        frame.i+=command.end;
    }
```

Obr. 5.2: Spracovanie príkazu opakuj(repeat) interpreterom. V aktuálnom rámci si pamätáme, koľkokrát musíme cyklus ešte zopakovať.

5.6 Scéna

Objekt Scene v súbore scene.js implementuje zobrazenie sveta s robotom a načítanie modelov do programu. Oboje s využitím knižnice Three.js. Pri vytvorení objektu scéna sa vytvorí renderer a zobrazí sa na stránke. Do sveta sa pridajú svetlá a skybox. Modely sa načítavajú asynchrónne. Po načítaní všetkých modelov sa zavolá funkcia, ktorú scéna dostane ako parameter pri vytvorení. Vykresľovanie prebieha v dvoch fázach. Najskôr sa vykreslí skybox, potom svet. To zaručí, že celý svet je vždy viditeľný a nie je zakrytý skyboxom.

5.7 Hlavná slučka

Objekt Executor implementovaný v súbore mainloop.js ovláda chod programu. Je v ňom uložená informácia o tom, akú akciu aktuálne robot vykonáva a kedy sa má táto akcia skončiť. Po skončení akcie vyberie novú akciu a pridelí ju robotovi. Vytvorenie tohto objektu spôsobí vytvorenie scény. Metódy objektu Executor:

- **restart()** Program na ovládanie robota sa má spustiť odznova. Zastaví akciu, ktorú robot aktuálne vykonáva.

- **run()** Program na ovládanie robota sa má spustiť celý bez zastavenia. Keď robot dokončí príkaz, hneď začne ďalší.

- **pause()** Pozastaví program na ovládanie robota. Robot nezačne vykonávať ďalší príkaz po dokončení aktuálnej akcie.

- **step()** Prikáže robotovi vykonať jeden príkaz.

- **setAnimationSpeed(speed)** Nastaví, ako rýchlo sa majú akcie robota vykonávať. Speed je konštanta, ktorou sa vydelí základný čas akcie. Výsledná hodnota je skutočné trvanie akcie.

- **mainLoop()** Hlavná metóda. V prípade, že robot dokončil akciu a je potrebné vykonať novú akciu, zavolá metódu next objektu interpreter a začne vykonávať novú akciu podľa príkazu, ktorý interpreter vrátil. Ak je program na ovládanie robota dokončený a robot už nemusí vykonať žiaden príkaz, skontroluje sa splnenie úlohy, ktorú programátor rieši. Taktiež sa tu vykonávajú potrebné úkony pre jednotlivé akcie robota. Napríklad, ak sa robot otáča, pri každom volaní robota otočíme o istý uhol. Po dokončení akcie bude robot otočený o 90 stupňov doľava. Každé volanie metódy mainLoop spôsobí nové vykreslenie scény. Opätovné volanie metódy zabezpečuje funkcia requestAnimationFrame.

5.8 Úlohy

Objekt Level zo súboru level.js zabezpečuje vytvorenie sveta a robota pre každú úlohu a kontrolu, či je úloha správne vyriešená. Každá úloha je odvodený objekt z objektu Level. Úloha musí definovať nasledujúce metódy:

- **build(world, robot)** Vytvorí svet a robota. Do sveta je možné umiestniť tehličky, vytvoriť na niektorých políčkach diery a podobne. Robotovi je potrebné nastaviť pozíciu a jeho smer, prípadne mu dať nejaké zafarbené tehličky, ktoré je možné položiť.

- **check((world, robot))** Skontroluje, či je úloha správne vyriešená. Ak áno, vráti true, inak false spolu so správou, prečo úloha nebola správne splnená.

- **next()** Vykoná jeden príkaz a prejde na nasledujúci. Zo zásobníka vyberie (bez vymazania) vrchný rámec a zistí, na akom príkaze sa nachádza. Podľa typu príkazu vykoná, čo je potrebné (ale iba v rámci interpreteru, robota a mapu nemení) a presunie sa na ďalší príkaz. Metóda vráti vykonaný príkaz a informácie o prípadných chybách.

Tiež je potrebné mať pre každú úlohu iný text zadania. Na to slúži property taskHTML. Okrem týchto metód má objekt Level dve ďalšie metódy:

start() Spustí úlohu. Zmení text zadania úlohy na stránke na text zadania aktuálnej úlohy. Zavolá metódu build aktuálneho levelu.

verify() Táto metóda je volaná z hlavnej slučky, keď skončí vykonávanie programu pre robota. Metóda volá metódu check aktuálneho levelu. Zobrazí okno s informáciou o tom, či bola úloha správne splnená, prípadne vypíše dôvod, prečo to tak nie je.

5.9 Tlačidlá

Aplikácia je ovládaná pomocou viacerých tlačidiel. Tlačidlá nad editorom slúžia na spúšťanie programu pre robota. Prvé tlačidlo zľava slúži na sparsovanie kódu parserom. Po kliknutí na toto tlačidlo sa zavolá metóda loadProgram objektu Interpeter. Potom je možné pomocou zvyšných tlačidiel ovládať beh programu pre robota. Kliknutím na jednotlivé tlačidlá sa volajú metódy objektu Exucutor. Ďalšie tlačidlá sú tlačidlá na výber úlohy. Po kliknutí na jedno z týchto tlačidiel sa vytvorí nový objekt pre daný level a zavolá sa jeho metóda start.

Záver

Výsledkom práce je aplikácia umožňujúca riešenie úloh v modifikovanom jazyku Karel. Aplikácia je prístupná prostredníctvom webovej stránky `http://www.st.fmph.uniba.sk/~zbin1/web/`. Na používanie aplikácie je potrebný len webový prehliadač, žiadna ďalšia inštalácia softvéru nie je potrebná. V jednotlivých úlohách používateľ ovláda robota pomocou programov, ktoré píše v nami navrhnutom programovacom jazyku. Robot je zobrazený ako trojrozmerný model, v súčasnosti však bez animácií jednotlivých činností. Používatelia si môžu vyberať z viacerých úloh rôznych kategórií, ktoré slúžia hlavne na oboznámenie s programovacím jazykom.

Aplikáciu mali možnosť vyskúšať si dvaja žiaci z osemročného gymnázia v Púchove. Vzhľadom na to, že toto testovanie nemá príliš veľkú výpovednú hodnotu kvôli malému počtu účastníkov, sa mu nevenujeme v samostatnej kapitole. Žiaci mali pred týmto testom len malé skúsenosti s programovaním, konkrétne v programe Imagine a programovacom jazyku Pascal. Aplikácia ich zaujala, mali pocit, že hrajú hru. Za vymedzený čas sa im podarilo vyriešiť všetky úlohy z prvých troch kategórií úloh.

Zdrojový kód aplikácie je voľne dostupný. To poskytuje možnosť rozširovať funkcionalitu našej aplikácie. Sme si vedomí, že sa aktuálne v aplikácii nenachádza dostatok úloh na riešenie. Je však možné ich jednoducho vytvárať a pridávať na stránku. Medzi ďalšie rozšírenia, ktoré môžu viacej zatraktívniť aplikáciu, patrí zobrazovanie animácií robota. Zobrazovania animácií je v aplikácii podporované, je však potrebné ich vytvoriť pre každú činnosť robota v Blenderi, alebo inom programe na tom určenom. Príkazy programovacieho jazyka sú v slovenskom jazyku. V prípade, ak by bol záujem používať aplikáciu žiakmi neovládajúcimi slovenčinu, je príkazy možné preložiť do iných jazykov len prepísaním ich názvov a vygenerovaním nového parsera.

Literatúra

[1] jquery — new wave javascript. [online] Dostupné na internete: https://github.com/jquery/jquery. Citované: 2017-05-12.

[2] Oficiálna stránka aplikácie blender. [online] Dostupné na internete: https://www.blender.org/. Citované: 2017-05-12.

[3] Oficiálna stránka dokumentácie projektu jison. [online] Dostupné na internete: https://zaa.ch/jison/docs/. Citované: 2017-05-12.

[4] Oficiálna stránka projektu ace. [online] Dostupné na internete: https://ace.c9.io/. Citované: 2017-05-14.

[5] Oficiálna stránka projektu font awesome. [online] Dostupné na internete: http://fontawesome.io/. Citované: 2017-05-12.

[6] Oficiálna stránka projektu three.js. [online] Dostupné na internete: https://github.com/mrdoob/three.js/. Citované: 2017-05-12.

[7] Oficiálna stránka prostredia scratch. [online] Dostupné na internete: https://scratch.mit.edu/. Citované: 2017-05-14.

[8] Robot karol ++ v1.0. [online] Dostupné na internete: http://tahaj.sme.sk/software/128. Citované: 2017-05-10.

[9] Techniques for resolving common grammar conflicts in parsers. [online] Dostupné na internete: https://efxa.org/2014/05/17/techniques-for-resolving-common-grammar-conflicts-in-parsers/. Citované: 2017-05-13.

[10] David Flanagan. *JavaScript: The Definitive Guide (6th ed.)*. O'Reilly Media, 2011.

[11] PhD. RNDr. Andrej Blaho, doc. RNDr. Ivan Kalaš. *1. zošit z programovania*. Slovenské pedagogické nakladateľstvo - Mladé letá, s.r.o., 2005.

[12] Richard E. Pattis, Jim A. Roberts, Mark Stehlik. *Karel the robot: a gentle introduction to the art of programming.* Wiley, 1995.

Príloha

Súčasťou práce je elektronická príloha obsahujúca zdrojové súbory aplikácie. Aktuálna zdrojové súbory sa dajú nájsť aj v repozitári projektu na GitHube na adrese `https://github.com/AndrejZbin/bakalarka`.