VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

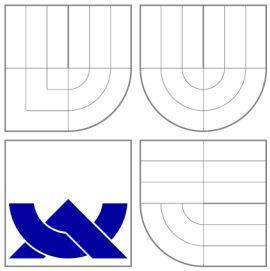
GRAFICKÝ VÝUKOVÝ SYSTÉM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

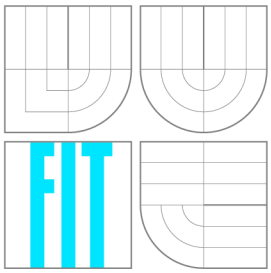
AUTOR PRÁCE
AUTHOR

Bc. ROMAN HOTAŘ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÝ VÝUKOVÝ SYSTÉM

GRAPHICAL EDUCATIONAL SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ROMAN HOTAŘ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍT ŠTANCL

BRNO 2010

Abstrakt

Účelem této diplomové práce je přiblížit problematiku návrhu výukového software pro výuku algoritmizace. Práce přistupuje k problému, jak z teoretického hlediska vzdělávání se a učení se novým věcem, tak z praktického pohledu důležitého pro osvojení si procesu algoritmizace. Zabývá se také stavem výuky algoritmizace na českých školách a popisuje výsledný program, který byl v rámci diplomového projektu vytvořen. Přínos této práce spočívá ve shrnutí požadavků, kladených na daný software a implementaci softwaru tohoto typu, na kterém lze názorně demonstrovat problémy a požadavky návrhu programů tohoto druhu. V práci jsou také popsány reakce pedagogů na vytvořenou aplikaci.

Abstract

The purpose of this thesis is the proposal of the issue of education software for learning algorithms. Work approach to the problem, both from a theoretical point of view, of education and learning to new things, so and more important from a practical point of view for introduction to the process algorithmization. It also discusses the problem of learning algorithms in Czech schools and describes the resulting program, which was created along with this thesis. Contribution of this thesis is the summary of the requirements for the software and implementing software of this type, where is possible to demonstrate the problems and requirements of the draft programs of this kind. This thesis also describes the reactions of teachers to created application.

Klíčová slova

Algoritmy, výuka algoritmizace, knihovna Qt, grafický výukový systém, multiplatformní aplikace, výukový program, uživatelské rozhraní, základy programování, jazyk Lua, teorie výuky, didaktické zásady.

Keywords

Algorithms, learning algorithms, Qt library, graphical education system, multiplatform application, education system, user interface, programming basis, Lua language, learning theory, didactics principles.

Citace

Roman Hotař: Grafický výukový systém, diplomová práce, Brno, FIT VUT v Brně, 2010

Grafický výukový systém

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Víta Štancla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Roman Hotař
26. května 2010

Poděkování

Zde bych chtěl poděkovat vedoucímu mé diplomové práce Ing. Vítovi Štanclovi za ochotu a pomoc při tvorbě diplomové práce a také panu Petru Naske, který mi ochotně pomohl při vyplňování dotazníku.

Obsah

Kapitola 1

Úvod

Problém algoritmizace byl řešen lidmi ještě před prvními počítači, ovšem nikoliv v takové podobě jako dnes. Chceme-li být důslední, lze tvrdit, že za algoritmus můžeme považovat jakýkoli návod jak něco vyrobit, sestavit, nebo uvařit.

Nejstarší známé matematické algoritmy vymysleli Babyloňané již 1600 let před našim letopočtem. Jednalo se o algoritmus faktorizace a nalezení kořenů kvadratické rovnice. Postupně byly vymýšleny další a další algoritmy popisující většinou řešení matematických úloh. Některé z nich jako Eukleidův algoritmus (300 př. n. l.) a Eratosthenovo síto (200 př. n. l.), jsou s oblibou používány dodnes, ať pro výukové účely díky své jednoduchosti, nebo pro řešení úkolů v praxi.

Díky nezadržitelnému vývoji techniky na konci druhé poloviny dvacátého století a představení prvních počítačů, vzrůstala potřeba rozvoje algoritmizace, a ta se postupně stávala synonymem slova programování, které označuje převod algoritmu do podoby srozumitelné pro počítač.

První počítače byly využitelné pouze pro výzkumné účely, protože jejich nasazení při řešení reálných problémů bylo příliš komplikované a přinášelo více problémů, než užitku. Se vzrůstající výkonností prvních strojů a zdokonalením jejich vlastností se však nasazení do praxe stávalo čím dál tím reálnější. Díky nástupu osmibitových počítačů se počítače dostaly i do domácností a staly se postupně součástí života každého z nás, aniž bychom si to uvědomili. V dnešní době se s počítači setkáváme v jejich klasické podobě stolních počítačů, přenosných notebooků a jejich variantami, nebo také u vestavěných zařízení v automobilech, domácích spotřebičích a mobilních telefonech. Všechny tyto počítače jsou řízeny nějakým programem, který používá složitější, nebo jednodušší algoritmy pro zajištění správné činnosti počítače.

Společně s vývojem počítačů se vyvíjely také programovací techniky, které byly pro jejich programování používány, zatímco u prvních počítačů se programovalo poměrně složitě děrnými štítky, nebo ručním propojováním obvodů, později byl vyvinut efektivnější způsob programování, a to nejdříve zápisem programu v jazycích blízkých stroji a poté v jazycích vyšší úrovně, kterými dříve byly Fortran, Lisp, Cobol, nebo dnes používanější Basic, jazyk C nebo Pascal.

Zatímco tedy do doby před půl stoletím byla algoritmizace spojena většinou s řešením matematických problémů a vytvořené algoritmy používal člověk k výpočtu matematických úloh, v dnešní době je algoritmizace spojena více s vytvářením postupu, jak vyřešit obecný problém, a často je algoritmus zapsán v programovacím jazyku. Tento algoritmus reprezentovaný programem je pak vykonáván počítačem.

Algoritmizace a potažmo programování se společně s počítači dostaly od vědců a tech-

niků k laické veřejnosti, které stačí mít přístup k počítači a dostatek času a trpělivosti, aby se mohli pustit do základů algoritmizace.

Programování se již stalo součástí osnov technicky zaměřených vysokých a středních škol a proniká také do vyššího stupně základních škol, kde je dobrovolně vyučováno v zájmových kroužcích výpočetní techniky, zvláště zde je nutno řešit problém, jaký programovací jazyk zvolit, aby i ti nejmladší měli stejnou možnost vyjádřit svůj algoritmus programem. Tento trend výuky algoritmizace a programování již na základních školách vede žáky k logickému a systematickému řešení zadaných úkolů, čehož lze využít při řešení školních a také reálných problémů.

Začátky programování nebývají vůbec jednoduché a nemálo uživatelů odradí složitost zápisu programu v daném programovacím jazyce, syntaktická a sémantická pravidla, nutnost starat se o přidělenou paměť, nastudování často dlouhých manuálů k překladu programu a podobné počáteční problémy, které mohou vyvstat při osvojení si daného programovacího jazyka. Tento problém řeší grafické programovací jazyky, nebo obecněji jazyky pro výuku algoritmizace, které budou podrobněji rozebrány v jedné z pozdějších kapitol. Společným rysem těchto jazyků je fakt, že se snaží jejich uživatelům co nejvíce zjednodušit programování zastíněním problémů, které se řeší automaticky v pozadí a také tím, že poskytují připravené některé složité operace. Cílem těchto jazyků je tedy umožnit uživateli co nejpřesněji vyjádřit svůj algoritmus, aniž by strávil více času učením se programovacího jazyka.

Účelem této práce je podobné programy popsat, určit jejich společné vlastnosti, definovat požadavky kladené na programy a navrhnout grafický výukový systém, který je použitelný pro výuku algoritmizace. Podmínkou pro implementaci systému je jeho použitelnost na více platformách.

Práce nejdříve definuje základní pojmy algoritmizace a algoritmu, popisuje vlastnosti, které by měly dobré algoritmy splňovat, osvětluje jednotlivé způsoby zápisu algoritmů. Další kapitola této práce je věnována výuce algoritmizace, procesu učení, didaktickým zásadám, představení používaných aplikací pro výuku a jejich zhodnocení a shrnutí stavu výuky algoritmizace na základních a středních školách. V pořadí čtvrtá kapitola je pak věnována popisu použitých knihoven a nástrojů a za ní následující kapitola představuje implementovaný program a jeho návrh. Předposlední kapitola hodnotí dosažené výsledky na základě krátkého dotazníku a v poslední kapitole jsou shrnuty výsledky práce.

Kapitola 2

Algoritmus a algoritmizace

Na úvod bude potřeba definovat základní pojmy a teorii k algoritmizaci, které se budou v dalším textu hojně objevovat. Tuto kapitolu spolu s obrázky jsem převzal ze svého semestrálního projektu [2], na který práce navazuje.

Algoritmizace je proces tvorby algoritmů. Tato procedura vyžaduje správné logické uvažování a schopnost dekompozice problému na dílčí podproblémy.

Algoritmus pro tento elementární pojem existuje mnoho definic, použiji zde definici ze zdroje [1]. *Algoritmus je konečná uspořádaná množina úplně definovaných pravidel pro vyřešení zadaného problému. Intuitivně algoritmem rozumíme postup, který nás dovede k řešení úlohy. Formálněji vyjádřeno se jedná o přesně definovanou konečnou posloupnost příkazů (kroků), jejichž prováděním pro každé přípustné vstupní hodnoty, získáme po konečném počtu kroků, odpovídající výstupní hodnoty. Slovo "algoritmus" je odvozeno ze jména arabského učence, který se jmenoval Muhammad ibn Musa Abdallah Al Khowarizmi (Chorezmí) a žil na přelomu 8. a 9. století na území dnešního Uzbekistánu. Zasloužil se zejména o rozšíření algoritmů pro aritmetické operace v poziční soustavě (prakticky vytvořil systém arabských číslic). Zkomolením názvu jeho díla o řešení lineárních a kvadratických rovnic vzniklo slovo "algebra".*

2.1 Vlastnosti algoritmů

Korektní algoritmy by měly dle zdroje [1] splňovat následující vlastnosti: rezultativnost, hromadnost, jednoznačnost a efektivnost. Až na efektivnost jsou tyto vlastnosti velmi důležité a určují zda je algoritmus použitelný pro řešení problému. Význam jednotlivých vlastností a jejich vliv na algoritmus bude popsán dále.

Rezultativnost (konečnost), zaručuje vyřešení úlohy v konečném počtu kroků. Jinak řečeno, každý algoritmus, který tuto vlastnost splňuje, skončí za konečný čas. Tato vlastnost je pro algoritmus velmi podstatná. Pokud algoritmus není konečný, znamená to, že mohou nastat situace, kdy algoritmus neskončí a tím se stává nepoužitelným pro vyřešení dané úlohy.

Hromadnost někdy je tato vlastnost označována také jako obecnost. Algoritmus, který je hromadný (obecný) musí být navrhnut tak, aby jej bylo možno použít na všechny úlohy podobného typu. Musí tedy správně vyřešit danou úlohu při dosazení jakékoli

hodnoty z množiny vstupních dat. Pokud by vytvořený algoritmus nesplňoval podmínku obecnosti, byl by použitelný pouze pro konstantní vstupní data. Takovýto algoritmus ztrácí význam a lze ho nahradit konstantním výstupem.

Jednoznačnost tato vlastnost stanovuje, že algoritmus je možné zapsat pouze konečným počtem jednoznačných pravidel. V každém kroku algoritmu je přesně stanoveno to, jak má algoritmus dále pokračovat. Při nedodržení jednoznačnosti by docházelo k situacím, kdy by se algoritmus nedeterministicky rozhodoval, jak dál pokračovat, což by znamenalo, že při zadání stejných vstupních dat bychom obdrželi různé výsledky.

Efektivnost na výsledky algoritmu nemá efektivnost žádný vliv, ovlivňuje ovšem to, jak dlouho se bude algoritmus vykonávat, nebo jaké bude mít nároky na zdroje prostředků. Algoritmus by tedy neměl provádět zbytečné operace, které nejsou nezbytné k dosažení výsledku. Určení efektivnosti a potažmo složitost algoritmů je poměrně sofistikovaný problém, který se stává složitějším společně se zvyšující se složitostí algoritmu, natož optimalizace algoritmů k dosažení dané složitosti.

2.2 Způsoby zápisu algoritmů

Existuje několik způsobů zápisu algoritmů. Principielně bychom je mohli rozdělit na dvě třídy, a to zápisy textové a grafické. Obě třídy zápisu jsou rovnocenné co do schopnosti vyjádření algoritmu, ovšem grafický zápis je většinou používán pro reprezentaci méně rozsáhlých algoritmů, protože u rozsáhlejších se stává nepřehledným, pro tyto algoritmy je používán spíše zápis textový.

2.2.1 Textový zápis

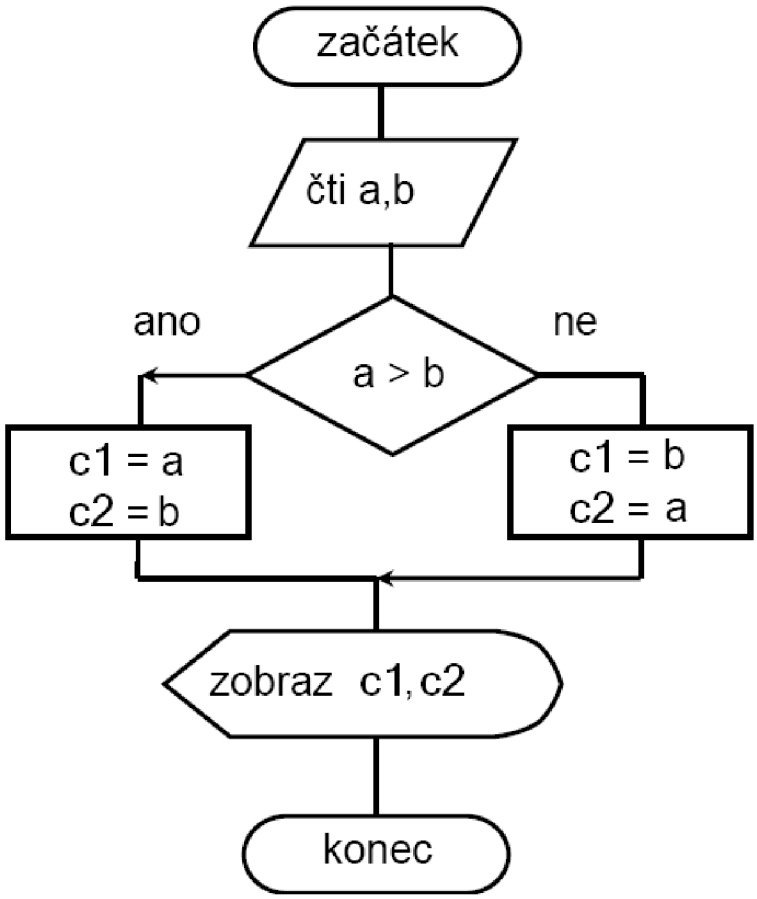
Velmi častá reprezentace algoritmu je jeho textový zápis přirozeným jazykem. Používá často při definici nějakého problému, např. ve slovních úlohách, nebo také pro jednoduché nastínění řešení problému, jak jej známe z manuálů, nebo kuchařek. Dalším vyjádřením algoritmů je zápis pomocí pseudokódu. *Pseudokód* je neformální způsob zápisu algoritmu, který je podobný zápisu v programovacím jazyce s tím rozdílem, že je zde snaha vyhnout se detailní syntaxi jazyka, jako jsou deklarace proměnných, klíčová slova, přístupy ke zdrojům atp. Často je pseudokód zapisován stručným přirozeným jazykem, ale více se používá zápis se syntaxí podobnou jazykům C a PASCAL. Asi nejčastěji se však setkáme se zápisem algoritmů přímo v některém z programovacích jazyků. Takto zapsaný algoritmus může být velmi praktický pro ty, kteří jazyku rozumí a mohou algoritmus v daném jazyce ihned použít a otestovat, ale na druhou stranu takto zadaný algoritmus může být pro ostatní neznalé jazyka nevypovídající o funkci algoritmu. Při zápisu algoritmů nejen textově je vždy dobré zvážit, za jakým účelem je algoritmus zaznamenám a podle toho zvolit vhodný způsob zápisu.

2.2.2 Vývojové diagramy

Mnohem více obecněji a formálně lze algoritmus zapsat, nebo spíše zobrazit graficky. Takto vyjádřit algoritmus je možno několika prostředky, mezi které patří také vývojové diagramy. Vývojové diagramy zobrazují posloupnost operací, které je třeba provést při transformaci vstupních dat na výstupní. Tento zápis algoritmů by se v Česku měl řídit českou normou ČSN ISO 5807 z roku 1996.

Diagramy se skládají ze symbolů pro vlastní operace, včetně symbolů pro definici stanoveného toku, sem patří symboly např. pro cyklus, podmínku a vstup od uživatele. Další skupinou jsou symboly spojení, které indikují tok řízení. Chceme-li zpřehlednit diagram lze použít zvláštních symbolů, a ty zde mohou mít funkci komentáře. Na obrázku 2.1 je zobrazen vývojový diagram pro algoritmus, který na vstupu přečte 2 čísla a vypíše je seřazené od největšího po nejmenší.

Dříve byly diagramy používány velmi hojně, dnes se od nich postupně upouští a nahrazují je metody jiné. Jednou z těchto metod jsou strukturogramy

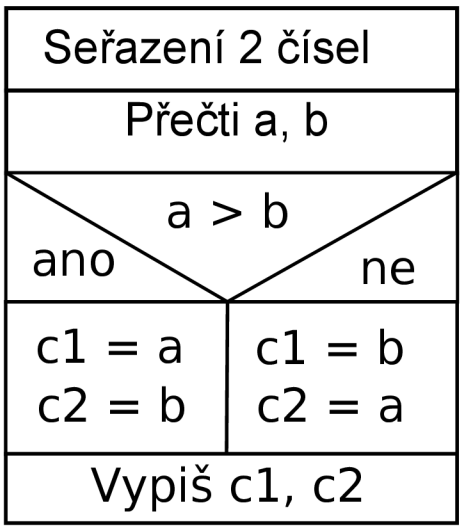


Obrázek 2.1: Vývojový diagram pro algoritmus seřazení 2 čísel [2]

2.2.3 Strukturogramy

Nassi-Shneiderman diagramy, nebo také strukturogramy, graficky znázorňují strukturu algoritmu. Pro základní algoritmické struktury, jako posloupnost, cyklus a podmínka, používají různé barevné, nebo tvarové rozlišení.

Základem strukturogramu je vždy obdélník, který má v záhlaví název algoritmu, nebo operace. Uvnitř obdélníku jsou popsány jednotlivé kroky algoritmu, lze do něj vnořovat další strukturogramy.



Obrázek 2.2: Strukturogram algoritmu seřazení 2 čísel podle velikosti [2]

Podmínka	**1**	**2**	**3**
A = B	A	N	N
A < B	N	A	N
A > B	N	N	A
Vypiš A, B	X	X	
Vypiš B, A			X

Tabulka 2.1: Seřazení 2 čísel zapsané v rozhodovací tabulce. [2]

Na obrázku 2.2 je ukázán strukturogram pro algoritmus seřazení 2 čísel z předchozího příkladu.

2.2.4 Rozhodovací tabulky

Způsob zápisu algoritmů pomocí rozhodovacích tabulek je využíván zejména pro algoritmizaci úloh se složitějším rozhodováním při zpracování hromadných dat. Jsou prostředkem pro vytváření komplexní logiky rozhodování relativně jednoduchým způsobem. V tabulkách se postupně zobrazuje variace činností, které se mají provést při různých kombinacích vstupních podmínek.

Algoritmus seřazení dvou čísel je pomocí rozhodovacích tabulek znázorněn v tabulce 2.1, splnění podmínky je zapsáno písmenem A, nesplnění písmenem N a provedení činnosti je označeno písmenem X.

Kapitola 3

Výuka algoritmizace

Výsledkem praktické části této práce je grafický výukový systém, který lze použít k výuce algoritmizace a programování. Pří návrhu výukových programů obecně je potřeba myslet především na to, aby program splnil svůj účel, kterým je umožnit uživateli učit se prostřednictvím programu, v případě této práce poskytnout uživateli nástroj pro učení se algoritmizace.

Problém výuky nemůžeme dostatečně zvládnout, pokud si nepřiblížíme samotný proces učení, faktory, které učení ovlivňují a nepřizpůsobíme program těmto faktorům.

Tato kapitola spolu s obrázky je převzata a rozšířena ze semestrálního projektu [2], z kterého práce vychází.

3.1 Učení

Učení je proces, který člověka provází celý život, ať už je to učení vědomé nebo nevědomé. Existuje několik definic učení, budeme vycházet z definice ze zdroje [11] a po definici si představíme některé základní druhy učení.

Učení je proces nabývání nových znalostí a nových způsobů reagování . . . Učení zahrnuje mnohem víc než úmyslné učení nazpaměť a nácvik – učení není jen jedním specifickým druhem aktivity. Je to změna, která se vyskytuje v organismu během mnoha druhů aktivity. Projevuje se později jako její následný účinek. Pozdější aktivita je odlišná v důsledku aktivity předcházející – učení vytváří relativně trvalé následné účinky.

3.1.1 Druhy učení

Učení podmiňováním Učení podmiňováním je nejjednodušší druh učení, jde v podstatě o vytváření podmíněných reflexů. Toto učení lze dále rozdělit na *klasické*, kdy se jedinec učí novému chování na základě opakujících se podmínek a naučené chování se potom projeví vždy za daných podmínek (pokusy I.P. Pavlova s krmením psů za stejných podmínek).

Dalším typem podmiňovacího učení je *instrumentální*, které spočívá v aplikaci principu pokus-omyl, při zjišťování toho co vede k úspěchu.

Posledním typem podmiňovacího učení je *operační*, ve kterém je jedinec za správné řešení odměněn a za špatné trestán.

Senzomotorické učení Dalším typem učení je učení senzomotorické, kterým se rozvíjí koordinace vjemu a pohybu. Příkladem takového učení je hra na hudební nástroj nebo

práce s nářadím. Učením se jedinec snaží provádět co nejúspornější a nejefektivnější pohyby. Při tomto druhu učení je velmi důležitá sebekontrola.

Učení verbální Toto učení je nejrozšířenějším druhem učení u člověka, jde o učení se slov, jejich významu, funkci a porozumění projevu druhých lidí. Také se toto učení vztahuje na osvojení slov a dalších symbolů např. matematických.

Základem verbálního učení jsou asociace mezi slovy a jejich reálnou prezentací. Výsledkem tohoto učení jsou vědomosti, které tvoří základ k dovednostem.

Učení řešením problému Při učení řešením problému se jedinec učí správně definovat problém, případně jej zařadit mezi typy problémů, zorientovat se a připravit řešení problému. Součástí je také ověření správnosti řešení, popřípadě zdokonalování řešení problému a samotná aplikace řešení na problém.

Způsob řešení problémů může být analytický s využitím myšlenkových postupů nebo intuitivní metodou pokus-omyl.

Sociální učení Při tomto učení si jedinec osvojuje sociální zkušenosti získané komunikací s ostatními lidmi, které se pro něj stávají tzv. modelem, a tento model potom napodobuje. Sociálním učením se formují důležité znaky osobnosti, jako jsou umění jednat s lidmi, společenský takt, ohleduplnost a morální charakter člověka.

3.1.2 Faktory ovlivňující učení

Existuje obecně mnoho faktorů ovlivňujících učení [10] [11], ale podíl jednotlivých faktorů na úspěchu učení je u každého jiný.

Asi nejdůležitějším faktorem ovlivňujícím učení obecně je motivace. Platí, že nejvhodnější je středně silná motivace, při silné nebo slabé motivaci se organizmus učí pomalu. Silná motivace vede spíše k dezorientaci chování.

Faktor emoce působí podobně jako motivace, silné emoce ztěžují proces učení, protože snižují schopnost koncentrace.

Mezi další důležité faktory patří inteligence, která přímoúměrně ovlivňuje rychlost učení. Únava, ať fyzická nebo psychická, naopak stěžuje učení. Faktor úspěšnosti předchozího učení a dosažených mezivýsledků pozitivně motivuje k dalšímu učení, naopak neúspěchy učení znesnadňují.

3.1.3 Didaktické zásady

Didaktické zásady [7] [12] slouží pro učitele nebo žáka při učení sebe sama, k dosáhnutí maximální efektivity a účinnosti učení. Na tyto zásady by měl být brán ohled i při tvorbě takové pomůcky, jakou jsou výukové programy pro algoritmizaci. Zde jsou jednotlivé zásady.

Uvědomělost a aktivita Je potřeba v žákovi vzbudit kladný vztah k učení. Tato zásada je úzce spojena s motivací, kterou je dobré žákovi předat. Nutností také je, aby se žáci učili s porozuměním a uvědoměle.

Vědeckost Učená látka musí odpovídat nejnovějším poznatkům vědy a techniky a musí být učena v souladu se současnými poznatky pedagogiky.

Cílevědomost Učitel musí jasně stanovit, zdůvodnit a objasnit specifické cíle každého celku.

Postupnost Dodržením této zásady musí být látka vyučována od lehčího k těžšímu, jednoduchého k složitějšímu, od blízkého ke vzdálenému, obecného k abstraktnímu atp.

Trvalost Osvojené znalosti a dovednosti musí být trvale uloženy v paměti. K dosažení trvalosti je potřeba používat opakování již naučeného učiva a expozici. Expozice je předkládání nových poznatků a měla by zajistit dokonalé pochopení učiva.

Názornost Zásada vychází z toho, že žáci mají o vyučované problematice určité představy a ty je potřeba uchopit tak, abychom žákům problém vysvětlili. Názornosti dosahujeme využitím zrakových, sluchových, čichových, hmatových a pohybových vjemů.

Přiměřenost Tato zásada stanovuje, že je potřeba přizpůsobit obsah, rozsah, obtížnost a způsob vyučování učiva, duševní a tělesné vyspělosti a znalostí žáků. Neúměrná obtížnost látky vede k mechanickému učení učiva bez osvojení jeho hlubšího pochopení.

3.2 Vlastnosti výukových programů

Jelikož se práce zaměřuje na nástroje pro výuku algoritmizace určené začátečníkům, a to převážně žákům vyššího stupně základních škol a studentům středních škol, musí být vybírány nástroje přiměřeně jednoduché a názorné. Takové programy, aby jejich ovládání a tvorbu byli schopni pochopit všichni zájemci o programování, kteří nemají žádné zkušenosti s programováním a aby byl uživatel schopen po krátkém seznámení se s programem začít vytvářet jednoduché algoritmy.

Cílovou skupinou těchto systémů jsou dospívající v rozmezí od šesti do patnácti let, tato kategorie uživatelů velmi ráda používá počítač více k zábavě než k práci a potrpí si na přívětivé uživatelské rozhraní s pokud možno nejnovějším vzhledem.

Pokud chceme tuto skupinu oslovit a zaujmout měl by mít program blízko ke hře, nebo umožňovat uživateli hru vytvořit, aby udržel pozornost uživatelů, také by samotné vytváření algoritmů nemělo být pouhým psaním textu, nebo zadáváním hodnot.

Pokusím se v dalších odstavcích popsat jaké vlastnosti by podle mě měl ideální program pro výuku algoritmizace mít a proč si myslím, že právě tyto vlastnosti jsou tak důležité.

3.2.1 Přehlednost

Tato vlastnost lze zobecnit snad pro všechny programy nabízející uživatelské rozhraní, dobrý program by měl mít přehledné a snadno pochopitelné uživatelské rozhraní, speciálně programy pro výuku žáků základních škol. Při návrhu uživatelského rozhraní by měl být kladen důraz na jednoduchost a intuitivnost.

Jak bylo zmíněno výše, programy pro výuku dospívajících by měly používat co nejvíce multimediálních prvků, které by měly být v ideálním případě použity k interakci s uživatelem, platí tu ovšem stejně jako jinde pravidlo o střídmosti, proto je důležité předem zvážit, které prvky do programu začlenit, a kterým se vyhnout.

Přehlednost uživatelského rozhraní je svázána s didaktickou názorností, program, který má uživatelské rozhraní nepřehledné, určitě nemůže být použit jako názorná didaktická pomůcka.

Je asi zřejmé, že pokud se budeme snažit navrhnout rozhraní co nejjednodušší, budeme nuceni skrýt velké množství funkcí, aby zůstaly opravdu jen ty nejzákladnější, nejpoužívanější. Je potřeba navrhnout jednoduché ovládání, a přitom úplně neskrýt pokročilejší funkce. Řešením může být definování předem několika úrovní uživatelského rozhraní a

nechat uživatele, ať si zvolí jaká úroveň mu vyhovuje, tímto lze splnit další didaktickou zásadu, kterou je přiměřenost. Sám uživatel si zvolí na jakou úroveň se cítí, ve které bude dosahovat lepších výsledků a postupně se zkušenostmi může úroveň zvyšovat.

3.2.2 Jednoduchost

V návaznosti na předchozí požadavek přehlednosti uživatelského rozhraní, bych rád uvedl další vlastnost, kterou je jednoduchost. Jednoduchost v tom smyslu, že jakýkoliv uživatel, ať už programováním nepoznamenaný žák základní školy, nebo pokročilý programátor, by měl být schopen po spuštění programu vytvořit jednoduchý program, aniž by musel zdlouhavě experimentovat, nebo pročítat nápovědu, ta by ovšem v systému neměla chybět, aby v případě potřeby uživatele přivedla na správnou cestu.

Chceme-li podmínku jednoduchosti splnit, měli bychom vytvořit takový systém, který by nabízel zápis algoritmu pro člověka co nejpřirozenější formou. U textových zápisů algoritmů, by tedy například kód algoritmu po přečtení od začátku do konce měl dávat smysl a jednotlivé příkazy zapsané po sobě by měli vytvářet prosté věty. U grafických reprezentací algoritmů by použité symboly měly co nejvíce znázorňovat funkci příkazů, které reprezentují. Pokud bychom chtěli tuto vlastnost reflektovat v nějaké ze zmíněných didaktických zásad, tak je to nejspíše zásada názornosti.

3.2.3 Obecnost a znuvupoužitelnost

Obecnost je poměrně klíčovou vlastností popisovaných programů, ty by neměly uživatele nutit používat návyky jednoho, či druhého jazyka, ve kterých se programy inspirují. Konkrétně například pro definici nové proměnné by se měly vyhnout takovým názvům jako jsou *mallock*, nebo *new* známé z jazyků C a C++ a použít radši nestranný název *nova_promenna*. Ovšem zrovna tento příkaz, alokace paměti pro novou proměnnou, by měl být prováděn automaticky v pozadí, aby zbytečně nezatěžoval uživatele a ten mohl svůj čas věnovat algoritmu, nikoli tomu jestli přidělenou paměť správně uvolňuje, nebo jestli nealokuje příliš velké množství.

Určitě však není třeba puntičkářsky vyžadovat pro všechny operace odlišný zápis, nebo dokonce vymýšlet nějaké extra postupy, které by uživatele naučily nevhodným návykům, stačí se vyhnout převzetí příliš konkrétní konstrukce z uvažovaného programovacího jazyka. Je vhodné použít obecně platné způsoby tedy ty, které jsou používány v nejrozšířenějších jazycích.

Obecnost se netýká pouze pojmenování příkazů, ale také způsobu zápisu algoritmu, ten by mělo být možné zapsat jak graficky, tak textově, aby si uživatel mohl sám zvolit způsob podle toho na co je zvyklý a co mu více vyhovuje.

Znuvupoužitelnost u těchto programů zaručuje, že algoritmus, který si uživatel vytvořil a uložil může po nějaké době otevřít a znovu použít starý program pro návrh nějakého lepšího, nebo složitějšího programu. Je také dobré, když existuje možnost vytvořený algoritmus pojmenovat a použít v programu vícekrát odkazem přes jeho jméno aniž by musel být algoritmus několikrát kopírován. Tento postup by do určité míry mohl nahradit funkce, jak je známe z běžných programovacích jazyků.

3.2.4 Jazyková nezávislost

Aby výukový program byl opravdu použitelný i pro nejmladší generaci, musí nabízet možnost návrhu algoritmu bez nutnosti jakéhokoli psaní zdrojového kódu uživatelem. Tvrzení vy-

chází z toho předpokladu, že zápis algoritmu musí být co nejvíce jednoduchý a pro zápis algoritmu v nějakém jazyce bychom museli naučit uživatele nejdříve používat danou syntax jazyka, a až potom by byl schopný řešit zadaný problém.

Nezávislost na konkrétním jazyku lze také pochopit tak, jak je popsána v části o obecnosti, neměli bychom tedy uživatele učit jak se algoritmus zapisuje v daném jazyce, ale jak algoritmus vymyslet a složit z dostupných bloků reprezentujících příkazy.

Pokud uživatel v programování objeví zálibu a bude chtít používat běžné programovací nástroje pro vývoj reálných aplikací bude mít dobré základy pro to, aby použil jakýkoli jazyk a nebude se mu plést syntaxe naučená z výukového nástroje se syntaxí jiného složitějšího jazyka.

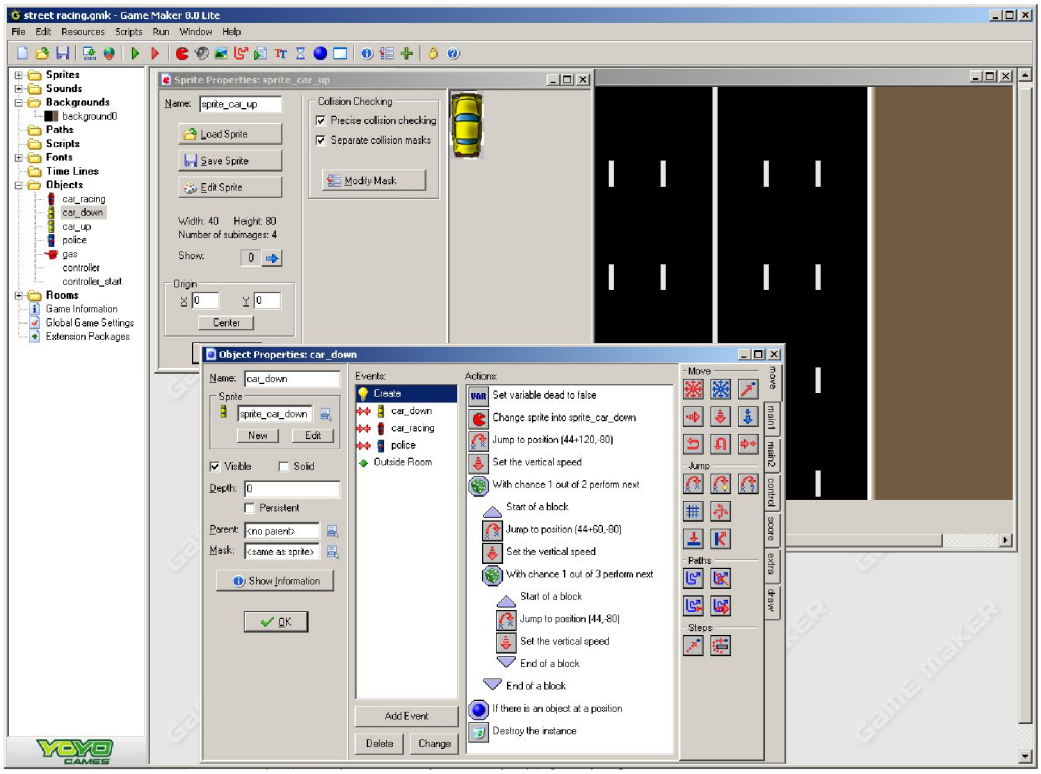
3.3 Současné programy pro výuku algoritmizace

Protože tento typ programů je úzce specializován a není předurčen k masovému používání, neexistuje těchto programů mnoho, zvláště na Českém trhu je nabídka poměrně slabá a výhradní postavení má Baltík, kterého popíši dále. Snažil jsem se podobné programy vyhledat, bral jsem v potaz pouze takové programy, která mají české uživatelské prostředí a jsou zaměřeny na výuku algoritmizace u dětí školního věku. Našel jsem několik zajímavých programů, ovšem některé jako Oživlé algoritmy ze Střední průmyslové školy elektrotechnické v Úžlabině v Praze nejsou dostupné pro veřejnost, nebo o programech nejsou dostupné žádné podrobnější informace, tak zde uvedu pouze ty, o kterých lze získat více informací a můžeme si je vyzkoušet. V zadání práce je jako příklad programu pro výuku algoritmizace uvedený Game Maker, a tak výčet začnu u něho.

3.3.1 Game Maker

Popis: Game Maker [13] je komerčním nástrojem, ale lze jej vyzkoušet v omezené verzi (Lite Edition). Program vyvíjí firma YoYo Games, konkrétně profesor Mark Overmars. Historie programu sahá až do roku 1999, kdy začal vývoj tohoto nástroje. Nástroj je celý napsán v Delphi, a protože používá knihovny WinAPI a DirectX 8, je předurčen k běhu v prostředí MS Windows. Tento nástroj není primárně vytvořen pro výuku algoritmizace, ale můžeme jej k ní poměrně dobře použít. Jak název napovídá, jedná se o program pro tvorbu her, ve světě je velmi oblíbeným prostředkem a ne zřídka se využívá ve školách, zájmových kroužcích a různých letních kempech. Na internetu lze nalézt několik stránek s návody, češtinou a zajímavými hrami vytvořenými právě v Game Makeru. Při návrhu hry si uživatel postupně definuje mapu hry, sprity, objekty, zvuky, obrázky a další zdroje. Pomocí formulářů nastavuje spritům vlastnosti a reakce na události, které chce uživatel ošetřit a v tzv. objektech se definuje logika hry. Objekty si lze představit jako třídy, které mají různé metody včetně konstruktoru a mohou reagovat na různé události uživatele jako stisk klávesy, nebo tlačítek myši. Vytvářet hru lze také přímo v speciálním jazyce GML (Game Maker Language). Na obrázku 3.1 je ukázka prostředí tohoto nástroje.

Hodnocení: Tento nástroj pro tvorbu her vede uživatele k rozložení komplexního problému (hry) na podproblémy (jednotlivé objekty). Nástroj také učí uživatele vytvářet logické vazby mezi jednotlivými objekty a velkou výhodou je přítomnost ladícího módu, kdy uživatel vidí za běhu jeho algoritmus a může si uvědomit chyby nebo lepší postup.



Obrázek 3.1: Ukázka prostředí programu Game Maker [2]

Díky těmto rysům může být program Game Maker použit pro lidi, kteří se chtějí naučit programovat. Dle mého názoru výborně řeší pomocí objektů práci s událostmi a použitý přístup je podobný objektově orientovanému návrhu.

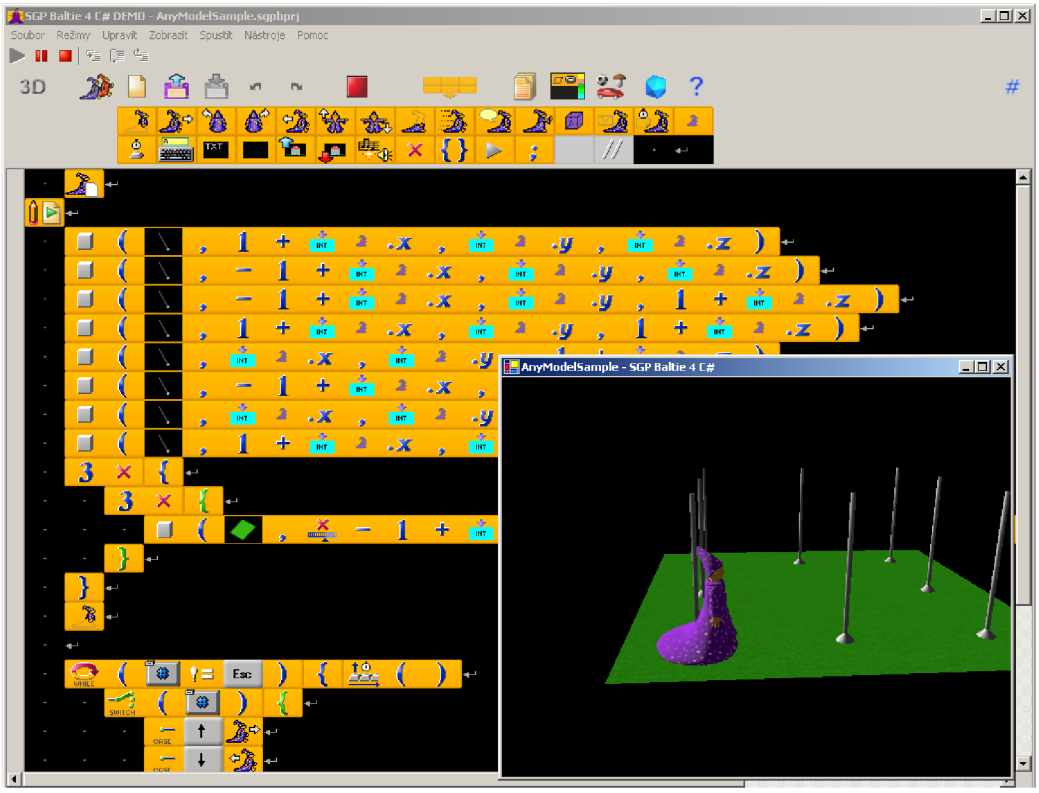
Jako zápory programu bych uvedl orientaci pouze na platformu MS Windows, nutnost platit za plně funkční verzi a hlavně složitost ovládání, se kterým se lze sžít až po prozkoumání několika přiložených příkladů, nebo pročtení tutoriálu k programu.

Pokud bych měl program hodnotit z pohledu didaktických zásad, tak motivace uživatele je poměrně jasná, vytvořit si vlastní hru. Uživatel tedy cílevědomě tvoří názorné příklady. Co tomuto programu chybí je postupnost.

Díky složitosti bych Game Maker zařadil spíše do kategorie pro pokročilejší uživatele, kteří nemusí mít velké zkušenosti s programováním, ale měli by mít dobré základy práce s počítačem a dostatek trpělivosti k pochopení toho, jak jsou hry vytvářeny. Odměnou za jejich trpělivost ovšem získají dobrý způsob jak rychle a graficky vytvářet zajímavé hry, které nemusí být vůbec jednoduché.

3.3.2 Baltík

Popis: Baltík [18], nebo také Baltazar, je systémem pro výuku algoritmizace, jak jsem jej popisoval výše. Tento placený software vyvíjí česká firma SGP Systems a sklízí s ním velké úspěchy, jak v Česku, tak na Slovensku. Baltík je s oblibou využíván na školách a pro nadšence jsou pořádány zajímavé soutěže.



Obrázek 3.2: Ukázka okna programu Baltie 4 C# [2]

První grafická verze programu vznikla roku 1996 a byla založena na jazyce Baltazar, který byl napsán v jazyce C. Od verze Baltie 4 C# je Baltík vyvíjen v jazyku C# na platformě .NET.

Princip tvorby aplikací v Baltíkovi je poměrně jednoduchý, je založený na ovládání čaroděje Baltíka, který se může pohybovat po scéně a kouzlit různé obrázky. Podle toho v jakém režimu jsme, můžeme Baltíka různě ovládat.

Skládací režim umožňuje ruční skládání obrázků z banky obrázků do scény, tu je potom možné uložit a použít v programovacím režimu. V čarovacím režimu již můžeme ke skládání scény použít Baltíka a pomocí jednoduchých příkazů jako krok vpřed, otočení a vykouzlení obrázku sestavíme celou scénu.

A konečně v režimu programování můžeme pospojováním jednoduchých příkazů z režimu čarování zapsat posloupnost příkazů (algoritmus), podle které se bude Baltík řídit. Je zde možno použít základní programové konstrukce jako cykly a podmínky, základní matematické funkce, animace, práce se soubory, používání vlastních proměnných, obsluha klávesnice, podpora základních zvuků a videa a další potřebné funkce pro tvorbu programů.

Uživatel si může definovat vlastní procedury, ale ne funkce. Výsledný program je možné spustit pouze v programu Baltík a není jej možno uložit jako exe soubor.

Od verze Baltie 4 C# přibylo několik novinek oproti verzi 3. Jak jsem se zmínil již dříve, je tato verze napsaná v jazyce C# a obsahuje i kompilátor tohoto jazyka, takže

pokročilejší programátoři mohou zapisovat program přímo v C#. Díky kompilátoru mohou uživatelé svůj program uložit jako spustitelný exe soubor. Novinkou je zde také rozšíření o 3D grafiku.

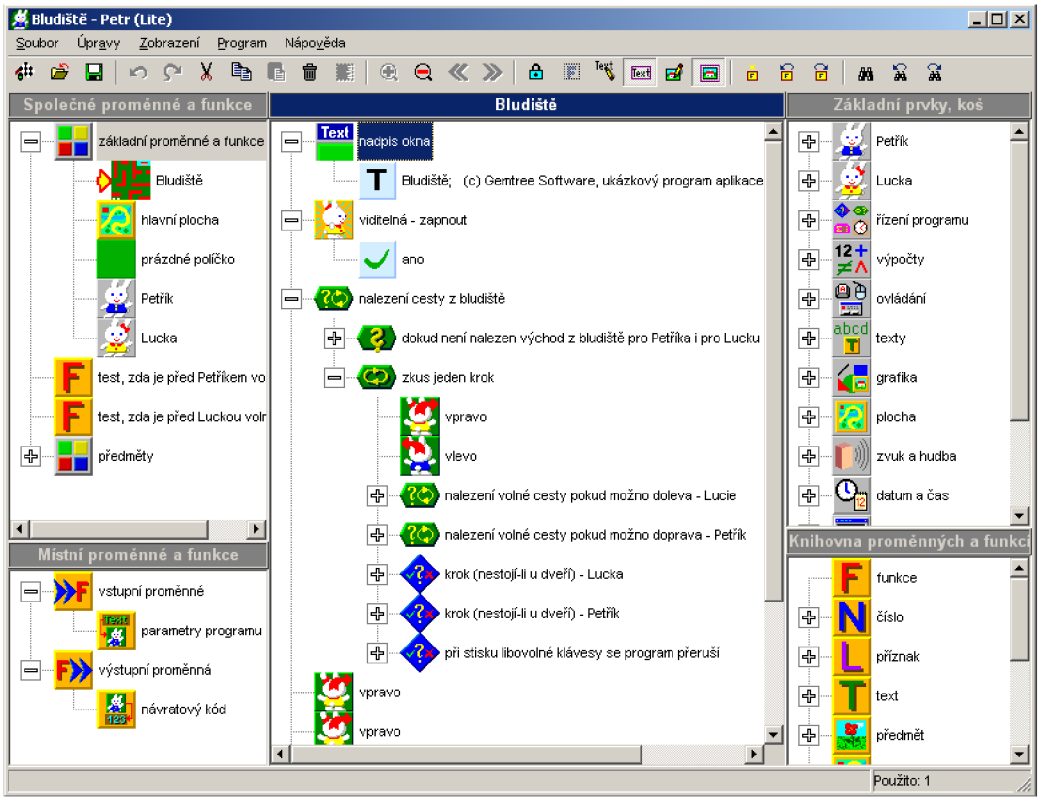
Obě verze si lze stáhnout na stránkách SGP Systems jako demoverze, ve kterých není možné uložit výsledky práce. Na obrázku 3.2 je zachyceno okno programu Baltie 4 C#, ve kterém je jedna z ukázkových aplikací, dodávaná k programu. Je zde vidět poměrně přívětivé, intuitivní a pěkné uživatelské rozhraní.

Hodnocení: Baltíka hodnotím jako velmi povedený nástroj pro výuku programování na základních školách, ostatně fakt, že se program používá na některých školách a pořádají se v něm celorepublikové soutěže, svědčí o jeho kvalitě. Systém si vysloužil také řadu ocenění od Softwarových novin a veletrhu Invex. Škoda jen, že se jedná o komerční systém, i když existuje časově a funkčně omezená demoverze.

Aplikace nepostrádá ani motivaci, ani názornost a díky třem režimům práce tvorby scény ani postupnost, pro tyto vlastnosti je program vhodný i pro výuku mladších žáků.

3.3.3 Petr

Popis:



Obrázek 3.3: Okno aplikace Petr s ukázkovým příkladem [2]

Petr [14] je jednoduchý programovací nástroj, který vyvíjí firma Gemtree Software, s.r.o. Program je placený a lze jej získat ve verzích Standard a Professional, zdarma lze získat jako freeware omezenou verzi Lite pro domácí použití.

Petr umožňuje tvořit programy pomocí poskládání grafických příkazů. Hlavními postavami tohoto programu jsou králíci Petr a Lucka, se kterými uživatel programu pohybuje po mapě podobně jako v Baltíkovi s čarodějem. Pracuje se zde s předměty, které králíci přemísťují po mapě.

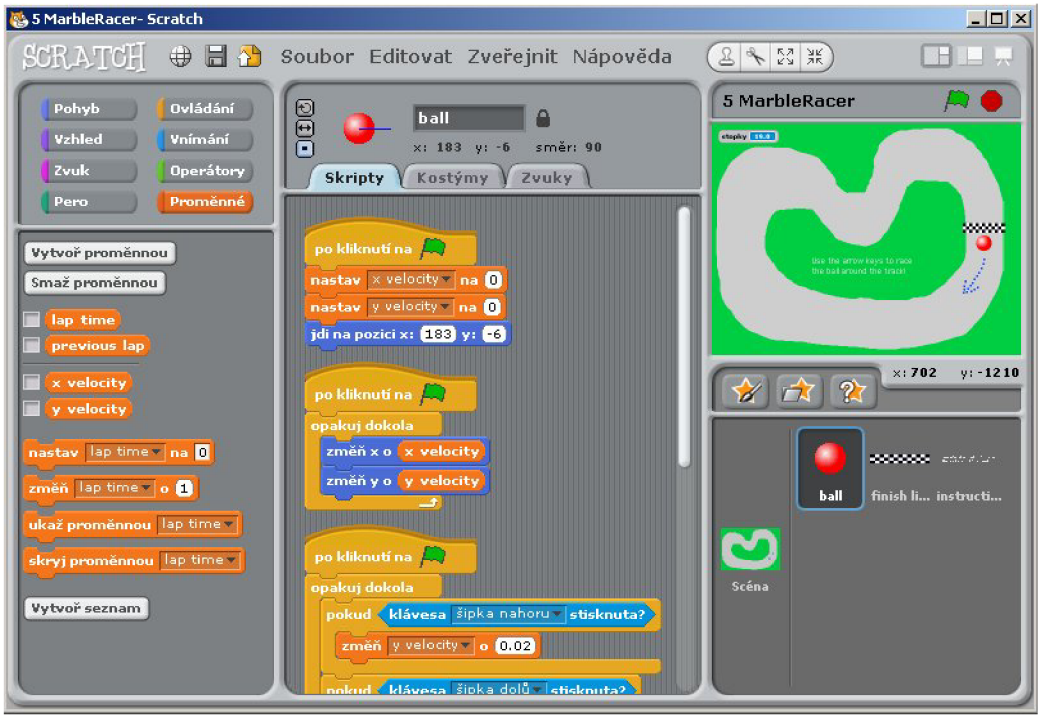
Stejně jako předchozí programy podporuje načítání multimediálních formátů, 3D grafiku s použitím OpenGL u starších verzí, od verze 2. je používána knihovna DirectX, najdeme zde také podporu síťové komunikace s použitím DirectPlay a překlad programu do spustitelného formátu.

Na obrázku 3.3 je zobrazeno okno hlavní aplikace Petr, v které je načten ukázkový příklad.

Hodnocení: Program se mi jeví na stejné úrovni jako Baltík, jak co do množství funkcí, tak do přehlednosti a intuitivnosti ovládání, zaostává snad jen v tom, že nenabízí více režimů tvorby scény jako Baltík a jeho rozhraní je sice intuitivní a přehledné, ale podobá se spíše klasické aplikaci, než výukovému programu.

Při svém průzkumu jsem bohužel nenarazil na soutěže, nebo reference na školy, které program k výuce používají, to může být signálem, že programu něco chybí. Opět je škoda, že se jedná o placený program a zdarma lze získat jen funkčně omezenou verzi.

3.3.4 Scratch



Obrázek 3.4: Okno aplikace Scratch

Popis: Tento program [17] je vyvíjen skupinou z MIT Media Lab a jeho tvůrci Scratch popisují jako programovací jazyk, ve kterém si snadno vytvoříte vlastní interaktivní

příběh, animaci, hru, muziku nebo umění a můžete svůj výtvor jednoduše sdílet na internetu.

Jde o poměrně mladý program, který je povedeným grafickým rozšířením jazyka Logo. Scratch se zaměřuje na děti od osmi let, uživatelé v něm mohou vytvářet jednoduché animace se zvuky a hry. Velkou výhodou je sdílení vytvořených programů na internetu, kde si aplikaci návštěvník stránek může vyzkoušet v podobě java appletu, bude-li se mu aplikace líbit, může si po bezplatné registraci stáhnout zdrojový soubor spolu s vývojovým studiem Scratch a začít vytvářet vlastní programy.

Program se zde skládá z příkazů, které mají podobu kousků stavebnice puzzle, tyto kousky do sebe zapadají a vytváří jednotlivé bloky kódu, ty pak mohou být volány podobně jako procedury.

Ovládání programu je přeloženo do češtiny, neměl by tedy být problém s jeho pochopením u uživatelů, kteří angličtinu nezvládají tak dobře. Problém je ovšem s manuály a dalšími návody jak začít, které jsem našel pouze v angličtině, ale to je pochopitelné vzhledem k tomu, že program byl uveden před třemi roky a zatím nemá v Česku tolik uživatelů.

Na obrázku 3.4 můžete vidět ukázku rozhraní programu.

Hodnocení: Program sice nenabízí tolik funkcí jak Baltík, nebo Petr, ale poskytuje důležité základní příkazy, které by mohl uživatel potřebovat, nechybí zde ani možnost program krokovat.

Aplikace je velmi zajímavá zvláště pro mladší uživatele, díky propracovanému uživatelskému rozhraní a snadnosti pochopení principu tvorby programu.

Na stránkách aplikace je přes 3000 vytvořených projektů, ze kterých lze čerpat inspiraci, velké množství uživateli vytvořených projektů jasně vypovídá o oblíbenosti aplikace.

Z pohledu didaktických pravidel na programu nevidím žádné nedostatky.

3.3.5 Ostatní programy

Zde uvedu zbylé programy, na které jsem narazil při hledání nástrojů na výuku algoritmizace, a které stojí dle mého názoru za zmínku.

Alice [19] je open source projekt vyvíjený na Carnegie Mellon University, program je implementovaný v programovacím jazyce Java. Systém se používá pro tvorbu 3D animací. Hlavním důvodem vývoje takovéto aplikace byla snaha vytvořit program, který by sloužil k výuce programování bez nutnosti učení se syntaxe programovacích jazyků. Alice se zdá jako poměrně povedený nástroj, škoda jen, že neexistuje počeštěná verze a je určen pouze k tvorbě animací.

V případě nástroje Karel [8] se nejedná o grafické programování, ale o ovládání robota Karla na mapě zápisem textových příkazů. V programech se pak řeší situace jako kolize se zdí a podobně.

Jazyků pro ovládání robota jako je Karel, existuje více, poměrně povedenou aplikaci vytvořila společnost Lego. Jde o software dodávaný k stavebnici s názvem Mindstorm [16], kde skládáním příkazů za sebe, které ovládají robota, můžete naprogramovat zakoupeného robota od firmy Lego. Aplikace je dodávána pouze jako součást ne příliš levné stavebnice spolu s robotem.

Dalším způsobem jak učit algoritmizaci, který zde zmíním jako poslední, je použití programovacího jazyka Logo [9]. Logo je dostupné v poměrně velkém počtu verzí, které jsou vyvíjeny jako open source, nebo komerčně.

Hlavní myšlenkou tohoto jazyka je ovládání želvy chodící v písku, která za sebou zanechává v písku ocasem stopu, pro tento druh programování byl zaveden nový pojem želví grafika. Jazyk se využívá, kromě kreslení 2D a 3D grafiky, také k výuce umělé inteligence a matematické vizualizaci.

3.4 Výuka algoritmizace na školách

Jak jsem zjistil, otázka výuky algoritmizace na základních školách a netechnicky zaměřených středních školách zůstává stále otevřená. Pokud jde o základní školy, tak podle Rámcového vzdělávacího programu pro základní vzdělání [5] z roku 2007 by se měla výuka informačních a komunikačních technologií pro 1. stupeň skládat ze základů práce s počítačem, vyhledávání informací a komunikace a zpracování informací pro 2. stupeň se fáze až na základy práce s počítačem opakují ovšem s většími požadavky na žáka.

Zavedení základů algoritmizace a programování je tedy pouze na aktivitě učitelů a vzdělávacím programu základní školy. V praxi se s povinným, častěji povinně volitelným předmětem, výuky základů algoritmizace a programování můžeme setkat nejspíše pouze na základních školách se zaměřením na informatiku, nebo gymnáziích. Pro ostatní žáky, kteří takovéto školy nenavštěvují zbývá vyhledat některý ze zájmových kroužků na jiných školách, nebo organizacích, typu domova dětí a mládeže.

Žákům základních škol musí být prezentována algoritmizace zábavnou formou, využívají se proto často některé z výše popsaných grafických programovacích jazyků, jsou však i výjimky, kdy žáci začínají programovat jazycích jako Pascal, nebo Visual Basic.

Pokud jde o školy ukončující vzdělání výučním listem, je v jejich rámcovém vzdělávacím programu výpočetní technika, ale jde o pokračování té jak je praktikována na základních školách, tedy se snahou naučit žáky využívat informační technologie ve svůj prospěch jako obyčejní uživatelé nikoli programátoři.

Jinak je tomu u středních škol zakončených maturitní zkouškou. Ty se dají rozdělit podle výuky algoritmizace na dvě skupiny. V první jsou školy netechnického zaměření, u nich je v rámcovém programu většinou zmínka o ovládnutí principů algoritmizace úloh. Rozsah výuky algoritmizace opět závisí na zvážení vedení školy, můžeme se setkat se školami, kde se žáci v rámci obecného předmětu informační technologie seznámí s definicí algoritmizace a vyzkouší si nějaké praktické ukázky, ale jsou i takové školy, které tomuto problému věnují někdy i celý předmět.

Druhou skupinou jsou školy technického zaměření a gymnázia. U technických středních škol, jsou programování a algoritmizace vyučovány většinou v samostatném předmětu, nebo i několika na sebe navazujících předmětech. Studenti v rámci těchto předmětů získávají základy algoritmizace a učí se programovat v některém z “dospělých” programovacích jazyků. Stejně jsou na tom gymnázia, kde jsou kladeny obecně větší nároky na studenty.

Kapitola 4

Použité knihovny

V zadání práce není stanovený jazyk a knihovny, které lze použít, jedinou uvedenou podmínkou je, že program musí být multiplatformní. Jelikož nejvíce zkušeností v programování mám s jazyky C a C++, rozhodl jsem se pro použití C++ spolu s jednou z multiplatformních knihoven pro tvorbu grafického uživatelského rozhraní (dále jen GUI).

Po kratším hledání jsem se rozhodoval mezi těmito toolkity: Qt, wxWidgets. S ohledem na možnost použití propracovaného designeru GUI, kvalitní dokumentaci a podpory ze strany vývojářů jsem zvolil knihovnu QT, která je ve verzi Qt Open Source Edition dostupná pod licencí GPL, tato volba mi velmi ulehčila práci a knihovnu můžu každému jenom doporučit.

Další důležitou volbou před samotným návrhem bylo rozhodnutí, jestli pro interpretaci vytvořeného programu použít nějaký již vytvořený dostupný jazyk, nebo vytvořit vlastní.

Pokusil jsem se o vytvoření interpretu jednoduchého jazyka za použití generátorů FLEX a BISON, díky čemuž, jak se později ukázalo, jsem strávil poměrně dlouhou dobu ve slepé uličce. Tyto nástroje sice umožňují výborně a efektivně vytvořit interpret, ale jejich propojení s Qt programem by bylo poměrně složité, tak jsem raději od tohoto řešení upustil. Stejně tak by potom bylo neefektivní další přidávání nových funkcí do programu.

Po domluvě s vedoucím práce jsem byl odkázán na skriptovací jazyk LUA, který výborně posloužil svému účelu. Oba použité nástroje budou podrobněji popsány v dalších kapitolách.

4.1 Qt

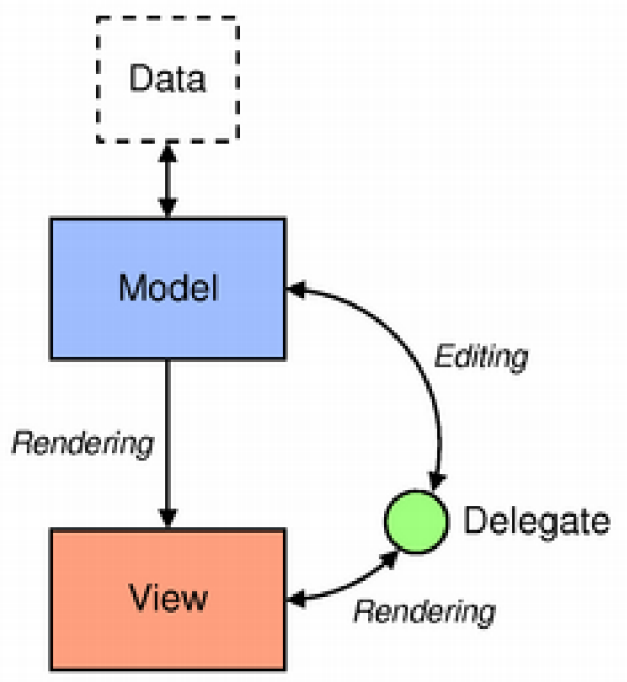
Jedná se o jednu z nejpoužívanějších multiplatformních knihoven [6] pro tvorbu GUI. Knihovnu vyvíjela společnost Trolltech, nyní se o ní stará společnost Nokia. Programy napsané s použitím Qt je možné přeložit a používat na platformách MS Windows, Mac OS X, GNU/Linux a také na Symbianu S60 třetí edice, nebo v některých embedded systémech.

Qt je napsáno v C++ a používá vlastní renderovací jádro, díky němu je vzhled aplikací závislý na systému, ve kterém jsou používány. Kromě vytváření GUI umožňuje tato knihovna i využití dalších nástrojů nezbytných pro moderní programy jako vlákna, parsování XML, síťovou komunikaci, propojení s databázemi a OpenGL, práci s multimédii a také nabízí podporu pro práci s vektorovou grafikou, podporu překladu rozhraní aplikací do jiných jazyků a další užitečné vylepšení .

U knihovny jsem především ocenil vyšší úroveň abstrakce obsluhy událostí pomocí signálů a slotů, oddělení návrhu grafického rozhraní od funkčního kódu a také mě překvapily výhody architektury model-view, kterou Qt v mnoha případech používá. Tato ar-

chitektura je inspirována Model-View-Controler modelem použitým v jazyce Smalltalk, výhodou zmíněného přístupu je úplné oddělení dat a jejich reprezentace od jejich zobrazení a od rozhraní pro jejich editování.

Na obrázku 4.1 je zobrazená architektura použitá v Qt, kde je zjednodušena část controler, pojmenována jako delegát.



Obrázek 4.1: Znázornění model-view v Qt [6]

Pro snadný vývoj Qt aplikací můžeme použít vývojové prostředí Qt Creator 4.2, které umožňuje kromě psaní kódu také překlad a ladění programu. Pro návrh GUI je v balíčku pro vývoj Qt aplikací nástroj Qt Designer, kde je možné za použití layoutů snadno vytvořit vzhled aplikace. Díky nástroji Qt Assistant je uživateli dostupná kompletní nápověda, která je opravdu propracovaná a nabízí mnoho zajímavých příkladů.

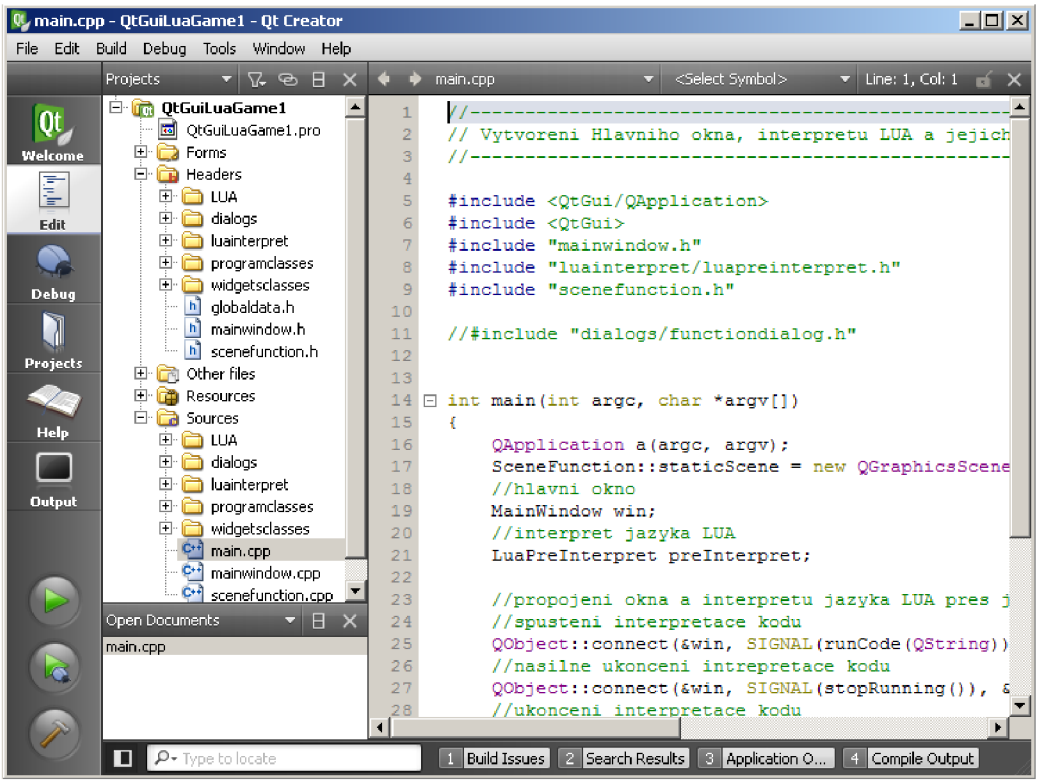
Asi nejznámější projekty vytvořené v Qt jsou grafické prostředí KDE, webový prohlížeč Opera, aplikace Google Earth, komunikátor Skype a virtualizační nástroj VirtualBox.

4.2 LUA

Lua [4] je slovy autorů účinný, rychlý, odlehčený a vestavitelný skriptovací jazyk. Jde o dynamicky typovaný jazyk s vlastní správou paměti garbage collectorem. Nabízí jednoduchou procedurální syntaxi, asociativní pole a rozšiřitelnou sémantiku.

Zápis programů uživatelům usnadňuje některými ze syntaktických vylepšení jako například vícenásobné přiřazení a přístup k prvkům asociativního pole přes tečkovou notaci, zajímavou vlastností jazyka také je, že přistupuje k funkcím stejně jako k ostatním datovým typům, tělo funkce lze tedy dynamicky měnit během vykonávání skriptu.

Velkou výhodou tohoto jazyka je snadné vestavění do aplikace, a to hlavně díky tomu, že jeho celá implementace včetně interpretu a překladače do bytekódu je naprogramována



Obrázek 4.2: Prostředí Qt Creator [6]

pouze v jazyce C, takže stačí přidat zdrojové soubory do projektu a spustit skript přes rozhraní interpretu. Všechny zdrojové kódy i přeložené binární soubory jsou dostupné pod liberální licencí MIT.

Jazyk se stal poměrně oblíbeným a je používán, jak v profesionálních aplikacích, tak v amatérských. Jako asi nejznámější využití mohu uvést podporu jazyka v programu Adobe Photoshop Lightroom a ve hře World of Warcraft, sami autoři dokonce tvrdí, že LUA je v současnosti lídrem skriptovacích jazyků pro hry.

Obrázek 4.3: Logo jazyka LUA [4]

Kapitola 5

Programová implementace

Aby se tato diplomová práce co nejvíce přiblížila zkoumanému tématu a abych mohl objektivněji posoudit složitost a úskalí návrhu aplikací pro výuku algoritmizace na základních a středních školách, implementoval jsem v rámci praktické části program, který lze použít pro výuku algoritmizace.

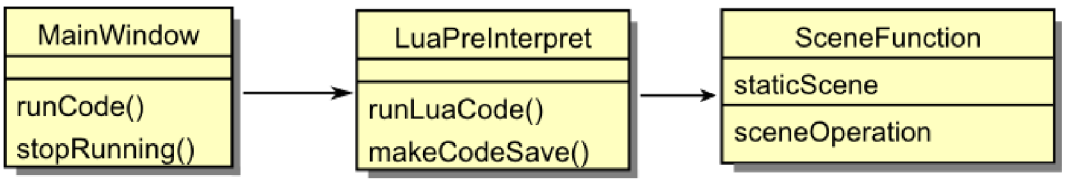
V programu jsem se snažil dodržet co nejvíce pravidel definovaných v předchozích kapitolách. Do jaké míry se mi podařilo dodržet tyto pravidla a jaké jsou reakce z praxe se zmíním v následující kapitole, zde rozeberu, jak je aplikace navrhnuta a jaké problémy jsem musel řešit.

5.1 Návrh a rozdělení funkčnosti

Návrh je důležitou součástí vývoje aplikací a pokud se podcení, je velmi pravděpodobné, že se čas vývoje softwaru prodlouží buď jednoduše proto, že byl návrh špatný a zjistilo se to při programování, v takovém případě se obě fáze návrhu a kódování musí opakovat nebo proto, že návrh nepočítal se všemi stavy programu, což může být zjištěno až ve fázi testování, takže proces nápravy bude časově mnohem náročnější.

Proto jsem věnoval návrhu dostatek času, v mém případě bylo třeba navrhnout aplikaci tak, abych oddělil hlavní program, interpretaci LUA kódu a operace se scénou. Třídy použité v aplikaci lze tedy logicky oddělit do třech skupin tříd 5.1, které jsou na sobě nezávislé. Jde o pomyslné části programu, které dále rozepíši.

V této kapitole jsou zobrazeny také UML diagramy tříd použitých v programu, jen pro upřesnění uvádím, že všechny uvedené relace mezi třídami nakreslené jednoduchou čárou zakončené plnou šipkou v tomto případě reprezentují, že třída z které je šipka vedena používá třídu na kterou šipka ukazuje.



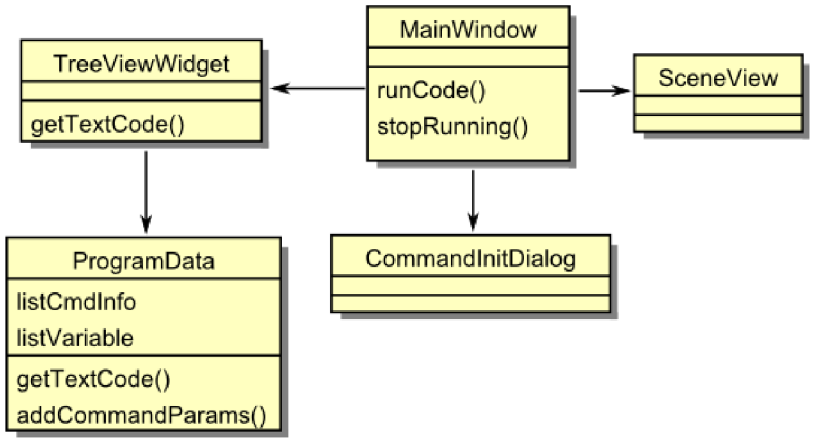
Obrázek 5.1: Hlavní třídy programu

Hlavní okno je nejrozsáhlejší část programu a obsahuje třídu hlavního okna, které se zobrazí po spuštění aplikace, třída při svém vytvoření inicializuje seznam příkazů zobrazených v levém panelu aplikace a vytvoří novou instanci třídy *TreeViewWidget*, na které se přetahováním příkazů bude skládat program.

Třída *TreeViewWidget* kromě grafické reprezentace a editace příkazů poskytuje také rozhraní k ukládání parametrů příkazů a evidenci deklarovaných proměnných. Poslední dvě zmiňované služby třídě *TreeViewWidget* poskytuje její privátní seznam instancí třídy *ProgramData*.

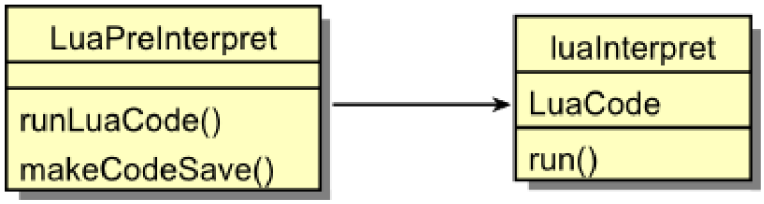
Třída *ProgramData* implementuje metody pro evidenci parametrů u jednotlivých příkazů, evidenci deklarovaných proměnných a převod programu do zápisu v jazyce LUA.

Do skupiny hlavního okna patří také dialogy pro zadání parametrů příkazů, třída pro zobrazení náhledu na scénu a třída s informacemi o příkazech.



Obrázek 5.2: Třídy části hlavní okno

Interpretace LUA kódu je v pořadí druhá část a stará se o propojení s jazykem LUA. První třídou, do které je rozhraním předán kód pro spuštění, je třída *LuaPreInterpret*. Ta kód upraví, aby byl více odolný proti zacyklení a spustí vykonávání interpretace kódu třídou *LuaInterpret*. Třída *LuaInterpret* pak již používá zdrojový kód jazyka LUA k interpretaci kódu, implementuje a registruje také pomocné metody, které volají funkce pracující se scénou. Tyto funkce jsou volány ze skriptu Lua, proto musí být statické, protože se v metodách volají metody pracující se scénou, musí být i ty statické.



Obrázek 5.3: Třídy části interpretu LUA

Operace se scénou jsou definovány pouze ve třídě *SceneFunction*, které se po inicializaci předá scéna na níž budou operace prováděny a pak již stačí volat metody pro práci se scénou. Mezi tyto metody patří operace pro přidávání, mazání, pohyb, detekci kolizí a nastavení vlastností předmětů ve scéně a doplňkové metody jako změna pozadí scény, zastavení provádění skriptu na nějakou dobu nebo zjištění rozměrů scény. Podrobnější seznam operací bude uveden v kapitole 5.4 popisující výslednou aplikaci.

5.2 Návrh uživatelského rozhraní

Při návrhu rozhraní a způsobu ovládání programu jsem se snažil použít co nejvíce prvků, které by uživateli práci s programem usnadnily. V programu je proto možné vkládat příkazy metodou *drag and drop* (táhni a pusť). Tyto příkazy jsou reprezentovány jak graficky, tak textově a stejně tak jednotlivé skupiny příkazů jsou rozlišeny textovým zápisem a ikonou, která symbolizuje k čemu jsou příkazy v dané skupině určeny.

Obrázková reprezentace neslouží v tomto případě pouze pro zkrášlení uživatelského rozhraní, ale má praktický význam pro rychlost vybírání příkazů. Jak jsem zjistil z krátkého pozorování jednotlivých uživatelů mého programu, uživatelé se po seznámení s programem řídili více podle obrázků, nikoli podle textu popisku příkazů. Proto jsem skupiny příkazů rozlišil také obrázky, aby uživatel mohl rychleji najít hledaný příkaz.

Další požadavek, který jsem si stanovil, byla minimalizace možnosti udělat syntaktickou chybu při skládání programu. Bylo potřeba proto vytvořil dialogy, pro zadání podmínek, přiřazení, rozsahu cyklu for a parametrů funkcí. Tyto dialogy by měly uživateli nabídnout plnohodnotný zápis, ale neměly by dovolit zadat chybnou hodnotu.

5.3 Problémy při implementaci

I když jsem návrhu věnoval dostatek času, přece jen jsem narazil na některé problémy, jedním z nich byl problém jak vykreslit scénu, se kterou bude uživatel pracovat, naštěstí je v Qt knihovně implementován užitečný model složený ze scény (*QGraphicsScene*), náhledu (*QGraphicsView*) a položek (*QGraphicsItem*), jednotlivé položky ve scéně mohou dokonce přijímat události. Zmíněný model programátorovi velmi usnadňuje práci s prvky scény a animacemi. Pro odstranění nechtěných efektů při vykreslování dynamické scény jsem v instanci náhledu na scénu použil akceleraci pomocí knihovny OpenGL.

Při návrhu jsem řešil také problém zajištění správné funkce aplikace i při interpretaci skriptu, který má nekonečnou smyčku s vloženými příkazy, mezi kterými není čekání.

Definoval jsem si proto vlastní pomocnou funkci, nazvěme ji kotvící, která je v předzpracování kódu vložena na začátek těla cyklu a uživatelem definované funkce. Kotvící metoda pak pouze zajistí zpracování událostí hlavního okna z fronty událostí po stanovenou dobu. Tím aplikace reaguje i při spuštění cyklícího programu a to i na méně výkonných počítačích.

Původně jsem kvůli ošetření "zamrznutí" uživatelského rozhraní interpretoval skript v samostatném vlákně, pak se ale ukázalo, že pro synchronizaci dvou vláken, kde jedno používá operace se scénou *QGraphicsScene* a druhé scénu vykresluje do komponenty *QGraphicsView*, nestačí použít synchronizaci zamykáním přístupu ke scéně, ale je potřeba synchronizovat přístup na úrovni každé položky. Kvůli této komplikaci jsem zvolil jednodušší variantu, kde vše běží ve vlákně hlavní aplikace a je použita metoda přerušení skriptu pro zajištění obsluhy událostí formuláře. Tento přístup přináší stejné výsledky jako varianta s vlastním vláknem.

Dalším věcí, na kterou jsme se více zaměřil, je problém přidávání nových příkazů. Snažil jsem se použít takový koncept, aby pro vytvoření nového příkazu bylo potřeba editovat co nejméně zdrojových souborů. Všechny příkazy proto identifikuji přes jejich jedinečné ID a veškeré potřebné informace o příkazu lze zjistit vytvořením instance třídy *ParentCmd* s parametrem identifikátoru příkazu.

Pro příkazy požadující parametry jsem vytvořil dynamický dialog, který se vytvoří opět na míru danému příkazu podle jeho počtu a typu parametrů.

5.4 Výsledná aplikace

5.4.1 Princip tvorby algoritmů

Aplikace poskytuje základní řídící příkazy, jejichž skládáním je možné vytvářet různě náročné algoritmy. Vytvořené algoritmy mohou být jakkoli složité a nápadité, co jim však bude chybět je nějaké znázornění toho jak fungují, pro tento případ jsou v aplikaci poskytnuty příkazy, které pracují se scénou, na této scéně může uživatel dle svého uvážení zobrazovat libovolné obrázky a text. Operace pro práci s těmito obrázky ovšem nejsou pouhým vykreslením obrázku na scénu, to by asi nebylo použitelné pro znázornění algoritmů, ale obrázky jsou přidávány do scény buď jako objekt, nebo jako robot.

Rozdíl mezi těmito dvěma způsoby přidání spočívá v přístupu k položce při operacích. Pokud bude přidán obrázek jako objekt, bude s ním zacházeno jako se statickým předmětem, ty může uživatel pouze přidávat, nastavit jim pozici a natočit na určitý úhel, objekt nemůže kreslit čáru, nebo se pohybovat jinými metodami než nastavením pozice. Může ovšem být více objektů se stejnými názvy a na všechny budou aplikovány stejné operace.

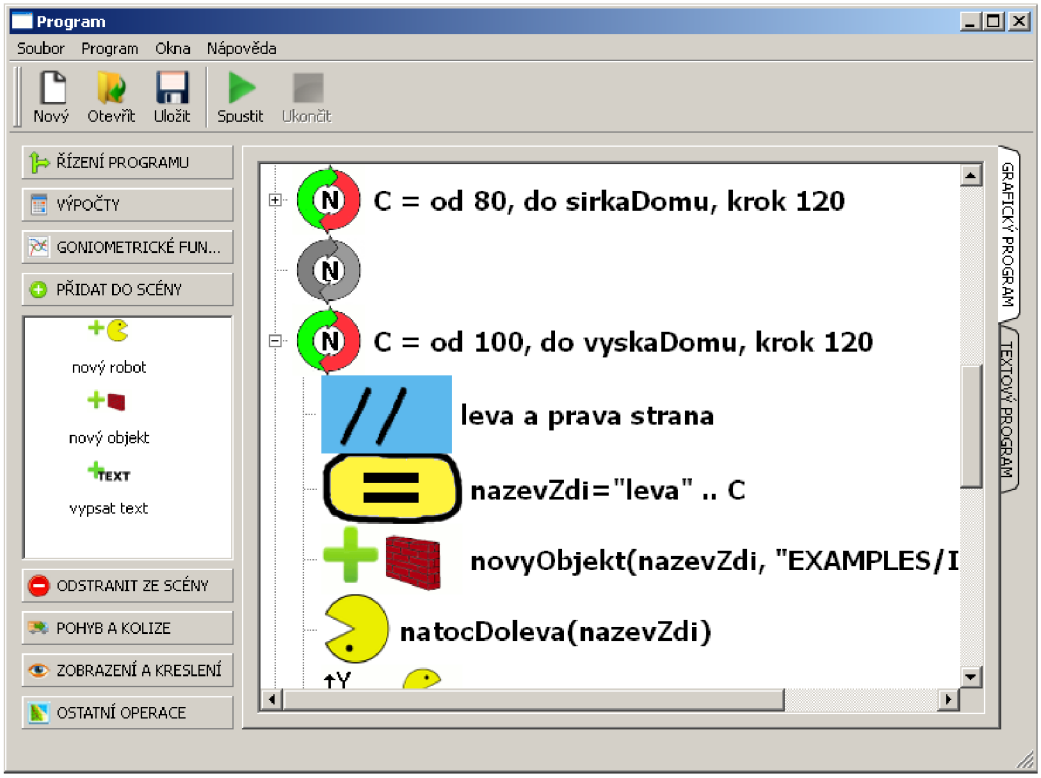
Pokud přidá uživatel obrázek do scény jako robot, může s robotem provádět všechny dostupné operace, ale pokud se ve scéně objeví dva roboti se stejným jménem, tak bude při volání operací s tímto jménem reagovat pouze první přidaný robot.

Reprezentace obrázků těmito dvěma způsoby lze využít také pro rozlišení role obrázků ve scéně, toho se využívá v operacích mazání a detekci kolizí, které mohou pracovat se všemi roboty, nebo předměty. Například pokud chceme ve scéně vytvořit zeď, která po nárazu robota zmizí, stačí zeď sestavit z objektů libovolných názvů a pak po každé změně polohy robota detekovat kolizi s objektem, pokud dojde k nějaké kolizi, smažeme všechny objekty ve scéně a celá zeď zmizí.

Robotům lze kromě jiného nastavit také jejich vlastnost kreslení, po přidání robota do scény robot nemá tuto vlastnost nastavenou. Ta lze změnit příkazy *kresli čáru* a *kresli body*, kreslení lze potom zrušit příkazem *nekresli*. Jakmile je robotovi nastaveno kreslení, bude všude kde se pohybuje kreslit čáru, nebo body, v případě čar se jedná o propojení předchozí a současné pozice robota ve scéně, v případě bodů na těchto pozicích pouze zanechává značky. S využitím této vlastnosti může uživatel vytvářet podobnou grafiku jako v programovacím jazyce Logo.

5.4.2 Popis funkčnosti

Jak už to u většiny grafických aplikací bývá, je i zde hlavní okno programu rozhraním přes které komunikuje uživatel s programem. Ukázka rozhraní programu je zobrazena na obrázku 5.4, okno je rozděleno na 2 části, levý panel s příkazy a pravý blok, kde uživatel vytváří program.



Obrázek 5.4: Okno vytvořené aplikace

V levé části jsou všechny příkazy, které může uživatel k programování použít, přehledně seskupeny dle svého účelu, jejich podrobný popis bude uveden později. Příkazy lze z tohoto panelu jednoduše přetáhnout do pravé části programu, kde se příkaz vloží na místo, kam jej uživatel přetáhnul. K prezentaci programu pomocí grafických prvků, jsem použil komponentu, která umožňuje jednotlivé položky seskupovat do stromové struktury, proto je výsledný program přehledně odsazen podle úrovně zanoření a lze jednotlivé bloky programu v náhledu zjednodušit na počáteční a koncový příkaz tak, jak je tomu ve většině vývojových prostředích.

Po vložení příkazu, který vyžaduje nastavení hodnot, se zobrazí dialog pro jejich zadání, uživateli je v tomto dialogu dovoleno zapisovat hodnoty pouze tam, kde by se mohla vyskytovat konstanta v ostatních případech je uživateli nabídnut výběr z operátorů, nebo proměnných, při takovém použití se uživatel prakticky nemůže dopustit syntaktických chyb.

Jednotlivé dialogy byly navrhnuty tak, aby uživateli neumožňovaly vytvořit syntaktickou chybu, ale také tak, že uživatele neomezují v zadání hodnot příkazů. Abych uvedl příklad, tak pro zadání hodnot podmínky platí, že proměnnou lze porovnat s další proměnnou, nebo konstantami, jak číselnými, tak řetězcovými. Tato možnost výběru dosazení konstanty, nebo proměnné je implementována ve všech dialozích.

Pokud si je uživatel jist svým algoritmem a chce jej vyzkoušet, stačí použít tlačítko spustit a v okně s náhledem na scénu se promítnou všechny změny, které uživatel v algoritmu se scénou provedl. Nezadal-li uživatel do některého z použitých příkazů potřebné parametry, bude na tento fakt upozorněn chybovým hlášením a zvýrazněním neúplného příkazu.

Aplikace nabízí pro pokročilejší uživatele export programu do textové podoby, v tex-

tovém zápisu mohou uživatelé využít bez zábran všech vlastností jazyka LUA, tato vlastnost je pro zkušenější a je v aplikaci pouze jako doplňková funkce, uživatel se v textovém editoru nesetká s žádnými jinými vylepšeními, které by mu usnadňovali psaní kódu, tak jako tomu je v grafické programové části.

Pokud tedy uživatel úspěšně vytvoří vlastní algoritmus, může jej spustit přes hlavní menu program-spustit, nebo ikonou zelené šipky. Po stisku se zobrazí okno se scénou a začne se provádět algoritmus zapsaný uživatelem. V okně scény uživatel za běhu vidí jak jeho algoritmus funguje, pokud chce program zpomalit, za účelem větší názornosti, může si do programu přidat příkaz čekej, který provádění algoritmu pozastaví na zadanou dobu. Je-li v programu nějaká chyba, je na ní uživatel patřičně upozorněn chybovou hláškou. Při spuštění algoritmu s nekonečnou smyčkou, který nemusí být nutně chybný (i když nesplňuje základní vlastnost algoritmů konečnost), se program nezastaví a je na uživateli, kdy program ukončí použitím akce ukončit v menu programu, nebo ikonou červeného čtverce.

5.4.3 Implementované příkazy

Aplikace nabízí 8 skupin příkazů, nejdůležitější skupinou jsou řídící příkazy, které jsou alfou a omegou sestavování algoritmů, tyto příkazy jsou umístěny nejvýše v seznamu skupin příkazů, protože očekávám nejčastější využívání v sestavovaných algoritmech.

Dalšími dvěmi skupinami jsou příkazy pro jednodušší matematické výpočty a pokročilejší matematiku v podobě goniometrických funkcí. Pod těmito matematickými příkazy jsou příkazy, které do scény umožňují přidávat objekty, text a roboty. Následují příkazy pro odstranění vložených položek ze scény.

Třetí a druhá skupina od konce poskytují operace s roboty a objekty ve scéně, jde o skupinu s operacemi pro pohyb a kolize, druhou skupinu tvoří příkazy pro zobrazení a skrytí položek scény a kreslení pomocí robotů. V poslední a nejníže umístěné skupině příkazů nalezne uživatel obecné operace, jako nastavení pozadí scény, získání rozměrů scény a pozastavení scény. Dále zde podrobně vypíši seznam všech operací seřazený podle toho jak jsou operace uskupeny v programu. Pro zjednodušení zápisu ovšem ještě definuji některé symboly, které budu při zápisu používat.

<proměnná> jde o proměnnou, předem deklarovanou v programu

<porovnávací operátor> reprezentuje tyto operátory ==, `<,>`, =, >=, <= v tom smyslu, že levá strana operátoru je rovna, je menší, je větší, není rovna, je větší, nebo rovna, je menší, nebo rovna v relaci k pravé straně operátoru.

<operátor> reprezentuje tyto operátory +, –, *, /, ∧, N ve smyslu sčítání, odčítání, násobení, dělení, umocňování a přiřazení.

<hodnota> za tento symbol lze dosadit jak proměnnou, tak celočíselnou, desetinnou, nebo řetězcovou konstantu.

<int> za tento symbol lze dosadit proměnnou a celočíselnou konstantu.

<float> za tento symbol lze dosadit proměnnou a desetinnou konstantu.

<string> za tento symbol lze dosadit proměnnou a řetězcovou konstantu.

Řízení programu: v této skupině jsou příkazy, pojmenovány záměrně stejně tak, jak jsou používány v negrafických programovacích jazycích, proto budu při jejich popisu stručnější.

- *if* – jednoduchá podmínka, tak jak je známa z programovacích jazyků. Příkaz požaduje následující parametry <proměnná><porovnávací operátor><hodnota>.

- *if else* – jednoduchá podmínka spolu s větví else, které se provede při nesplnění podmínky. Příkaz požaduje stejné parametry jako příkaz *if*.
- *while* – smyčka jejíž počet provádění záleží na podmínce ukončení smyčky, jsou požadovány stejné parametry jako u příkazu *if*.
- *for* – smyčka, u které je počet opakování předem znám a zadává se ve tvaru <proměnná> = <int><int><int>, celočíselné hodnoty popořadě určují počáteční hodnotu, která se přiřadí do proměnné, koncovou hodnotu, kterou když překročí hodnota proměnné, tak se cyklus ukončí a poslední číslo udává, kolik se má přičíst po každém průchodu smyčkou k proměnné.
- *break* – po vykonání tohoto příkazu algoritmus okamžitě přeruší vykonávání smyčky, v které je příkaz vložený a pokračuje za tělem smyčky, příkaz nemá parametry.
- *proměnná* – tento příkaz slouží k deklaraci proměnných, příkaz má jeden textový parametr, do kterého může uživatel zadat název proměnné, ten by měl splňovat následující regulární výraz $[a-zA-Z][a-zA-Z0-9]*$ a neměl by se shodovat z žádným jiným názvem proměnné nebo funkce, na tuto skutečnost, bude uživatel vhodně upozorněn.
- *komentář* – jde o obdobu komentáře, jak jej známe z jiných jazyků, jeho parametrem je libovolný jednořádkový textový řetězec, který nemá vliv na chování algoritmu.

Výpočty: tato skupina reprezentuje jednoduché matematické operace.

- *přiřazení* – tento příkaz poskytuje uživateli možnost do proměnně přiřadit nějakou hodnotu, nebo výsledek matematické operace definované symbolem <operátor>. Příkaz požaduje tyto parametry <proměnná> = <hodnota> <operátor> <hodnota>. Při výběru operátoru N, je provedeno pouze přiřazení první z hodnot do proměnné.
- *minimální číslo* – tento příkaz vrací menší ze dvou zadaných hodnot. Parametry příkazu jsou <proměnná><float><float>.
- *maximální číslo* – příkaz vrací maximální číslo ze dvou zadaných čísel, parametry jsou stejné jako u předchozího příkazu.
- *mocnina* – příkaz vrací do proměnné hodnotu prvního čísla umocněného druhým číslem. Parametry příkazu jsou stejné jako u maximálního čísla.
- *odmocnina* – příkaz vrací druhou odmocninu ze zadaného čísla. Parametry příkazu jsou <proměnná><float>.
- *absolutní hodnota* – příkaz vrací absolutní hodnotu zadaného čísla. Parametry příkazu jsou <proměnná><int>.
- *náhodné číslo* – příkaz vrací vygenerované náhodné číslo v intervalu dvou celých čísel. Parametry jsou <proměnná><int><int>.
- *exp* – vrací hodnotu čísla *e umocněného na zadanou hodnotu. Parametry jsou <proměnná><float>.*
- přirozený logaritmus – *vrací přirozený logaritmus zadaného čísla. Parametry jsou <proměnná><float>.*
- desítkový logartimus – *vrací desítkový logaritmus čísla. Parametry jsou <proměnná><float>.*

Goniometrické funkce: veškeré úhly pro goniometrické funkce musí být zadány v radiánech.

- *PI* – vrací hodnotu čísla π. Parametry příkazu jsou <proměnná>.
- *sin* – vrací *sinus* zadaného úhlu. Parametry příkazu jsou <proměnná><float>.
- *cos* – vrací *cosinus* zadaného úhlu. Parametry příkazu jsou <proměnná><float>.
- *tan* – vrací *tangens* zadaného úhlu. Parametry příkazu jsou <proměnná><float>.
- *arc sin* – vrací *arcus sinus* ze zadaného čísla. Parametry jsou <proměnná><float>.
- *arc cos* – vrací *arcus cosinus* ze zadaného čísla. Parametry jsou <proměnná><float>.
- *arc tan* – vrací *arcus tangens* ze zadaného čísla. Parametry jsou <proměnná><float>.
- *sinh* – vrací *hyperbolický sinus* ze zadaného čísla. Parametry jsou <proměnná><float>.
- *cosh* – vrací *hyperbolický cosinus* ze zadaného čísla. Parametry jsou <proměnná><float>.
- *tanh* – vrací *hyperbolický tangens* ze zadaného čísla. Parametry jsou <proměnná><float>.
- *stupně na radiány* – převede úhel zadaný ve stupních na radiány. Parametry jsou <proměnná><float>.
- *radiány na stupně* – převede úhel zadaný v radiánech na stupně. Parametry jsou <proměnná><float>.

Přidat do scény: zde jsou příkazy, umožňující přídání položky do scény.

- *nový robot* – přidá do scény novou položku reprezentující robota. Parametry jsou <string><string>, první řetězec je název robota, druhým řetězcem je cesta k souboru s obrázkem.
- *nový objekt* – přidá do scény novou položku reprezentující objekt. Parametry jsou <string><string>, první řetězec je název objektu, druhým řetězcem je cesta k souboru s obrázkem objektu.
- *nový text* – vypíše do scény text. Parametry jsou <string><int><int><int><string>, první řetězec je text, který se má vypsat, další parametry určují souřadnici x a y pozice kam se má text vypsat, čtvrtý parametr udává velikost textu v bodech a poslední parametr určuje barvu vypsaného textu.

Odstranit ze scény: zde jsou příkazy umožňující odebrání ploužky ze scény.

- *smaž* – smaže položku zadaného názvu ze scény. Parametrem je <string>.
- *smaž roboty* – smaže všechny roboty ve scéně. Příkaz nemá parametry.
- *smaž objekty* – smaže všechny objekty ve scéně. Příkaz nemá parametry.
- *smaž texty* – smaže všechny texty ve scéně. Příkaz nemá parametry.
- *smaž vše* – smaže všechny položky ve scéně. Příkaz nemá parametry.

Pohyb a kolize: zde jsou příkazy pracující s položkami ve scéně.

- *nastav pozici* – nastaví dané položce novou pozici ve scéně. Parametry jsou <string><int><int>, první parametr určuje název položky, druhý a třetí novou pozici x a y položky ve scéně.
- *pohni robotem* – posune robota o vzdálenost zadanou v souřadnicích scény. Parametry jsou <string><int><int>, první řetězec určuje název robota, druhý a třetí hodnotu x a y o kterou se má robot posunout.

- *pohni se ve směru* – posune robota ve směru natočení o zadanou vzdálenost. Parametry jsou <string><int><int>, první řetězec určuje název robota, druhý vzdálenost v pixelech o kterou se má robot posunout.
- *otoč* – otočí robota o zadaný úhel po směru hodinových ručiček. Parametry jsou <string><int>, prvním je název robota a druhým úhel otočení.
- *natoč doleva, natoč doprava, natoč nahoru, natoč dolů* – otočí robota o takový úhel, aby jeho směr odpovídal názvu příkazu. Kde základní pohled po přidání robota je nahoru. Parametrem je <string>, což je název robota.
- *natoč* – nastaví směr robota podle zadaného úhlu. Parametry jsou <string><int>, prvním je název robota a druhým úhel, na který se má robot nastavit.
- *s kým koliduje robot?* – vrací název robota s kterým koliduje robot. Parametry jsou <proměnná><string>.
- *s čím koliduje robot?* – vrací název objektu s kterým koliduje robot. Parametry jsou <proměnná><string>.
- *kolidují?* – vrací číselnou hodnotu 0, nebo 1 podle toho, jestli spolu dva roboti kolidují, nebo ne. Parametry jsou <proměnná><string><string>.

Zobrazení a kreslení: zde jsou příkazy, které umožňují robotům nastavit danou vlastnost.

- *zobraz* – nastaví položku s daným názvem ve scéně viditelnou. Parametrem je <string>.
- *schovej* – nastaví položku ve scéně na neviditelnou. Parametrem je <string>.
- *kresli čáru* – nastaví robotovi s daným názvem schopnost kreslit čáru. Parametrem je <string>.
- *kresli body* – nastaví robotovi s daným názvem schopnost kreslit body. Parametrem je <string>.
- *nekresli* – zruší robotovi s daným názvem schopnost kreslit. Parametrem je <string>.

Ostatní operace: zde jsou pomocné příkazy, které nespadají ani do jedné z předchozích tříd.

- *nastav pozadí* – nastaví pozadí scény na opakující se obrázek. Parametrem je <string>, který určuje cestu k souboru s obrázkem.
- *počkej* – pozastaví provádění operací scény na zadaný čas v milisekundách. Parametrem je <int>, kterým je čas, na který se má provádění pozastavit.
- *výška scény* – vrátí aktuální výšku scény (ta se může během provádění programu měnit). Parametrem je <proměnná>.
- *šířka scény* – vrátí aktuální šířku scény (ta se může během provádění programu měnit). Parametrem je <proměnná>.

Kapitola 6

Testování a vyhodnocení výsledků

Abych dokázal, že mnou vytvořený program pro výuku algoritmizace je využitelný v praxi, pokusil jsem se zjistit názory pedagogů, kteří se specializují na výpočetní techniku. Vytvořil jsem pro tento účel webový dotazník [3], kde jsem kladl respondentům otázky týkající se výuky algoritmizace a programování na jejich škole a dále jsem se snažil zjistit jejich názor na vytvořený výukový program.

Oslovil jsem proto emailem přes 50 náhodně vybraných kantorů výpočetní techniky jak základních, tak středních škol a gymnázií. Z této skupiny dotazník vyplnilo pouze 8 lidí, s podobnou úspěšností jsem víceméně počítal, vzhledem k tomu, že učitelům chodí elektronickou poštou denně různé nabídky, ať již reklamní, nebo od žáků a vedení školy a nemohou se jim všem věnovat. Oslovil jsem proto ještě profesní organizaci, s názvem Jednota školských informatiků (zkratkou JŠI) [15], tato instituce sdružuje pedagogy a odborníky, zabývající se ICT ve školství.

Za organizaci se mi ozval její předseda Petr Naske, který mou prosbu o vyplněné dotazníku přeposlal do konference členů JŠI, která podle slov Petra Naske čítá kolem 200 učitelů informatiky. Proto bych chtěl prostřednictvím této práce poděkovat panu Naske za pomoc při sběru dat.

V dalších podkapitolách popíši podrobněji výsledky průzkumu a vlivy na práci z něj odvozené.

6.1 Výsledky průzkumu

Celkově se dotazníku zúčastnilo 32 učitelů informatiky. Jelikož nebyly žádné odpovědi povinné, jsou některé otázky vyplněny méně uživateli.

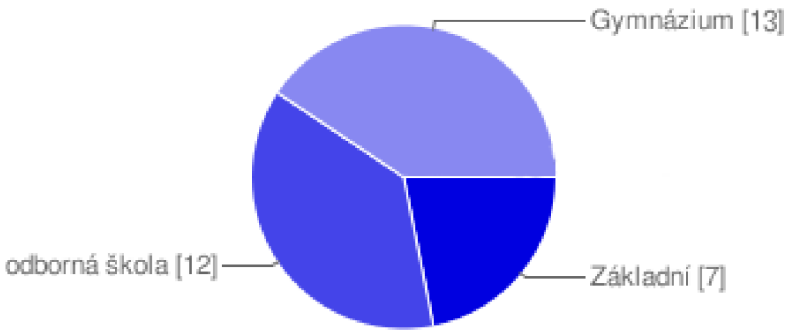
Otázky č.1 až 5 se týkající výuky algoritmizace na škole, kde respondent učí informační technologie. Otázky od čísla 6 se týkají hodnocení vytvořené aplikace.

V hodnocení programu jsem vypsal šest základních vlastností, které pedagogové hodnotili na stupnici od 1 do 5, podobně jako při známkování 1 (výborné), 5 (nedostatečné). Výsledky průzkumu znázorním grafy, které zachycují počet hodnocení danou známkou.

Otázka č.1: *Typ školy*

Otázka slouží k tomu, aby bylo možné zhodnotit další odpovědi vzhledem ke stupni školského zařízení.

Otázka č. 2: *Jaký programovací jazyk pro výuku žáků používáte ?* 6.1



Obrázek 6.1: Odpovědi na 1. otázku (*Typ školy*)

Programovací jazyk	**Celkem**	**ZŠ**	**SŠ**	**Gym**
Pascal	7	1	0	6
Java	2	0	1	1
Python	2	0	1	1
HTML	1	0	1	0
Jazyk C	5	0	1	4
C++	4	0	2	2
Karel	3	0	0	3
JavaScript	2	0	0	2
Baltík	5	3	1	1
C#	1	0	1	0
Delphi	4	0	2	2
PHP	1	0	0	1
Xbase	1	0	1	2

Tabulka 6.1: Souhrn odpovědí na otázku č.2 (*Jaký programovací jazyk pro výuku žáků používáte ?*)

U této otázky v dotazníku byla možnost volby více odpovědí, výsledky jsou shrnuty v tabulce níže 6.1, uvádím pouze ty jazyky, které byly alespoň jednou vybrány a celkový počet výběrů odpovědi upřesňuji podle druhu školy.

Otázka č. 3: *S jakými překážkami vzhledem k vybranému programovacímu jazyku se žáci nejvíce potýkají ?*

U této otázky uživatelé popisovali překážky výuky slovně, nevypíši zde jednotlivé odpovědi, ale uvedu nejčastější připomínky

Velmi často zmiňovaným problémem byla syntaxe jazyka, tu dotázaní zmínili celkem 6krát z 16 odpovědí této otázky. Nejvíce tento problém uváděli učitelé středních škol a gymnázií, kde se vyučuje jeden z pokročilejších programovacích jazyků jako Pascal a jazyk C.

Po syntaxi byl nejčastěji uváděný problém deklarace proměnných a alokace paměti zmíněný celkem 4krát, stejně jako u syntaxe i zde tento problém uvedli učitelé používající některý z pokročilejších jazyků.

Programovací jazyk	Celkem	ZŠ	SŠ	Gym
Baltík	8	3	1	4
Karel	5	0	1	5
Logo	4	0	1	3
Scratch	1	1	0	0

Tabulka 6.2: Souhrn odpovědí na otázku č.2 (*Zkoušeli jste využít některý z grafických programovacích jazyků (nebo nástrojů pro výuku algoritmizace)? Pokud ano, tak jaký ?*)

Posledními zmiňovanými problémy byly algoritmizace úlohy, problémy žáků s logikou a prostředím programu v anglickém jazyce.

Otázka č. 4: *Zkoušeli jste využít některý z grafických programovacích jazyků (nebo nástrojů pro výuku algoritmizace)? Pokud ano, tak jaký ?* 6.2

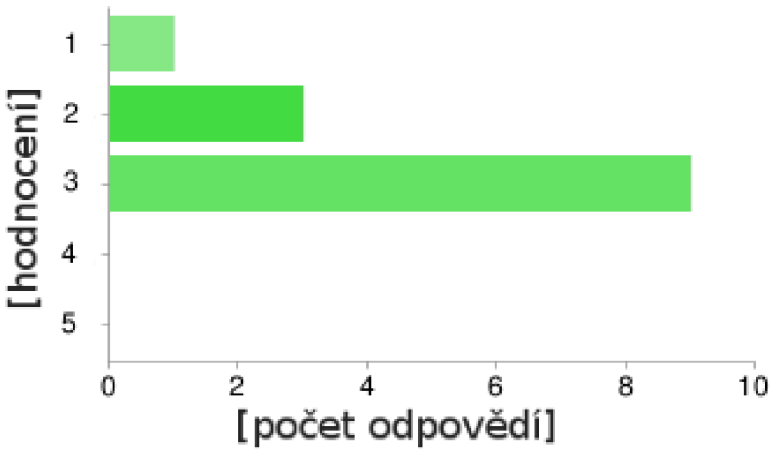
Kladné odpovědi shrnu stejně jako u otázky 3 do tabulky 6.2.

Otázka č. 5: *Jaké výhody a nevýhody vidíte ve Vámi použitém grafickém programovacím jazyce?*

Na tuto otázku uživatelé odpovídali stejně jako u otázky 3 textovou odpovědí. Jako výhody grafických programovacích jazyků byly uvedeny jednoduchost, srozumitelnost, vhodnost pro mladší žáky základních škol a také vyřešení problému se syntaxí.

Mezi nevýhodami pak učitelé zmiňovali, že tyto jazyky jsou pro starší žáky základních a středních škol příliš jednoduché, jako problém učitelé vidí také to, že tyto jazyky nenabízí objektový přístup, nebo že nejsou dostatečně abstraktní a pro žáka může být pak paradoxně těžší přejít na klasické programovací jazyky.

Otázka č. 6: *Grafické zpracování aplikace.* 6.2

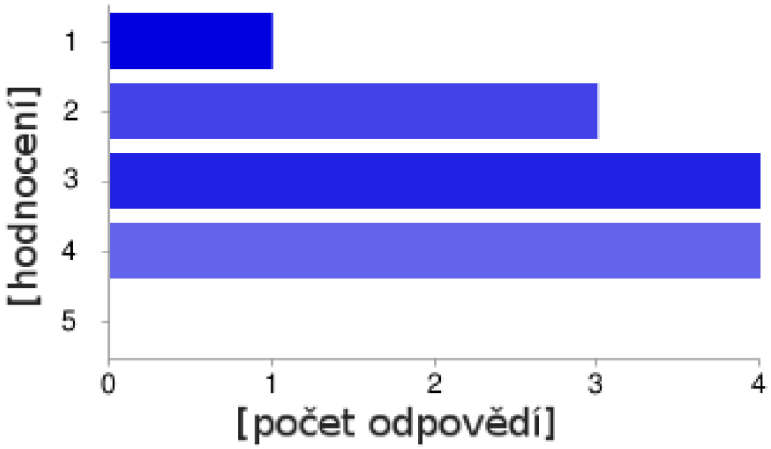


Obrázek 6.2: Odpovědi na 6. otázku (*Grafické zpracování aplikace.*)

Otázka č. 7: *Intuitivnost ovládání aplikace.* 6.3

Otázka č. 8: *Obtížnost zápisu programu v aplikaci.* 6.4

Obrázek 6.3: Odpovědi na 7. otázku (*Intuitivnost ovládání aplikace.*)



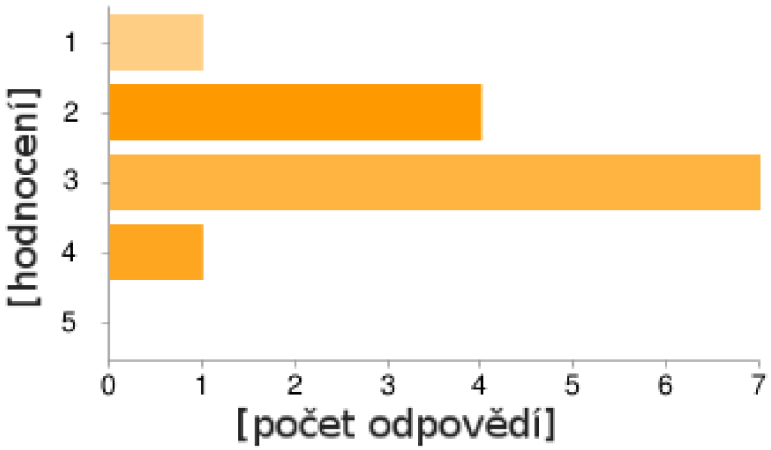
Obrázek 6.4: Odpovědi na 8. otázku (*Obtížnost zápisu programu v aplikaci.*)

Otázka č. 9: *Počet a výběr příkazů v aplikaci.* 6.5

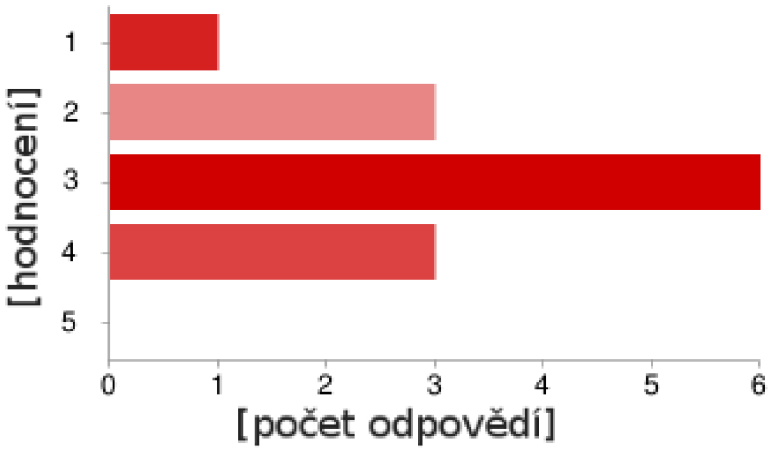
Otázka č. 10: *Vhodnost aplikace k výuce programování (algoritmizace).* 6.6

Otázka č. 11: *Vhodnost aplikace k výuce matematických funkcí.* 6.7

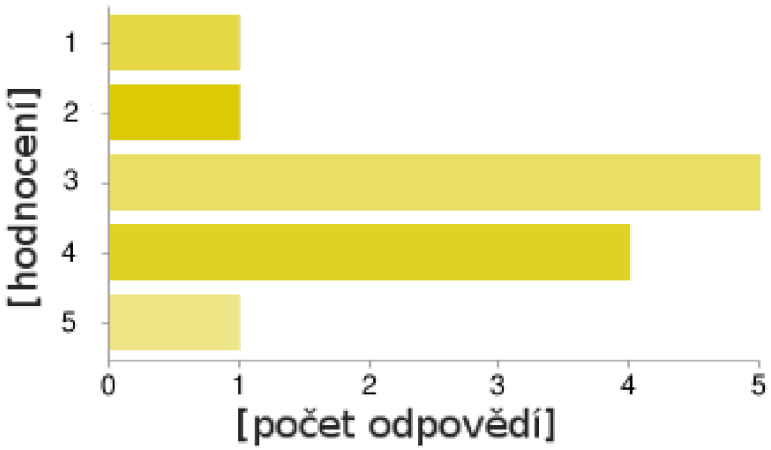
Otázka č. 12: *Uvažujete o vyzkoušení tohoto programu ve výuce ?* 6.8



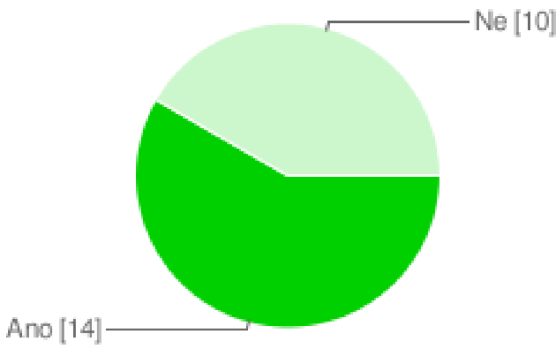
Obrázek 6.5: Odpovědi na 9. otázku (*Počet a výběr příkazů v aplikaci.*)



Obrázek 6.6: Odpovědi na 10. otázku (*Vhodnost aplikace k výuce programování (algoritmizace).*)



Obrázek 6.7: Odpovědi na 11. otázku (*Vhodnost aplikace k výuce matematických funkcí.*)



Obrázek 6.8: Odpovědi na 12. otázku (*Uvažujete o vyzkoušení tohoto programu ve výuce ?*)

Kapitola 7

Závěr

7.1 Zhodnocení dosažených výsledků

V této práci jsem se kromě nutného teoretického úvodu do problému věnoval definování vlastností výukových programů 3.2, které shrnují požadavky na programy pro výuku algoritmizace. Z těchto požadavků jsem pak vycházel praktické části. Provedl jsem zde také shrnutí nabízených programů na výuku algoritmizace na českém trhu a popsal jejich výhody a nevýhody.

Přínosem této práce je mimo jiné shrnutí stavu výuky algoritmizace na Českých školách a to jak z předpisů závazných pro školy 3.4, tak z provedeného dotazníku mezi učiteli výpočetní techniky 6.1.

Ze shrnutí vyplynulo, že čas věnovaný výuce algoritmizace a programování je závislý na přístupu školy k tomuto druhu vzdělávání. Pokud se algoritmizace vyučuje na základní škole, je k výuce většinou zvolený některý z grafických programovacích jazyků. Na středních školách jsou však již voleny standardní programovací jazyky typu Pascal a jazyk C. Z toho také vyplývají problémy výuky. Hlavním problémem je v nejvíce případech pochopení syntaxe na místo učení se algoritmů. Pokud se rozšíří výuka algoritmizace, což může nastat zavedením této výuky do povinných předmětů, bude nutné tyto problémy na školách řešit. Zde se asi jako nejlepší varianta nabízí použití grafických programovacích jazyků, kterými se práce zabývá.

V praktické části práce se podařilo vyvinout použitelný program pro výuku algoritmizace popsaný výše v kapitole 5, důkazem je hodnocení dotázaných pedagogů v kapitole 6.1. V rámci práce jsem vytvořil několik ukázkových projektů v tomto programu, které demonstrují jeho využitelnost a jsou přiloženy v CD.

Aplikace sice nedosahuje kvalit profesionálních řešení jako Baltík, nebo Petr, ale vyplňuje mezeru mezi snadno dostupnými grafickými jazyky jako Logo, nebo Karel, kde uživatelé programují textově a výsledek algoritmu je znázorněn graficky, a placenými programy jako již zmíněný Baltík a Petr, kde uživatel tvoří příkazy skládáním grafických primitiv. Aplikace umožňuje zápis programů v grafické formě podobným stylem jako v programu Petr, ale i textově. Jistým přínosem aplikace je také možnost převodu programu z grafické formy do textové, tak může uživatel porovnat jak by vypadal jeho grafický zápis textově, popřípadě může přejít, po získání dostatečných zkušeností, z grafické formy zápisu do textové. Uživatelé se tedy mohou vyzkoušením tohoto programu seznámit s algoritmizací, a pokud v aplgoritmizaci najdou zalíbení a nebude jim program již stačit, mohou si pořídit některý profesionální.

Pro některé školy, které se výukou algoritmizace a programováním zabývají jen okrajově,

jak bylo zmíněno v kapitole 3.4, může tato aplikace být dobrým a dostupným prostředkem, jak žákům algoritmizaci vysvětlit.

7.2 Další možné rozšíření práce

V praktické části práce existuje hodně možností, jak práci dále rozvíjet. Určitě by bylo přínosem do aplikace začlenit příkazy, pro obsluhu klávesnice a myši, od zavedení tohoto vylepšení by se odvíjela další úprava způsobu programování, aby umožňovalo obsluhu událostí.

Jako důležité rozšíření se nabízí ladící režim, kde by uživatelé mohli krokovat program a řešit případné problémy. Dalších nápadů, jak program vylepšit, lze nalézt nespočet, od přidávání nových příkazů, přes vylepšení grafického rozhraní, až po větší podporu multimédií.

Aplikace by mohla být také v budoucnu využita ve výuce, a získané poznatky z užívání aplikace studenty, by mohly být výbornou zpětnou vazbou a zdrojem užitečných vylepšení. Z poslední otázky dotazníku vyplývá, že někteří učitelé mají v plánu tento program vyzkoušet, proto je velmi pravděpodobné, že rozšiřování možností aplikace se může v budoucnu vyvíjet právě touto zpětnou vazbou od pedagogů.

Literatura

[1] Doc. RNDr. Kreslíková, J., Csc.: *Základy programování - úvod do algoritmizace.* FIT VUT v Brně, 2005.

[2] Hotař, R.: *Diplomový projekt: Grafický výukový systém.* FIT VUT v Brně, 2010.

[3] Hotař, R.: Dotazník výuky programování na středních a základních školách [online]. `http://spreadsheets.google.com/viewform?formkey=dDA2TWh2eGR4SGtHMjlCTEpXNGMwNWc6MA`, 2010-05-01 [cit. 2010-05-20].

[4] Ierusalimschy, R.; Celes, W.; de Figueiredo, L. H.: The Programming Language Lua [online]. `http://www.lua.org/`, 2010-03-03 [cit. 2010-05-20].

[5] Jeřábek, J.; Tupý, J. a kolektiv: Rámcový vzdělávací program pro základní vzdělávání [online]. `http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPZV_2007-07.pdf`, 2007-09-01 [cit. 2010-05-20].

[6] Nokia Corporation and/or its subsidiaries: Qt - Cross-platform application and UI framework [online]. `http://qt.nokia.com/`, 2010 [cit. 2010-05-20].

[7] Přispěvatelé Wikipedie: Didaktické zásady [online]. `http://cs.wikipedia.org/w/index.php?title=Didaktick=3808634`, 2010-04-03 [cit. 2010-05-20].

[8] Přispěvatelé Wikipedie: Karel (programovací jazyk) [online]. `http://cs.wikipedia.org/w/index.php?title=Karel_(programovacoldid=5296859`, 2010-05-03 [cit. 2010-05-20].

[9] Přispěvatelé Wikipedie: Logo (programming language) [online]. `http://en.wikipedia.org/w/index.php?title=Logo_(programming_language)&oldid=362177787/`, 2010-05-14 [cit. 2010-05-20].

[10] Průcha, J.: *Přehled pedagogiky.* Portál, 2000, iSBN 978-80-7367-567-7.

[11] Vacek, P.: *Morální vývoj v psychologických a pedagogických souvislostech.* Gaudeamus UHK, 2000, iSBN 80-7041-148-1.

[12] WWW stránky: Didaktické zásady [online]. `http://www.infogram.cz/article.do?articleId=1613`, 2010-01-01 [cit. 2010-05-20].

[13] WWW stránky: History of Game Maker [online].
http://wiki.yoyogames.com/index.php/Game_Maker_History, 2010-01-15
[cit. 2010-05-20].

[14] WWW stránky: Popis programovacího nástroje Petr [online].
http://www.gemtree.cz/peter.htm, 2010-03-01 [cit. 2010-05-20].

[15] WWW stránky: Jednota školských informatiků [online]. http://www.jsi.cz/,
2010-05-20 [cit. 2010-05-20].

[16] WWW stránky: LEGO.com MINDSTORMS [online].
http://mindstorms.lego.com/, 2010 [cit. 2010-05-20].

[17] WWW stránky: Scratch, imagine, program, share [online].
http://scratch.mit.edu/, 2010 [cit. 2010-05-20].

[18] WWW stránky: Výukové programovací nástroje C# pro děti, mládež, i dospělé
[online]. http://www.sgpsys.com/cz/, 2010 [cit. 2010-05-20].

[19] WWW stránky: What is Alice? [online].
http://www.alice.org/index.php?page=what_is_alice/what_is_alice, 2010
[cit. 2010-05-20].

Příloha A

Obsah CD

- Adresář *bin* obsahuje přeložený program pro prostředí MS Windows.
- Adresář *src* obsahuje zdrojové projekty programu.
- Adresář *text* obsahuje tuto zprávu ve formátu *pdf*.
- Adresář *latex* obsahuje zdrojový text této zprávy
- Adresář *examples* obsahuje ukázkové projekty.
- Adresář *doc* obsahuje jednoduchý uživatelský manuál.