SIMULACE PRŮCHODU PROGRAMU VÝVOJOVÝM DIAGRAMEM PRO VÝUKU ALGORITMIZACE

Bakalářská práce

Miroslav Bartyzal

Vedoucí bakalářské práce:
doc. PaedDr. Jiří Vaníček, Ph.D.

Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky
2012

PROHLÁŠENÍ

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích

dne: …… ……

Podpis autora práce

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Miroslav BARTYZAL**
Osobní číslo:	**P09501**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie ve vzdělávání**
Název tématu:	**Simulace průchodu programu vývojovým diagramem pro výuku algoritmizace**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Úkolem studenta je vytvořit aplikaci umožňující simulaci průchodu programu vývojovým diagramem pro účel výuky algoritmizace. Simulace průchodu vývojovým diagramem bude realizována animací, důraz bude kladen na srozumitelnost, přívětivou formu vizualizace a intuitivní ovládání.
Sestavení vývojového diagramu bude umožněno dvěma způsoby. Uživatel si bude moci vývojový diagram sestavit sám, pomocí vestavěného editoru, nebo využít funkci automatického vygenerování diagramu z vloženého zdrojového kódu programovacího jazyka Pascal nebo Java. Podporovány budou základní příkazy strukturovaného programování, jako např. proměnné, pole, rozhodování, cykly, vstupy a výstupy. Aplikace bude následně umožňovat simulaci průchodu takto vytvořeným vývojovým diagramem tak, že bude zřejmé, jak jím program prochází, jak se mění parametry cyklu, jakým způsobem rozhoduje a jak se mění proměnné. Diagram bude možno opatřit komentáři. Výsledný diagram bude možné uložit opět ve formě zdrojového kódu.
Student dále vytvoří sadu příkladů pro použití ve výuce a ověří aplikaci v terénu. Součástí práce bude rovněž uživatelská příručka a kompletní dokumentace k aplikaci.

Rozsah grafických prací:	**CD ROM**
Rozsah pracovní zprávy:	**60**
Forma zpracování bakalářské práce:	**tištěná**
Seznam odborné literatury:	**viz příloha**

Vedoucí bakalářské práce:	**PaedDr. Jiří Vaníček, Ph.D.** Katedra informatiky
Datum zadání bakalářské práce:	**12. dubna 2011**
Termín odevzdání bakalářské práce:	**30. dubna 2012**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 8. dubna 2010

Příloha zadání bakalářské práce

Seznam odborné literatury:

1. TAUFER, I., HRUBINA, J., TAUFER, J.. Algoritmy a algoritmizace: vývojové diagramy, sbírka řešených příkladů. Pardubice: Univerzita Pardubice, 2001. Kopp, České Budějovice, 1997. ISBN 80-901342-2-X.

2. CHYTIL, Jiří. Programujte.com [online]. 24. 07. 2005 [cit. 2011-04-03]. Vývojové diagramy - 1. díl.
Dostupné z WWW: <http://programujte.com/?akce=clanek&cl=2005080105-vyvojove-diagramy-1-dil>.

3. CHYTIL, Jiří. Programujte.com [online]. 11. 10. 2005 [cit. 2011-04-03]. Vývojové diagramy - 2. díl.
Dostupné z WWW:
<http://programujte.com/?akce=clanek&cl=1970010171-vyvojove-diagramy-2-dil>.

4. DIVIŠ, Jozef. Algoritmizace [online]. c2011 [cit. 2011-04-03].
Dostupné z WWW: <http://www.spsemoh.cz/vyuka/algor/index.htm>.

5. DIVIŠ, Jozef. PASCAL [online]. c2011 [cit. 2011-04-03].
Dostupné z WWW: <http://www.spsemoh.cz/vyuka/pascal/index.htm>.

6. PADRTA, David. Algoritmy a programování [online]. c1999, poslední změna 02. června 2009 [cit. 2011-04-03].
Dostupné z WWW:
<http://www.sse-lipniknb.cz/7ucivo/vypocetka/pascal/pascal/alg_prog.html>.
Aivosto : Visustin v6 Flow chart generator [online]. >1997 [cit. 2011-04-03].
Dostupné z WWW:
<http://www.aivosto.com/visustin.html>.

7. Aivosto : Visustin v6 Flow chart generator [online]. >1997 [cit. 2011-04-03].
Dostupné z WWW: <http://www.aivosto.com/visustin.html>.

8. VEUPL : ALVIS Live! [online]. 5.2.2005, Last modified on May 4, 2007 [cit. 2011-04-03].
Dostupné z WWW: <http://eecs.wsu.edu/ veupl/soft/alvis/index.htm>.

9. VOBORNÍK, Petr. Program Algoritmy [online]. c2005 [cit. 2011-04-03]. Dostupné z WWW: <http://algds.cronos.cz/>.

ANOTACE

Práce se zabývá vývojem aplikace pro rychlé a efektivní sestavování algoritmů pomocí vývojových diagramů a jejich následnou vizualizaci průchodu pro výuku algoritmizace.

První část práce sestává z popisu základní terminologie, v níž jsou popsány zejména jednotlivé symboly vývojového diagramu. Další části práce jsou věnovány analýze zpracovávaného tématu, návrhu aplikace a interesantním úsekům jejího vývoje. Poslední části práce pak shrnují průběh ověření aplikace v praxi a výsledek této práce.

Klíčová slova:

Vývojový diagram, algoritmizace, algoritmus, výuka, program, vizualizace, animace, průchod, diagram, ICT

ABSTRACT

The work is concerned with development of application for fast and efficient way of assembling algorithms using flowcharts and their subsequent walkthrough visualization for purpose of teaching algorithms.

First part of the work consists of basic terminology description, which describes in particular the different flowchart symbols. Other parts of the work are devoted to the analysis of the treated subject, application design and interesting sections of development. The last part of the thesis summarizes the verification of the application in practice and the results of this work.

Keywords:

Flowchart, algorithms, algorithm, teaching, program, visualization, animation, walkthrough, diagram, ICT

PODĚKOVÁNÍ

Rád bych touto cestou poděkoval svému vedoucímu bakalářské práce doc. PaedDr. Jiřímu Vaníčkovi, Ph.D. za cenné rady a doporučení při vedení mé bakalářské práce.

Děkuji rovněž učiteli Mgr. Aleši Janatovi za jeho ochotu, čas a zpětnou vazbu při testování aplikace. Váš zájem o mou bakalářskou práci pro mne byl během programování velice motivující.

Dále bych chtěl poděkovat učiteli Ing. Josefu Turkovi za jeho ochotu, laskavost a nadšení během praktického testování aplikace.

Poděkování patří také mé přítelkyni Janě, rodině a přátelům, kteří mne při studiu podporovali.

OBSAH

1 Úvod

Grafické znázornění algoritmů je nedílnou součástí výuky algoritmizace a úvodu do programování. Jedním z nejpoužívanějších prostředků pro takové znázornění je vývojový diagram, jenž poskytuje nejen názornou formu zápisu algoritmu, ale i možnost přehledné počítačové simulace jeho průchodu.

Prostřednictvím této bakalářské práce bych rád podpořil výuku algoritmizace nástrojem, pomocí něhož by uživatel byl schopen jednoduchým a intuitivním způsobem vývojový diagram vytvořit a následně pozorovat simulaci jeho algoritmického průchodu.

1.1 Motivace

Vývojové diagramy jsou v současnosti běžnou součástí výuky algoritmizace na většině středních škol. Jejich nevýhodou jsou však zejména obtížné možnosti průběžných úprav, kdy je žák často nucen celý diagram překreslit či znovu přeorganizovat většinu symbolů i s jejich spojnicemi. Této věci nenapomáhá ani skutečnost, že školy často nedisponují specializovaným softwarem pro tvorbu vývojových diagramů a jsou tak nuceny využívat aplikací, které pro tento účel nejsou příliš vhodné (*viz kapitola 3.1.1*).

Rozhodl jsem se proto pokusit se o vytvoření takové aplikace, která by výše uvedené komplikace odstranila, a nabídnout tak školám specializovaný nástroj pro tvorbu vývojových diagramů.

Dostupného softwaru pro simulaci algoritmického průchodu vývojového diagramu je v současnosti velice nízký počet, přičemž nalezená řešení trpí četnými nedostatky jak v oblasti návrhu diagramů, tak při vizualizaci jejich průchodu. Rád bych tedy do výuky zavedl takové řešení, které by tyto nedostatky redukovalo na minimum a eventuálně nabídl i něco navíc.

1.2 Cíle práce

Cílem práce je vytvořit aplikaci umožňující simulaci průchodu programu vývojovým diagramem pro účel výuky algoritmizace. Simulace průchodu vývojovým diagramem má být realizována animací, důraz by měl být kladen na srozumitelnost, přívětivou formu vizualizace a intuitivní ovládání.

Sestavení vývojového diagramu má být umožněno dvěma způsoby. Uživatel si bude moci vývojový diagram sestavit sám, pomocí vestavěného editoru, nebo využít funkci automatického vygenerování diagramu z vloženého zdrojového kódu programovacího jazyka Pascal nebo Java. Podporovány mají být základní příkazy strukturovaného programování, jako např. proměnné, pole, rozhodování, cykly, vstupy a výstupy. Diagram bude rovněž možné opatřit komentáři.

Aplikace bude následně umožňovat simulaci průchodu takto vytvořeným vývojovým diagramem tak, že bude zřejmé, jak jím program prochází, jak se mění parametry cyklu, jakým způsobem rozhoduje a jak se mění proměnné.

Výsledný vývojový diagram bude možné uložit buď ve formě souboru pro pozdější opětovné načtení, nebo opět ve formě zdrojového kódu podporovaného programovacího jazyka.

Pro výslednou aplikaci bude dále vytvořena uživatelská příručka a sada předpřipravených příkladů vývojových diagramů. Součástí práce bude rovněž technická dokumentace k aplikaci.

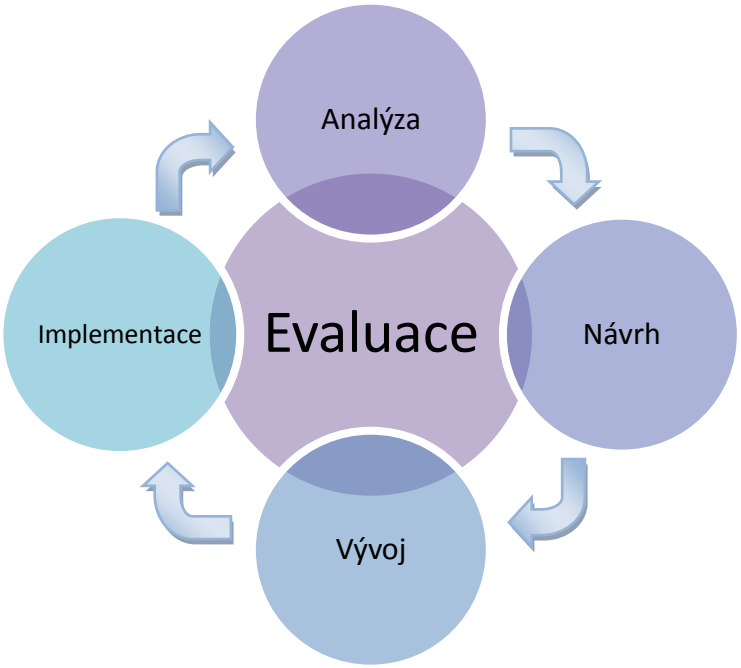
Přínos celé práce bude ověřen na některé střední škole, kde bude aplikace představena a otestována v praktickém provozu.

1.3 METODA PRÁCE

Pro vytváření výukového software se využívají různé přístupy. Mezi nejpoužívanější přístupy pak patří tzv. ADDIE model, který v pěti základních etapách definuje efektivní metodiku postupu [6]. Metoda této práce je založena právě na základě ADDIE modelu, od nějž se odvíjí i její struktura, která je rozčleněna podle jeho fází (Obr. 1).

Ve fázi analýzy byla zkompletována potřebná literatura týkající se vývojových diagramů, algoritmizace a programování. Dále byl na toto téma proveden dotazníkový průzkum na středních školách disponujících obory ICT a zanalyzovány existující softwarové projekty podobného nebo stejného tématu.

Na základě zjištění získaných z provedené analýzy byl pak vytvořen detailní návrh konkrétního řešení, které splňovalo stanovené cíle. V této fázi byl navrhnut i stěžejní objektový návrh kostry aplikace a harmonogram pro její vývoj.



OBR. 1 ADDIE MODEL

Ve vývojové etapě byla vytvářena samotná aplikace, jejíž vývoj byl metodou „rozděl a panuj“ rozčleněn na několik dílčích částí. Většina částí, které nějakým způsobem tvořily pilíře výsledné aplikace, byla následně konzultována a později otestována panem učitelem Mgr. Alešem Janatou, který vývojové diagramy ve výuce aktivně používá. Na základě jeho zpětné vazby pak probíhaly případné další úpravy aplikace. Po dosažení implementovatelné[1] verze aplikace následovalo vytvoření uživatelské příručky (*viz Příloha A*), technické dokumentace a sady příkladů (*lze nalézt na přiloženém CD*).

Ověření aplikace v praxi, jenž předchází samotné implementační části[1] ADDIE modelu, byla provedena na Střední odborné škole elektrotechnické v Hluboké nad Vltavou. Následovalo vyhodnocení, zda vytvořená práce zjednodušuje a urychluje práci s vývojovými diagramy, zda je pro žáky snadno ovladatelná a zda v ní učitelé vidí potenciál pro zlepšení kvality výuky.

Samotné nasazení aplikace do výuky bude realizováno rozesláním aplikace těm školám, které o ni při dotazníkovém průzkumu projevily zájem.[2]

[1] Implementací se v ADDIE modelu rozumí „stěžejní krok, v němž je vytvořená práce nasazena a spuštěna cílovému publiku“ ([8], s. 31).

[2] Jedná se celkem o 24 středních škol.

1.4 PŘÍLOHY PRÁCE

K práci je připojena příloha v podobě uživatelské příručky, která popisuje základní principy a koncepce vytvořené aplikace (*Příloha A*). Uživatelskou příručku lze tedy považovat i za popis výsledku této práce. Dále je k práci přiložen dotazník výzkumu výuky algoritmizace (*Příloha B*).

Součástí tištěné práce je pak disk CD, jež obsahuje tyto položky:

- Elektronická verze této práce (formát PDF)
- Elektronická verze uživatelské příručky k aplikaci (formát PDF)
- Elektronická verze dotazníku výzkumu výuky algoritmizace (formát PDF)
- Sada předpřipravených příkladů vývojových diagramů, které lze načíst a zobrazit vytvořenou aplikací (formát XML)
- Technická dokumentace k aplikaci (formát HTML)
- Spustitelná portable[3] verze aplikace[4] (formát JAR)
- Zdrojový kód aplikace v podobě projektu NetBeans IDE 7[5]

[3] Portable aplikaci není nutné instalovat a lze ji spustit i s omezenými uživatelskými právy.[15]

[4] Ke spuštění aplikace je nutné mít na počítači nainstalovanou aktuální verzi Java SE Runtime Environment (JRE), která je zdarma ke stažení na oficiálních stránkách Oracle.

[5] NetBeans IDE je vývojového prostředí pro programovací jazyk Java.[16]

2 TERMINOLOGIE

Pro to, aby bylo možné zabývat se tvorbou softwaru pro simulaci průchodu programu vývojovým diagramem je nutné, ujasnit si některé základní pojmy, s nimiž bude operováno. Jedná se především o pojmy algoritmizace, algoritmus a vývojový diagram. Těmto termínům budou věnovány následující podkapitoly.

2.1 ALGORITMIZACE

„Algoritmizace je metodický přístup k vytváření programu. Zabývá se formulací postupů řešení daného problému. Výsledkem algoritmizace je algoritmus, což je posloupnost příkazů popisující řešení daného problému.“([9], s. 10)

Je důležité poznamenat, že ačkoliv je obor informatiky obecně znám svou vysokou rychlostí evoluce, algoritmizace zůstává časem nedotčena (ibid.). Algoritmizace totiž popisuje řešení problému v obecné rovině a není tak závislá na realizaci konkrétním programovacím jazykem.

2.2 ALGORITMUS

„Algoritmus je schematický postup pro řešení určitého druhu problémů, který je prováděn pomocí konečného množství přesně definovaných kroků.“[17] Každý algoritmus se pak skládá z více dílčích kroků, které na základě vložených měnitelných vstupů vedou k získání kýženého výsledku. Tento výsledek je pak přesměrován na výstup algoritmu, který představuje odpověď na řešený problém.

2.2.1 VLASTNOSTI ALGORITMU

Algoritmus by měl splňovat jisté základní vlastnosti, které zaručí jeho správné vykonání a vlastní korektnost. Tyto vlastnosti zahrnují elementárnost, determinovanost, konečnost, rezultativnost a obecnost ([4], s. 5). Jednotlivě budou popsány v následujících blocích.

Elementárnost
Algoritmus tvoří konečný počet jednoduchých (elementárních), snadno realizovatelných kroků (ibid.).

Determinovanost
Determinovanost neboli určenost či jednoznačnost algoritmu znamená, že v žádné z jeho částí nesmí být pochyb o tom, co je třeba v daném místě provést a jaký krok bude následovat ([4], s. 5).

Konečnost
Každý algoritmus musí po konečném počtu kroků skončit. Počet těchto kroků je irelevantní, musí však být pro každý jednotlivý vstup konečný (ibid.).

Rezultativnost
Rezultativnost algoritmu znamená, že musí po vykonání konečného počtu kroků skončit a poskytnout výsledek (ibid.). Výsledek neboli výstup algoritmu tvoří veličinu, která je v požadovaném vztahu k zadaným vstupům a tím tvoří odpověď na problém, který algoritmus řeší.

Obecnost
Vlastnost obecnosti či hromadnosti algoritmus předurčuje k širšímu použití v dané problematice. Algoritmus by nikdy neměl být zaměřen na řešení jediné úlohy (výpočet 1+1), nýbrž na její širší, obecné pojetí (součet dvou čísel) ([5], s. 12).

2.2.2 Zápis algoritmu

Přirozený jazyk není pro zápis algoritmů příliš vhodný, zejména kvůli jeho časté nejednoznačnosti. Aby tedy zápisem algoritmu nevzniklo porušení jeho determinovanosti, jsou pro účel jeho záznamu zavedeny speciální formy zápisu. Obecně lze zápis algoritmu rozdělit do dvou skupin - na zápis v textové formě a na zápis ve formě grafické.

Pro demonstraci obou forem zápisů bude jako příklad použit jednoduchý algoritmus pro určení většího ze dvou čísel.

2.2.2.1 Textová forma

Programovací jazyk
Programovací jazyky jsou umělé jazyky vytvořené pro zápis algoritmu v podobě počítačového programu ([10], s. 1). „Mají přísně definovanou syntaxi, což je systém symbolů a pravidel, kterými se řídí formální zápis programu, a sémantiku, která určuje význam programu.“(ibid.)

```
int cislo1;
int cislo2;
BufferedReader reader;
reader = new BufferedReader(new InputStreamReader(System.in));
cislo1 = Integer.valueOf(reader.readLine());
cislo2 = Integer.valueOf(reader.readLine());
if (cislo1 > cislo2) {
    System.out.println(cislo1);
} else {
    System.out.println(cislo2);
}
```

OBR. 2 ALGORITMUS V PODOBĚ PROGRAMOVACÍHO JAZYKA

Pseudokód

Pseudokód je jazyk určený pro popis algoritmů, který se podobá programovacímu jazyku [19]. Na rozdíl od programovacích jazyků však nemá definovanou syntaxi a je tak pro člověka čitelnější, základní konstrukce však vychází právě z programovacích jazyků.

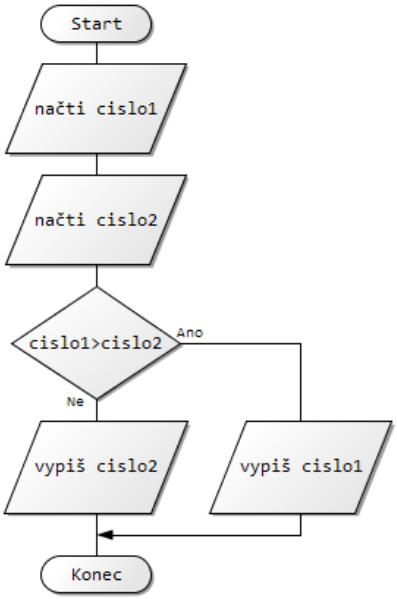
```
cislo1 = načti
cislo2 = načti
když cislo1 > cislo2 pak
    vypiš cislo1
jinak
    vypiš cislo2
```

OBR. 3 ALGORITMUS V PODOBĚ PSEUDOKÓDU

2.2.2.2 GRAFICKÁ FORMA

Vývojový diagram

Vývojový diagram představuje grafickou reprezentaci nadnárodní normou definovaných symbolů, které reprezentují jednotlivé dílčí kroky algoritmu a jejich výpočetní operace ([4], s. 10). Podrobněji budou popsány v samostatné kapitole 2.3.



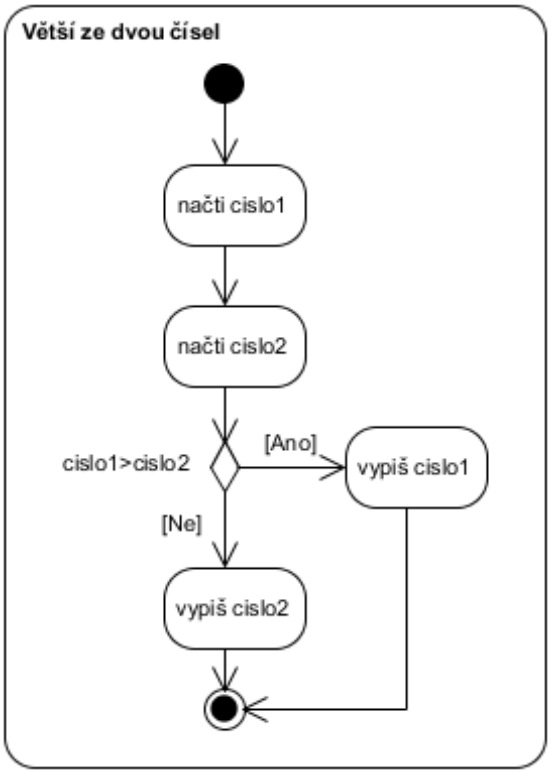
OBR. 4 ALGORITMUS V PODOBĚ VÝVOJOVÉHO DIAGRAMU

Unified Modeling Language (UML)

UML je unifikovaný modelovací jazyk s definovanou syntaxí a sémantikou, užívaný zejména pro vizualizaci, záznam, navrhování a dokumentaci programových systémů [30]. Nabízí pak standardní způsob zápisu návrhu nejen systému s konceptuálními prvky, ale i prvky konkrétními jako jsou např. jednotlivé kroky algoritmu či příkazy programovacího jazyka.

UML pak disponuje mnoha různými druhy diagramů, jež jsou členěné do dvou hlavních kategorií: diagramy popisující strukturu (např. diagram tříd) a diagramy popisující chování (např. diagram aktivit, diagram užití, stavový diagram) [31]. Je-li požadována dokumentace činnosti či procesu, který se skládá z posloupnosti navazujících kroků (algoritmus), je vhodné použít *diagramy aktivity* z kategorie diagramů chování.

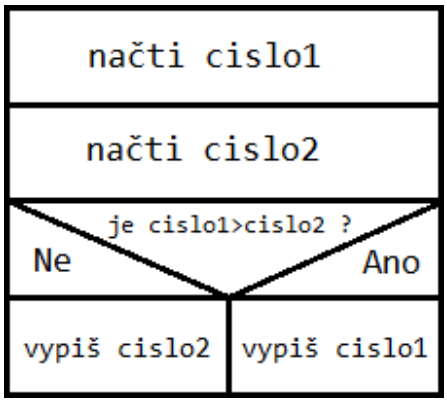
Diagram aktivit (Obr. 5) je velice podobný právě vývojovému diagramu programu s tím rozdílem, že je koncipován obecněji a nedisponuje tak specifickými symboly pro zobrazení konkrétních pokročilých příkazů algoritmizace, jako jsou např. cykly, vstupy a výstupy. Cyklus je pak zapisován symbolem podmínky nebo pomocí tzv. expanzního regionu [32].



OBR. 5 ALGORITMUS V PODOBĚ ACTIVITY DIAGRAMU

Strukturogram

Strukturogramy nebo také diagramy Nassi-Shneidermana jsou obdobou vývojových diagramů, nejsou však definovány normou. Podle Taufera (Taufer, 2009: 10) „představovaly určitý pokus o algoritmický jazyk umožňující zápis pouze strukturovaných algoritmů“, v praxi se však neujaly a dnes se již prakticky nepoužívají (ibid.).



OBR. 6 ALGORITMUS V PODOBĚ STRUKTUROGRAMU

2.3 VÝVOJOVÝ DIAGRAM

„Vývojový diagram je symbolický algoritmický jazyk, který se používá pro názorné zobrazení algoritmu zpracování informací a jejich případnou publikaci.“([4], s. 18) Vývojový diagram se skládá z přesně definovaných značek jednoznačného významu, které jsou mezi sebou propojeny dle jejich vzájemné souvislosti. Jeho použití je vhodné zejména pro prvotní výuku programování či algoritmizace, protože umožňuje přehledně a graficky zobrazit jednotlivé kroky postupu daného řešení (algoritmu), včetně vyznačení všech jeho možných alternativ. Využívá se však i jako univerzální prostředek k vyjádření algoritmu, který je nezávislý na konkrétním programovacím jazyce.

Zápis vývojových diagramů je definován českou státní normou ČSN ISO 5807 „Zpracování informací - Dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy programu a diagramy zdrojů systému“ [1], která reprezentuje nadnárodní normu ISO 5807:1985 „Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts“.

Norma specifikuje symboly používané v dokumentaci zpracování informací a poskytuje návod pro jejich použití v rámci několika druhů vývojových diagramů. Tato práce se zabývá pouze vývojovým diagramem programu, následující kapitoly proto budou věnovány právě jemu.

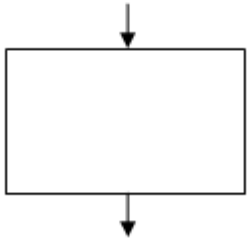
2.3.1 SYMBOLY VÝVOJOVÝCH DIAGRAMŮ

Symbol vývojového diagramu představuje grafickou značku označující jeden krok algoritmu. Tato značka má normou definovaný tvar a význam ([4], s. 18). Pro upřesnění konkrétní funkce symbolu se do jeho vnitřku umisťuje doprovodný text, jehož způsob zápisu a symbolika není normou nijak definována. Doporučuje se však použití krátkého a výstižného řetězce za použití matematických značek.

V následujících podkapitolách budou postupně popsány tvary a funkce jednotlivých symbolů vývojového diagramu programu.

2.3.1.1 ZPRACOVÁNÍ

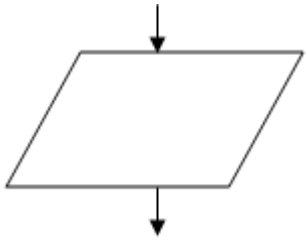
Tento symbol představuje jakýkoliv druh funkce, během níž dochází k transformaci dat (například sečtení dvou čísel či jiná matematická operace) ([5], s. 17). Symbol zpracování má vždy právě jeden vstup a právě jeden výstup.



OBR. 7 SYMBOL ZPRACOVÁNÍ

2.3.1.2 DATA

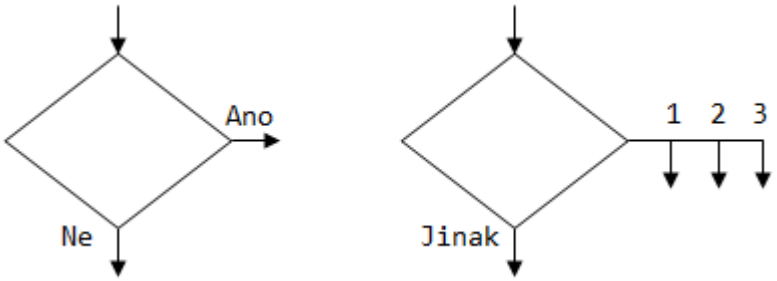
Symbol dat reprezentuje operace s vstupními nebo výstupními daty ([5], s. 17). Používá se tehdy, je-li programu potřeba data k jejich zpracování dodat nebo má-li algoritmus data zobrazit. Symbol může představovat vždy jen jednu z těchto dvou operací a je tedy nutné tyto činnosti od sebe náležitě rozlišit. Pro vstupní operaci se používá např. slovní popis „Čti:“ a popis „Zobraz:“ pro operaci výstupní. Symbol má vždy právě jeden vstup a právě jeden výstup.



OBR. 8 SYMBOL DATA

2.3.1.3 ROZHODOVÁNÍ

Symbol rozhodování slouží k větvení algoritmu na základě podmínkového výrazu, který je uveden uvnitř. Představuje tedy rozhodovací nebo přepínací funkci, přičemž má vždy právě jeden vstup a dva nebo více výstupů ([4], s. 19). Právě jeden z výstupů, jež musí být korektně označen, je pak aktivován na základě vyhodnocení vlastní podmínky symbolu.

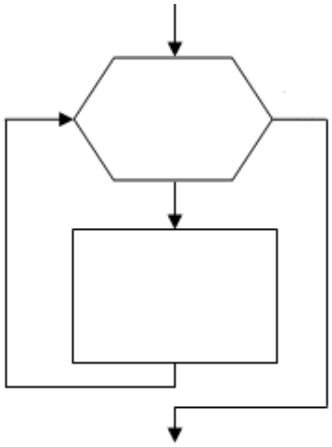


OBR. 9 SYMBOL ROZHODOVÁNÍ

2.3.1.4 PŘÍPRAVA

Tento symbol představuje úpravu činnosti, která mění vlastní postup činnosti následující ([4], s. 19). Symbol je obecně používán jako znázornění cyklu s pevným počtem opakování a úprava činnosti tedy znamená přiřazení následující nebo inicializační hodnoty do proměnné cyklu.

Symbol obsahuje právě dva vstupy a právě dva výstupy. První vstupně-výstupní pár slouží k sekvenčnímu účelu, kdy navazuje na symbol předešlý a následující, zatímco pár druhý má funkci vstupu a výstupu do samotného těla cyklu.



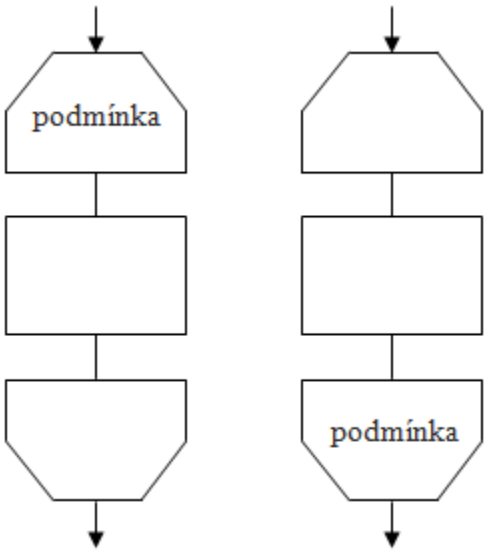
OBR. 10 SYMBOL PŘÍPRAVA

2.3.1.5 MEZ CYKLU

Tento symbol představuje ve dvou částech začátek a konec cyklu ([1], s. 9). Podmínkový výraz, který rozhoduje o vstupu do těla cyklu, je deklarován buď v prvním, nebo ve druhém mezním symbolu. Volitelně lze do obou těchto symbolů umístit identifikátor.

Norma ČSN ISO 5807 z roku 1996 definuje, že podmínkový výraz umístěný v mezním symbolu cyklu podmiňuje *ukončení* cyklu. Současné moderní programovací jazyky[6] však od této konvence upustily a v hlavičkách cyklů jsou deklarovány podmínky rozhodující o jeho *vykonání*.

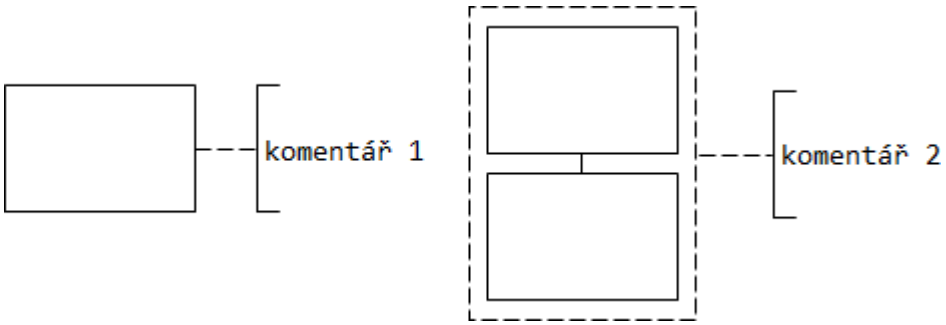
Na základě této skutečnosti jsem se tedy rozhodl používat modernější z těchto konvencí, protože výsledná aplikace má být využívána pro výuku algoritmizace a programování, nikoliv jejich historie.



OBR. 11 SYMBOL MEZ CYKLU

2.3.1.6 ANOTACE

„Symbol anotace se používá pro připojení (za účelem objasnění) popisných komentářů nebo vysvětlujících poznámek.“([1], s. 11) Anotace se připojuje pomocí přerušované čáry přímo k symbolům nebo jejich skupině.

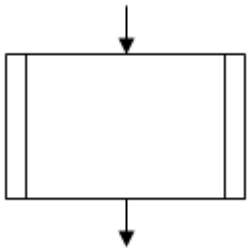


OBR. 12 SYMBOL ANOTACE

[6] Jedná se například o jazyky C, C#, C++, Java, PHP, Python a mnoho dalších.

2.3.1.7 PŘEDDEFINOVANÉ ZPRACOVÁNÍ

Symbol předdefinovaného zpracování neboli podprogram znázorňuje pojmenované zpracování, které může obsahovat větší počet kroků, jež jsou specifikovány jinde [1], s. 7).



OBR. 13 SYMBOL PŘEDDEFINOVANÉ ZPRACOVÁNÍ

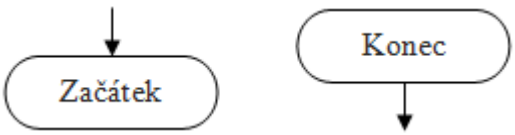
2.3.1.8 SPOJKA

Tento symbol představuje vstup nebo výstup z jedné části téhož vývojového diagramu do části druhé a používá se k přerušení spojnic v případě, hrozilo-li by jejich křížení ([1], s. 11, 15). Spojka se tedy ve vývojovém diagramu vyskytuje vždy v páru, přičemž má právě jeden vstup nebo jeden výstup. Tyto párové spojky musí rovněž obsahovat stejné jedinečné označení.

OBR. 14 SYMBOL SPOJKA

2.3.1.9 MEZNÍ ZNAČKA

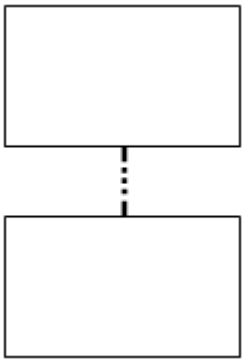
Mezní značka představuje vstup z vnějšího prostředí do programu nebo výstup z programu do vnějšího prostředí ([1], s. 11). Používá se pro označení začátku a konce programu nebo jeho samostatně zpracované části (podprogramu). Stejně jako spojka může mít buď jeden vstup, nebo jeden výstup.



OBR. 15 SYMBOL MEZNÍ ZNAČKA

2.3.1.10 VÝPUSTKA

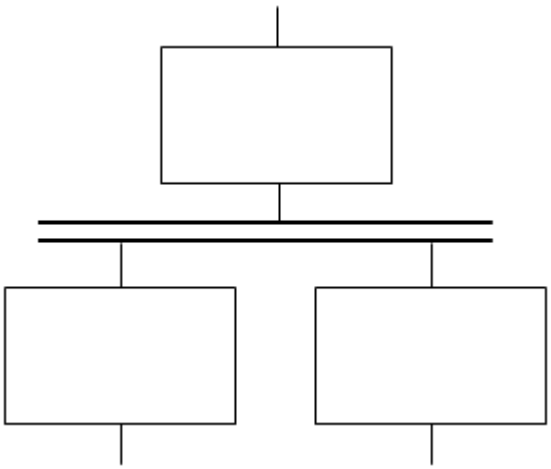
Výpustka se používá tehdy, chceme-li označit místo, kde dochází k vypuštění symbolu nebo skupiny symbolů, přičemž ani druh ani jejich počet nemusí být definován ([1], s. 12).



OBR. 16 SYMBOL VÝPUSTKA

2.3.1.11 PARALELNÍ REŽIM

Symbol paralelního režimu představuje „synchronizaci dvou nebo více paralelních operací“([1], s. 7). V programování lze tento symbol chápat jako tzv. vlákna[7] (thready).



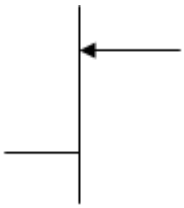
OBR. 17 SYMBOL PARALELNÍ REŽIM

[7] Vlákno je samostatně plánovatelný tok řízení programu [20].

2.3.1.12 SPOJNICE

Spojnice je symbol ve tvaru svislé nebo vodorovné čáry, představující tok dat nebo řízení ([4], s. 22). Zároveň slouží pro spojování jednotlivých symbolů vývojového diagramu tak, že je zřejmé jejich následné pořadí aktivace.

V případě, že směr spojnice nedodržuje standardní směr toku informací (*viz kapitola 2.3.2*), je nutné na jejím konci pro zvýšení názornosti použít plnou nebo otevřenou šipku ([1], s. 15).



OBR. 18 SYMBOL SPOJNICE

2.3.2 KONVENCE VÝVOJOVÝCH DIAGRAMŮ

Mezinárodní norma pro zápis vývojových diagramů ustanovuje i jisté konvence, které je doporučeno dodržovat ([1], s. 13, 15):

- Standardní směr toku informací by měl být zobrazen zleva doprava a shora dolů.
- Doporučuje se, aby spojnice vstupovaly do symbolu zleva nebo shora a vystupovaly vpravo nebo dole. Spojnice se doporučují směrovat ke středu symbolu.
- Znázorňujeme-li jakýkoliv symbol diagramu, úhly a ostatní veličiny ovlivňující jeho relativní tvar nesmí být změněny a jeho velikost by vzhledem k ostatním symbolům měla být jednotná.
- Doporučuje se, aby vzdálenosti mezi symboly byly stejné a spojnice dosahovaly přiměřené délky s minimálním množstvím dlouhých čar.
- Do symbolu je doporučeno vkládat minimální množství textu, které je nutné pro porozumění jeho funkce.

3 ANALÝZA

Analýza tvoří stěžejní krok a základ pro další fáze vývoje projektu [21]. Je proto velice důležité analýzu provést důkladně a na stanovené téma tak nashromáždit co největší množství informací, které bude tvořit základ pro následující kroky vývoje aplikace.

Na počátku analýzy bylo stanoveno těchto 5 stěžejních etap prvotního sběru informací, jimiž byla analýza řízena a na základě kterých pak bylo možné postupovat dále:

1. Zmapování a studium související literatury
2. Určení cílové skupiny uživatelů
3. Zmapování současného stavu problematiky uvnitř cílové skupiny uživatelů
4. Zájem o výstup práce
5. Analýza již existujících projektů

Kromě získávání a studia literatury týkající se vývojových diagramů, algoritmizace a programování, bylo nutné stanovení cílové skupiny uživatelů, pro kterou bude aplikace určena. Jak již vyplývá z názvu této práce, aplikace má být primárně adresována výukovému sektoru, jmenovitě výuce algoritmizace respektive programování. Jak uvádí Taufer (Taufer, 2009: 18), vývojové diagramy jsou „vhodné zejména u začínajících programátorů, protože dovolují názorným způsobem formulovat potup řešení daného úkolu s vyznačením všech jeho možných alternativ“. Jako cílová skupina uživatelů byla tedy zvolena oblast středních škol disponujících obory ICT, neboť právě zde se v současnosti s výukou úvodu do programování a algoritmizace lze setkat nejčastěji.

Po stanovení cílové skupiny uživatelů byla analýza soustředěna k získání informací o tom, zda jsou vývojové diagramy ve výuce v současnosti používány, jakým softwarem je prováděna realizace jejich tvorby a zda by školy měly o nové řešení zájem (*viz kapitola 3.1*). Tímto krokem tak byl splněn další stanovený milník analýzy, který spočíval ve zmapování zájmu a současného stavu této problematiky uvnitř cílové skupiny uživatelů.

Jako další krok byla poté provedena analýza již existujících projektů, které umožňují sestavení a průchod vývojovými diagramy (*viz kapitola 3.2*).

3.1 DOTAZNÍKOVÝ PRŮZKUM

Na základě stanovení cílové skupiny uživatelů jako středních škol, bylo dále nutné zajistit informace o jejich postoji k vývojovým diagramům. Byl proto navržen dotazníkový průzkum, který měl za úkol shromáždit odpovědi na tyto kladené otázky:

- Jsou při výuce algoritmizace a programování vývojové diagramy používány?
- Je případný vývojový diagram vytvářen na počítači pomocí vhodného softwaru?
- Jaký software je na středních školách pro účel vytváření vývojových diagramů využíván?
- Jaké programovací jazyky jsou na středních školách nejčastěji vyučovány?
- Měly by školy zájem o aplikaci, která by byla přímo určena pro sestavení a průchod algoritmu vývojového diagramu?

Pro zhotovení dotazníku byla zvolena služba *Google Docs*[8] a její dotazníkový nástroj, jež poskytuje bezplatné řešení pro vytvoření, zveřejnění a vyhodnocení dotazníků. Výsledný dotazník (*viz Příloha B*) byl následně distribuován pomocí elektronické pošty těm středním školám, které disponují oborem ICT. Školní e-mailové schránky byly získány pomocí serveru *firmy.cz*[9].

3.1.1 VÝSLEDKY PRŮZKUMU

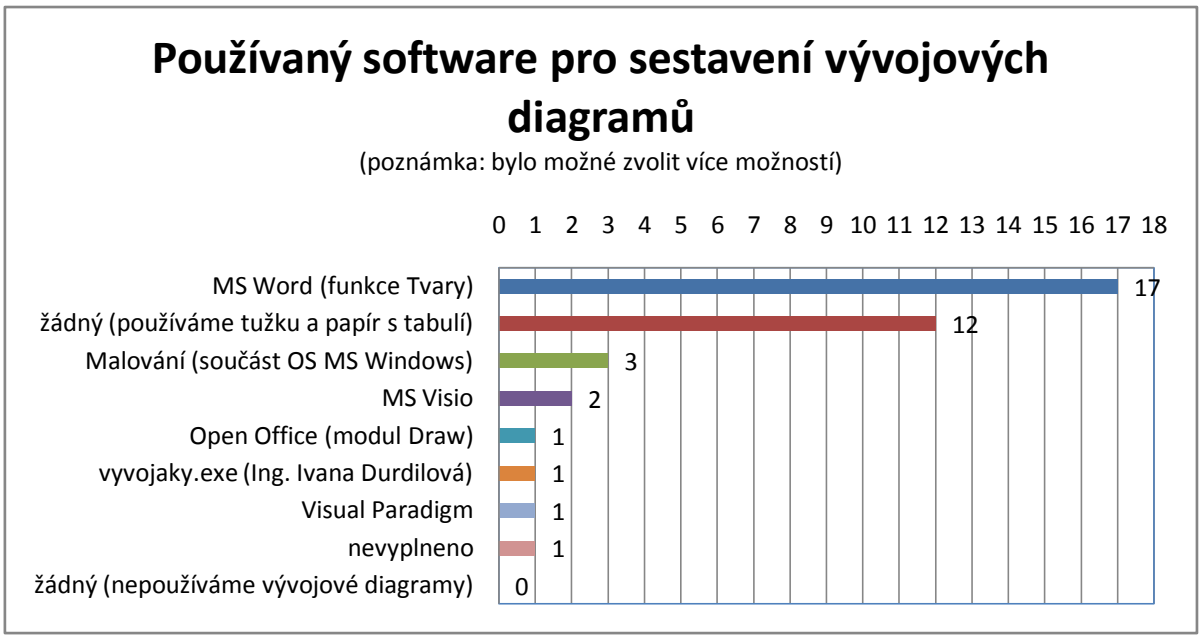
Dotazník byl rozeslán celkem 57 školám a konečná návratnost činila uspokojivých 26 vyplněných dotazníků. Všechny otázky které byly v dotazníku zahrnuty, byly nepovinné a na každou z nich byla poskytnuta možnost alternativní odpovědi. Výsledky průzkumu budou interpretovány na následujících řádcích.

Graf na Obr. 19 mapuje, jaký software je na středních školách používán pro sestavení vývojových diagramů. Z tohoto grafu lze vyčíst, že nejpoužívanější způsob jejich sestavení spočívá ve využití textového procesoru

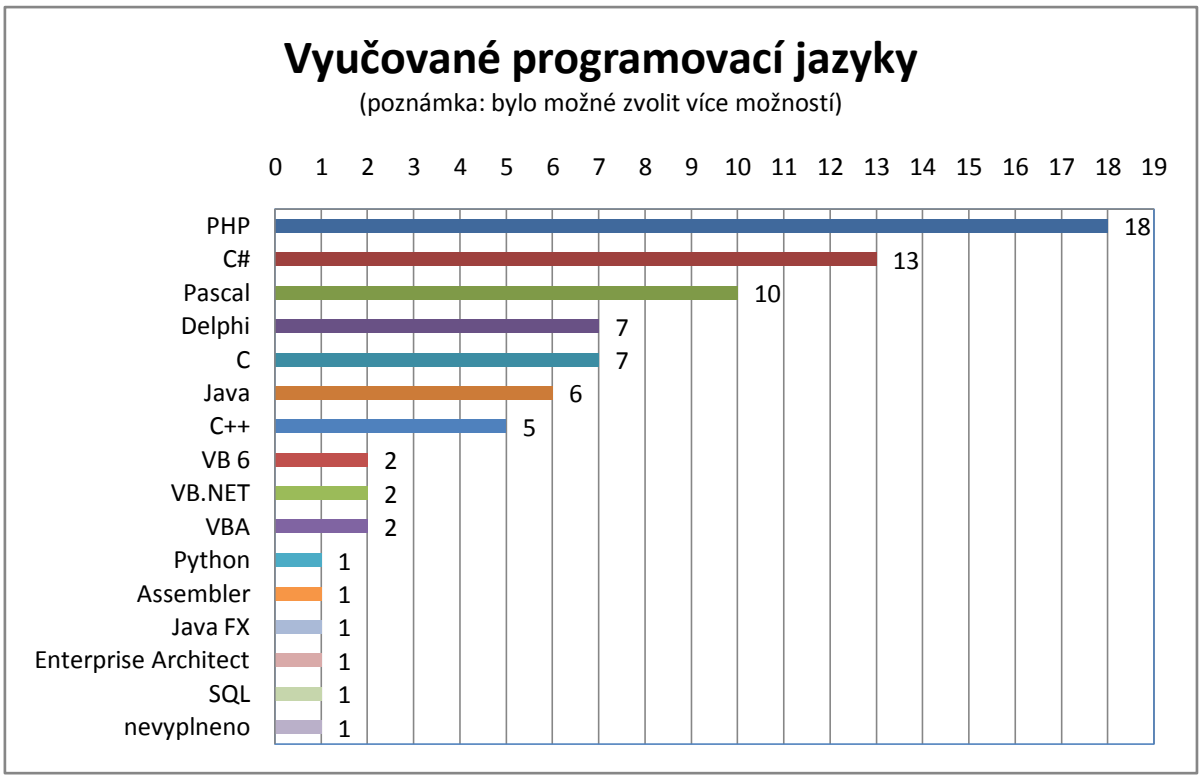
[8] Google Docs je služba webového kancelářského balíku, poskytovaná společností Google.

[9] Server *firmy.cz* v současnosti poskytuje přehledné informace o většině českých škol a jimi nabízených oborů.

Microsoft Word a jeho nástroje Tvary, následovaný použitím tradiční tužky s papírem a školní tabule. Zároveň tento graf dokazuje, že vývojové diagramy jsou školami při výuce plně uplatňovány.



OBR. 19 POUŽÍVANÝ SOFTWARE PRO SESTAVENÍ VÝVOJOVÝCH DIAGRAMŮ



OBR. 20 VYUČOVANÉ PROGRAMOVACÍ JAZYKY

Na Obr. 20 je patrné zastoupení vyučovaných programovacích jazyků na oslovených středních školách. Předmětem zájmu byly zejména programovací jazyky Pascal a Java, neboť právě ty byly předem určeny pro možnou podporu funkce importu a exportu ve výsledné aplikaci (*viz kapitola 1.2 Cíle práce*).

Poslední graf na Obr. 21 ilustruje výši zájmu o novou aplikaci, která by umožňovala sestavení a simulaci průchodu algoritmem vývojového diagramu.

OBR. 21 ZÁJEM O APLIKACI

3.1.2 SHRNUTÍ PRŮZKUMU

Průzkum poskytl jasné odpovědi na kladené otázky a byl tak pro další vývoj práce shledán velmi užitečným.

Ukázalo se, že všechny oslovené školy vývojové diagramy ve výuce používají. Překvapením se pak stala skutečnost, že valná většina odpovědí ohledně používaného softwaru pro sestavování diagramů neoznačovala žádný software, který by byl pro tento účel přímo specializovaný. Nejpoužívanějším řešením pro tvorbu vývojových diagramů byly označeny aplikace Microsoft Word a Malování, které tuto činnost činí velice obtížnou. Software cílený na práci s vývojovými diagramy se tedy jeví jako nedostatkový, čemuž odpovídá i zájem škol, který je značný.

Jelikož dotazník zahrnoval i nepovinné textové pole pro zadání sídla školy, bylo rovněž možné zhotovit orientační mapu (Obr. 22), na níž lze

pozorovat zeměpisné polohy škol v rámci České republiky, které řečený zájem projevily.

OBR. 22 ZEMĚPISNÁ MAPA ZÁJMU

Červený praporek v horní části mapy označuje Střední průmyslovou školu Trutnov, odkud mne na základě přijetí dotazníku kontaktoval tamní učitel informatiky pan Mgr. Aleš Janata. Po zodpovězení několika otázek, které mi byly položeny, byla panu učiteli učiněna nabídka ohledně spolupráce na tomto projektu, s níž následně souhlasil. Další postup práce tak mohl pokračovat na základě zpětné vazby pedagoga, který vývojové diagramy ve výuce aktivně používá. Druhý z červených praporků v dolní části mapy pak značí sídlo Střední odborné školy elektrotechnické Hluboká nad Vltavou, která se podílela na konečném otestování aplikace v praxi (*viz kapitola 6).*

3.2 ANALÝZA EXISTUJÍCÍCH PROJEKTŮ

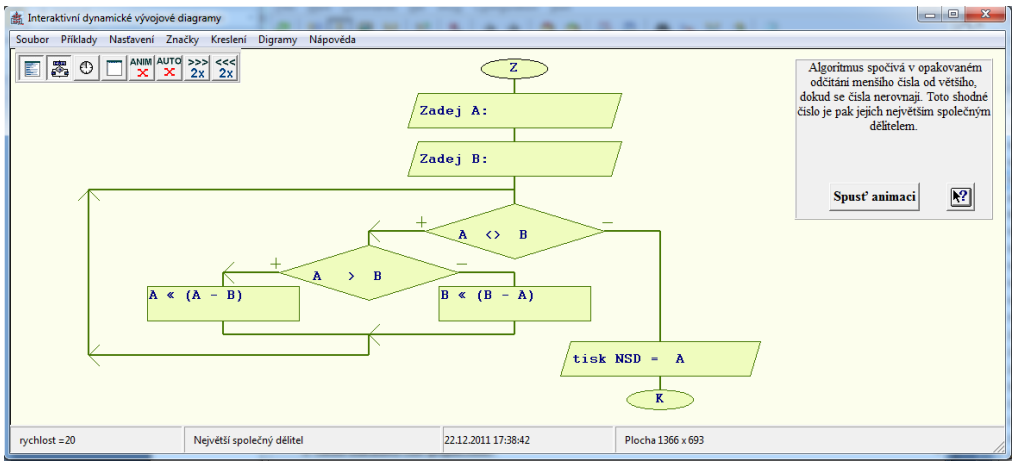
Posledním ze stanovených kroků analýzy byla analýza již existujících projektů, které se sestavováním a simulací průchodu algoritmu vývojových diagramů přímo zabývají.

Jako prvotní databáze relevantních projektů byly použity odpovědi z předešlého dotazníku, které označovaly software, jenž školy pro sestavování vývojových diagramů používají. Po jejich průzkumu se však ukázalo, že jediným softwarem, který umožňuje obě z řečených funkcí je v tomto výčtu

projekt označený jako „vyvojaky.exe“, vytvořený Ing. Ivanou Durdilovou. Následné pátrání internetem přineslo ještě jeden nález podobného projektu, jehož autorem je Ivo Snoza. Následující podkapitoly budou věnovány jejich analýze.

3.2.1 Animované vývojové diagramy - Ing. Ivana Durdilová

Tento projekt z roku 2006 [12] je používán právě na jedné z dotazovaných středních škol. Na jeho domovské internetové adrese[10] o něm bohužel příliš informací dohledat nelze. Jako jediný zdroj informací zde slouží nápověda aplikace, která je však zhotovena v zastaralém formátu a pro její chod je v novějších Windows (testováno s Windows 7) nutné doinstalovat zásuvný modul z oficiálních stránek společnosti Microsoft[11].



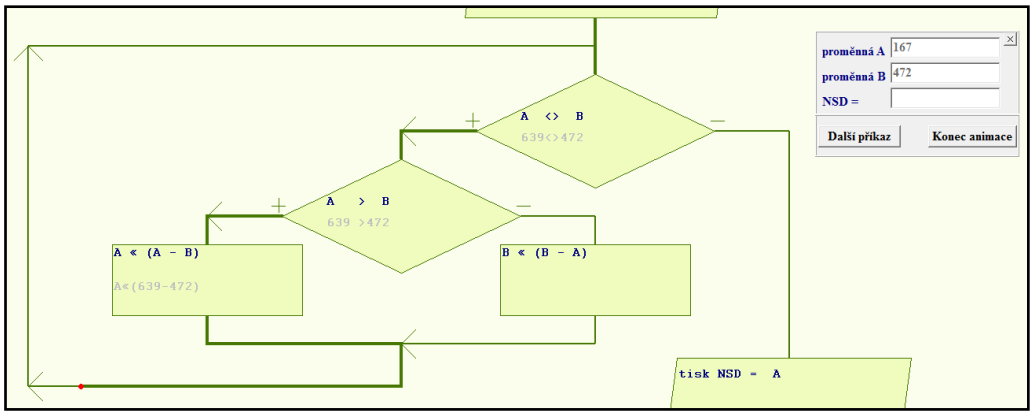
Obr. 23 Animované vývojové diagramy

Celkový vzhled aplikace nepůsobí příliš intuitivně a uživatel se v něm lehce ztratí. Při testování aplikace docházelo dokonce i na takové situace, kdy bylo jednodušší aplikaci restartovat, než hledat způsob návratu na předchozí obrazovku.

Simulace průchodu vývojovým diagramem je realizována krokováním nebo pomocí animace „putující“ kuličky, která symbolizuje tok programu a dává tak uživateli najevo, kudy se algoritmus jeho diagramu ubírá (Obr. 24). Trajektorie této kuličky je pak ztučňována, což naznačuje, které spojnice již byly aktivovány. Pohyb kuličky je však realizován konstantní rychlostí a při

[10] *http://lab.uzlabina.cz/~projekty/index.htm*
[11] Aplikace je určena pouze pro operační systém Microsoft Windows

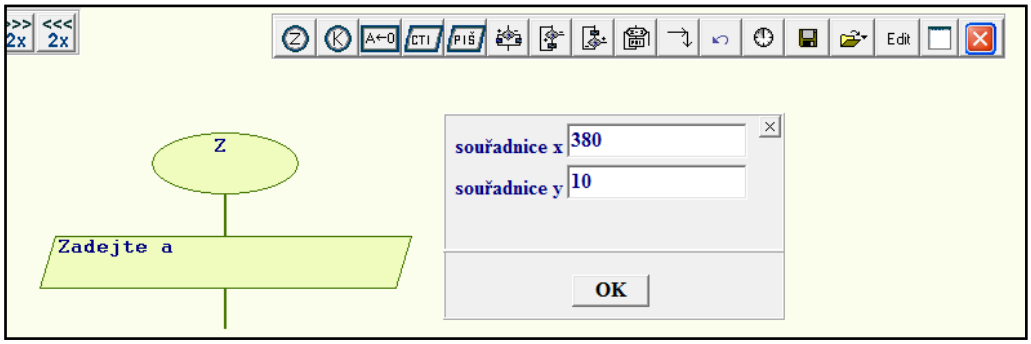
delší spojnici se tak animace stává velice zdlouhavou (nepomůže ani tlačítko pro její urychlení). Dále bylo zjištěno, že při probíhající animaci dosahuje zatížení procesoru 100%, což vypovídá o nesprávné implementaci animačního jádra aplikace.



OBR. 24 ANIMACE SIMULACE PRŮCHODU VÝVOJOVÝM DIAGRAMEM

Potěší naopak zobrazení mezivýpočtů funkcí symbolů, které pak při výuce představují hodnotný zdroj informací o chování algoritmu.

Zásadním nedostatkem této aplikace je systém pro vytváření vlastních vývojových diagramů. Uživatel je zde nucen jednotlivé prvky diagramu umisťovat pomocí manuálního určení souřadnic na plátně, symboly není možné přetahovat pomocí kurzoru myši. Pracovní plátno je navíc omezeno rozlišením uživatelovy obrazovky a celková plocha vývojového diagramu je tak tímto rozměrem limitována.



OBR. 25 SESTAVOVÁNÍ VÝVOJOVÉHO DIAGRAMU

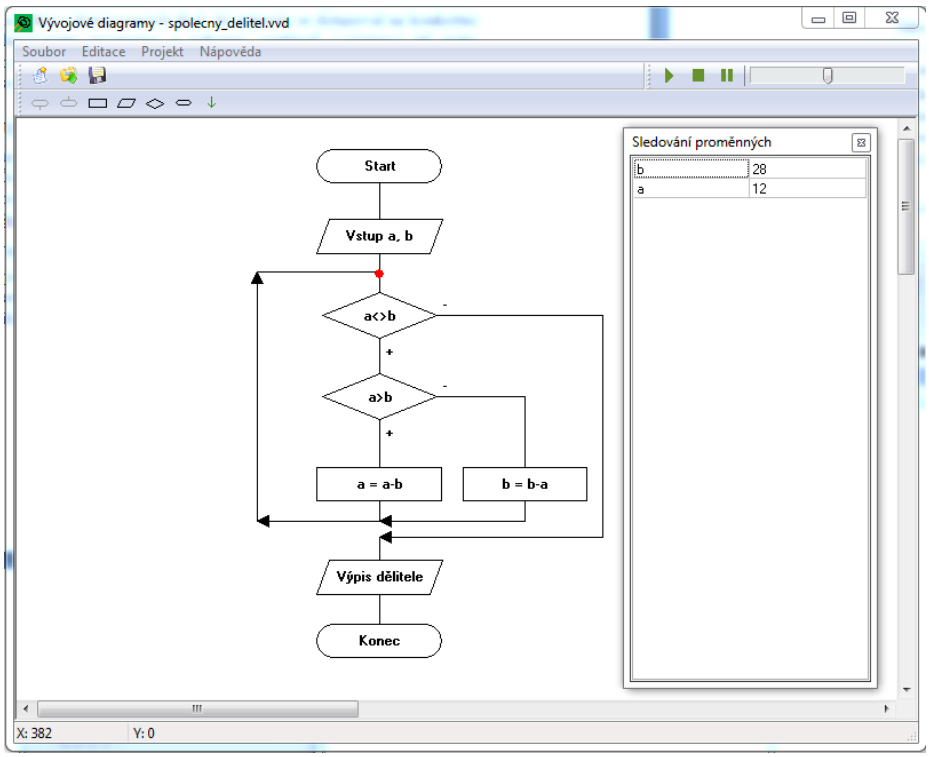
Vytváření vývojového diagramu představuje náročnou operaci a aplikace je tak odsouzena spíše jen k promítání předpřipravených algoritmů. Sestavování vývojového diagramu studenty během vyučovací hodiny je

z tohoto důvodu těžko realizovatelné. Program navíc nepodporuje práci s poli[12] a nabízí jen základní sadu symbolů vývojového diagramu programu.

3.2.2 Interaktivní dynamické vývojové diagramy – Ivo Snoza

Tento projekt není používaný na žádné z dotazovaných škol. Jedná se o jednoduchou aplikaci pro vytváření vývojových diagramů a pro jejich následnou simulaci průchodu. Projekt byl vytvořen jako bakalářská práce v roce 2009 [11].

Jako manuál zde opět slouží pouze nápověda aplikace, tak jako u předešlého projektu. Ta je přitom velice střídmá, nepopisuje proces vytváření diagramů, ani spuštění animace. Grafické rozhraní aplikace působí přívětivěji než u předešlého projektu, současně se intuitivněji ovládá. K dispozici je zde ale opět jen základní sada symbolů vývojového diagramu programu a aplikace postrádá možnost práce s poli[12].



Obr. 26 Interaktivní dynamické vývojové diagramy

[12] Pole je v programování kolekce proměnných stejného typu, které mohou být označovány společným identifikátorem [22].

Simulace průchodu vývojovým diagramem je realizována pouze animací průchozí kuličky, možnost jejího krokování implementována není. Při animaci bohužel nejsou zobrazovány jakékoliv mezivýpočty funkcí symbolů a uživatel je tak odkázán jen na okno s výpisem proměnných a jejich aktuálních hodnot. Spojnice nejsou po jejich aktivaci zvýrazňovány a jediným vodítkem postupu při průchodu vývojového diagramu se tak stává aktuální pozice průchodové kuličky (Obr. 26).

Výrazným krokem vpřed oproti předešlému projektu je zde samotné vytváření vývojových diagramů. Do procesu umisťování symbolů byla totiž zapojena interakce myší a je tak možné jednotlivé značky přesouvat pomocí funkce drag&drop[13]. Při případných úpravách vytvářeného diagramu se však uživatel setká s problémem, kdy není možné označit a přesunout více symbolů naráz. Když je pak žádáno mezi umístěné symboly vložit symbol další, je uživatel pro získání prostoru mezi nimi nucen přesunout každou tuto značku jednotlivě. Při manipulaci se symboly není aplikováno zarovnávání k mřížce a pracovní plátno je omezeno konstantní velikostí.

Aplikace celkově působí velmi „základním“ dojmem, kdy pro splnění svého účelu nabízí jen tu nejnezbytnější funkcionalitu. Způsob přidávání proměnných není nejvhodnější, je nutné je odděleně deklarovat a uživatel je pro tento záměr nucen otevřít množství dialogových oken. Chybí také export vývojového diagramu do obrázku.

3.2.3 Další projekty

Tvorbou vývojových diagramů se zabývá řada dalších projektů, již však nezahrnují možnost vizualizace jejich algoritmického průchodu.

Mezi komerčními projekty je to například Microsoft Visio[24] a SmartDraw[25], z bezplatných aplikací je vhodné zmínit yEd[26] a online editory lucidchart.com[27] a gliffy.com[28].

3.3 Shrnutí analýzy

Analýza pomocí dotazníkového průzkumu prokázala, že vývojové diagramy jsou v rámci výuky algoritmizace a programování používány, a to kompletně všemi oslovenými školami. Průzkum zároveň odhalil, že software, který školy

[13] Drag&drop je akce, která dovoluje uchopit grafický objekt kurzorem myši a přemístit jej na požadovanou destinaci [23].

využívají k jejich vytváření, není pro tento účel příliš vhodný, zejména z důvodu obtížných úprav uspořádání a vzájemného propojení symbolů. Nepochybně i z tohoto důvodu byl pak odhalen velký zájem o výstup této práce, který by měl uvedené komplikace redukovat a zároveň nabídnout možnost interaktivní simulace průchodu vytvořeného diagramu. Dále pak tato práce získala cenného spojence, pana učitele Mgr. Aleše Janatu, který souhlasil s poskytováním zpětné vazby při testování aplikace.

Analýza existujících projektů poukázala na jejich nedostatky při sestavování a úpravách vývojových diagramů. Tyto nedostatky pak nedovolují žákovi (či učiteli) rychlé sestavení či úpravu diagramu přímo během výuky a aplikace se tak hodí spíše jen k prezentaci předpřipravených příkladů. Aplikace Ing. Ivany Durdilové naopak přináší zajímavý přístup při simulaci průchodu vývojovým diagramem, kdy je uživateli přehledně zobrazen proces výpočtů funkcí symbolů a trajektorie průchodové kuličky.

3.3.1 Volba podporovaného programovacího jazyka

Dotazníkový průzkum mapoval mimo jiné i zastoupení všech programovacích jazyků, které oslovené školy vyučují. Jelikož cílová aplikace má umožňovat export a import vývojového diagramu do/ze zdrojového kódu programovacího jazyka Pascal nebo Java, bylo nutné rozhodnout, který z nich (nebo zda oba) bude pro tento účel podporován. Dle výsledků průzkumu je to právě Pascal, který je vyučován častěji (celkem na 10 školách). Java však nezůstala o mnoho pozadu (je vyučována na 6 oslovených školách) a bylo tedy nutné zvážit podporu obou těchto jazyků.

Java

Po porovnání obou jazyků vůči vývojovým diagramům bylo zjištěno, že Java nepodporuje příkaz skoku „goto“ a symbol Spojka, který tuto funkcionalitu představuje, by tak nemohl být ve vývojovém diagramu v případě exportu do zdrojového kódu zahrnut. Vývojové diagramy navíc nejsou koncipovány pro objektový přístup a Java, jakožto objektově orientovaný jazyk se tedy pro účel importu/exportu jeví nevhodně.

Pascal

Pascal však na druhou stranu nedovoluje pro cyklus s pevným počtem opakování stanovit jinou inkrementační konstantu než 1 nebo -1, zatímco ve

vývojových diagramech respektive ve výuce algoritmizace by mělo být umožněno tuto konstantu stanovit i jinými hodnotami.

Závěr

Po zvážení podpory obou programovacích jazyků byl nakonec pro funkcionalitu exportu a importu do/ze zdrojového kódu zvolen pouze programovací jazyk Pascal. Programovací jazyk Java byl pro tento účel shledán jako nevhodný, a to pro jeho objektový přístup, jímž se vývojové diagramy nezabývají (vývojové diagramy jsou navrženy pro strukturované programování [4]).

4 NÁVRH

Na základě zjištění získaných z provedené analýzy, bylo nyní nutné vytvořit takový návrh aplikace, který by nejen splňoval stanovené cíle práce, ale vyhnul se i nedostatkům, které byly vyčteny existujícím projektům. Bylo tedy nezbytné navrhnout nový, efektivní způsob sestavování vývojového diagramu, určit metodu přehledné vizualizace jeho průchodu, vytvořit objektový návrh a harmonogram vývoje aplikace a v neposlední řadě zvolit vhodný jazyk pro její naprogramování.

Volba programovacího jazyka byla po předchozích zkušenostech s jazykem Java jednoduchá. Po zvážení celkové rozsáhlosti budoucí aplikace bylo totiž usouzeno, že osvojit si jakýkoliv nový jazyk by bylo z časového hlediska pro tuto práci nereálné. Java je jazyk moderní a robustní. Nativně podporuje jak pokročilou práci s grafickým kontextem, tak technologie XML[14] a JavaScript[15], které budou pro následný vývoj aplikace klíčové. Používání Javy pro běžný vývoj (i komerční) je zdarma, výstupní aplikace jsou navíc mezi-platformě přenositelné[16] a mohou být distribuovány jako applety[17].

V následujících kapitolách budou pro podporu popisovaných idejí použity snímky z hotové aplikace[18].

4.1 VYTVÁŘENÍ VÝVOJOVÉHO DIAGRAMU

Stávající projekty byly v oblasti rychlého návrhu vývojových diagramů shledány jako nepříliš efektivní a tím utrpěl i jejich potencionál pro vytváření a úpravu algoritmů přímo během vyučovací hodiny. Následující podkapitoly popíší takový návrh metody vytváření diagramů, který by tímto nedostatkem neměl dále trpět.

[14] XML je rozšiřitelný značkovací jazyk, používaný zejména pro výměnu dat mezi aplikacemi a pro publikování dokumentů [33].

[15] JavaScript je skriptovací objektový a interpretovaný programovací jazyk [34].

[16] Přenositelné mezi všemi obvyklými platformami (UNIX, Windows, MAC OS…).

[17] Applet je program napsaný v Javě, spustitelný ve webovém prohlížeči [35].

[18] Původní nákresy byly prováděny na papír a postrádají tak potřebný prezentační potenciál.

4.1.1 LAYOUT

Stávajícím projektům byl vyčten náročný způsob sestavování diagramů a jejich komplikované úpravy. Jednotlivé symboly diagramu by rozhodně neměly být umisťovány definováním jejich souřadnic na zobrazovacím plátně a jejich následné úpravy, jako např. přemisťování, mazání, vsunutí dalšího symbolu apod., by neměly být spojeny s potřebou přestavění celé hierarchie diagramu.

Ideálním řešením by pak bylo navrhnout takový systém sestavování diagramů, který by symboly umisťoval na plátno automaticky. Byly-li by symboly rovněž automaticky propojovány spojnicemi, uživatel by byl zbaven veškeré starosti s formátováním diagramu a aplikace by tak byla použitelná i pro dynamické úpravy a vytváření algoritmů přímo během vyučovací hodiny. Nemusel-li by pak žák ztrácet čas s pozicováním jednotlivých symbolů na aplikačním plátně, byla by mu tím rovněž poskytnuta možnost plné koncentrace na řešení stanoveného problému.

Na základě této vize byl pak vytvořen návrh Normového layoutu, který by tuto funkcionalitu opravdu zprostředkovával. Tento layout, jak je patrné z jeho názvu, by pak zároveň symboly měl umisťovat tak, jak je to definováno normou ČSN ISO 5807.

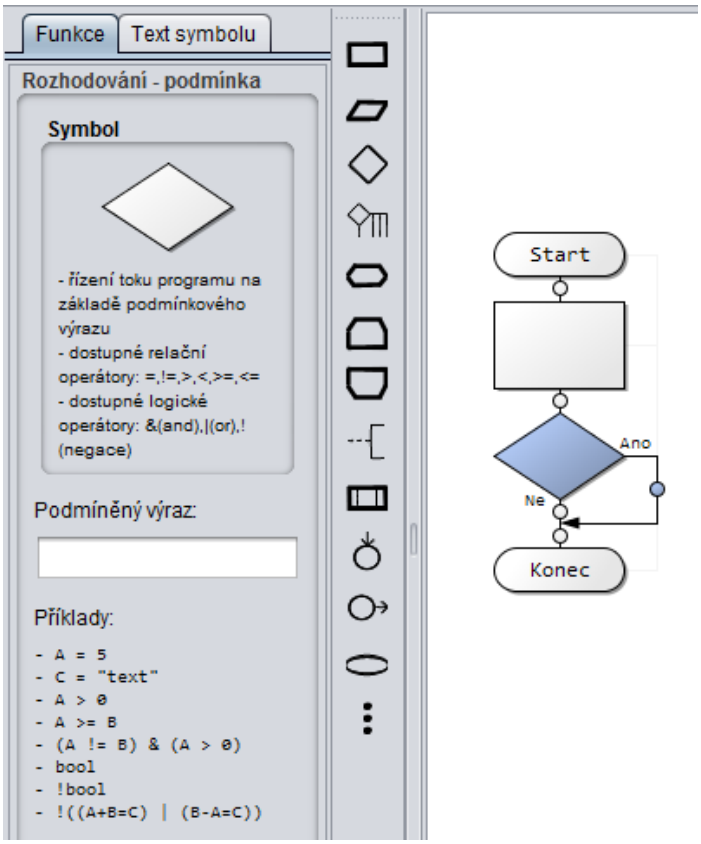
4.1.2 VKLÁDÁNÍ SYMBOLŮ

Během návrhu layoutu bylo nutné rovněž zkonstruovat takový systém pro přidávání symbolů, který by zachoval jeho přednosti a zároveň poskytl co nejvyšší svižnost této operace.

Byl proto navržen koncept přidávání symbolů pomocí tzv. „vkládacích bodů“. Tyto vkládací body pak budou vykresleny mezi symboly vývojového diagramu na takových místech, kam lze potencionálně vložit symbol nový (Obr. 27). Uživatel pak jen označí požadovaný vkládací bod, stiskne tlačítko pro přidání symbolu a ten se na vyznačené pozici automaticky začlení do vývojového diagramu.

Pro ještě vyšší efektivitu sestavování diagramu byl navržen inteligentní systém tzv. „párového označování“, který zajistí, že v jednom okamžiku je vždy označen pár symbol a jeho korespondující vkládací bod (Obr. 27, označení značí modrá barva). Tato funkcionalita zajistí svižnost při vkládání a editaci symbolů, neboť umožní tyto dvě činnosti provádět naráz, bez nutnosti dalšího označování. Po vložení symbolu je tak možné ihned editovat např. jeho

text či funkci, přičemž je aplikace stále připravena na přidání symbolu následujícího.



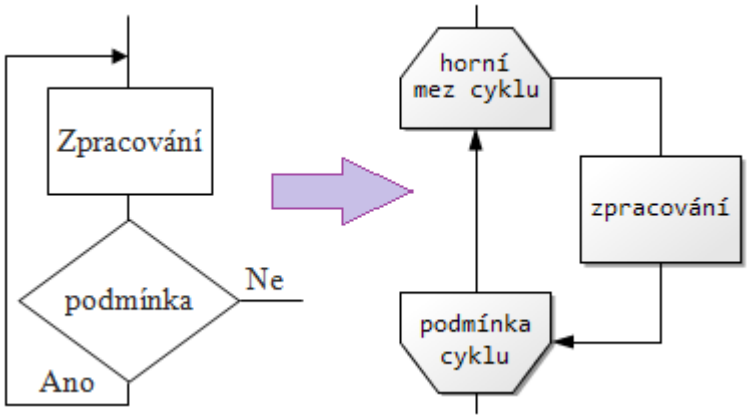
OBR. 27 SYSTÉM VKLÁDÁNÍ A EDITACE SYMBOLŮ

4.1.3 VÝBĚR SADY SYMBOLŮ VÝVOJOVÉHO DIAGRAMU

Ačkoliv si lze při vytváření jakéhokoliv algoritmu vystačit pouze se základními symboly diagramu (zpracování, vstup/výstup a podmínka), bylo po následující úvaze rozhodnuto, že výsledná aplikace bude podporovat kompletní sadu symbolů vývojového diagramu programu (vyjma značky paralelního zpracování, která se v základní algoritmizaci nepoužívá).

Byla-li by aplikace použita pro prvotní výuku programování, bylo by teoreticky pro tento účel výhodnější používat plnou škálu nabízených symbolů. Je-li totiž při řešení úlohy využit vývojový diagram s korektním použitím všech jeho příslušných značek, bude pro studenta přechod na programovací jazyk usnadněn tím, že každá z použitých značek má v programování taktéž svou jedinečnou zástupnou formu syntaxe.

Je pak škoda zobrazovat například cyklus s podmínkou dole pomocí značky rozhodování, když pro tento konstrukt ve specifikaci existuje konkrétní symbol (Obr. 28). Žák by pak mohl být při prvním styku s programovacím jazykem zmaten, neboť by se mohl snažit tento cyklus zkonstruovat pomocí podmínky, což obvykle v programování není možné. Využití plného rozsahu podporovaných symbolů pak proto bude z teoretického hlediska jen přínosem.



OBR. 28 POUŽITÍ MEZNÍ ZNAČKY CYKLU S PODMÍNKOU DOLE

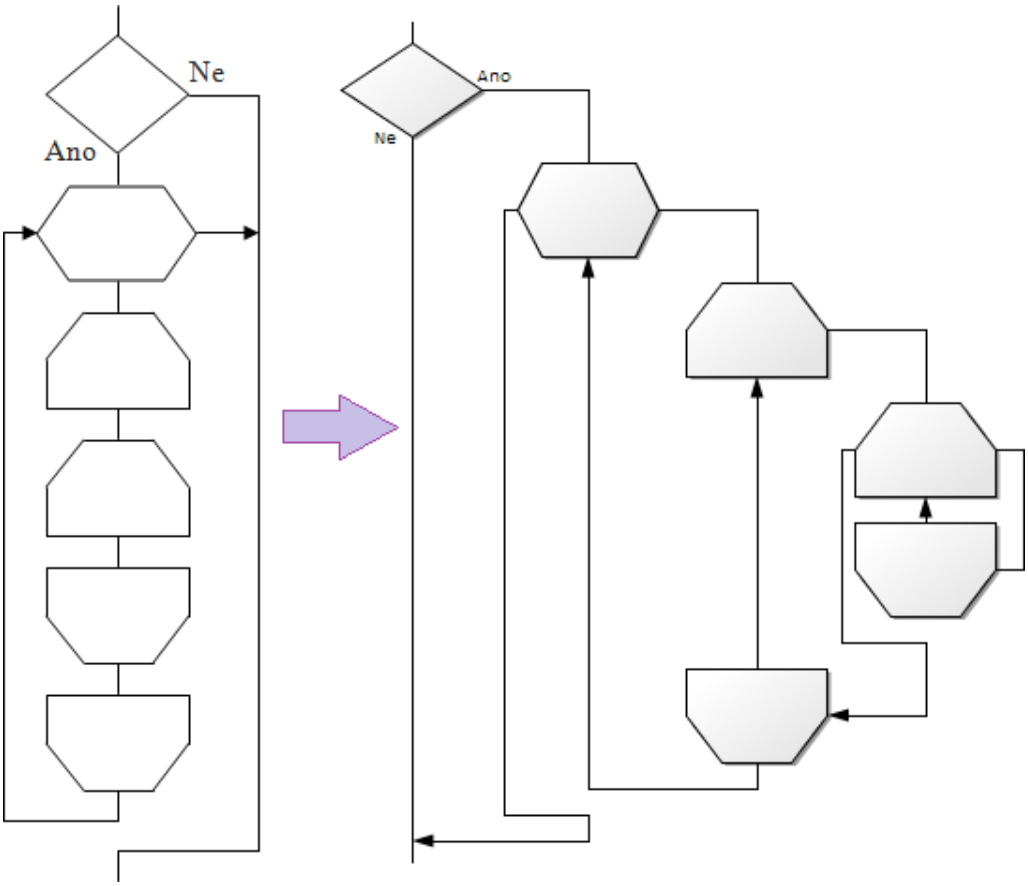
4.1.4 KONCEPCE USPOŘÁDÁNÍ SYMBOLŮ

Jelikož budou jednotlivé symboly umisťovány a propojovány spojnicemi automaticky pomocí layoutu, bylo nezbytné vytvořit pečlivý návrh koncepce jejich uspořádání.

Dle konvence normy (*viz kapitola 2.3.2*) by měl standardní tok informací vývojového diagramu být zobrazován shora dolů a zleva doprava. V různých publikacích o vývojových diagramech lze často pozorovat takové rozmístění symbolů, které vede k jejich kladení převážně směrem dolů (např. Taufer, Pšenčíková). Aby pak layout nezobrazoval symboly diagramu v kontinuální řadě orientované pouze tímto směrem, byla navržena taková koncepce jejich uspořádání, která jakoukoliv větev, která symbolizuje rozvětvení nebo expanzi[19], bude klást směrem doprava a až poté směrem dolů (Obr. 29). Použitím této koncepce pak bude zároveň možné pozorovat strukturu a zanořování vytvářeného algoritmu.

[19] Expanzní větví symbolu je myšleno tělo cyklu a tělo rozhodování (True větev).

Na Obr. 29 je patrný rovněž způsob, kterým byl implementován symbol mezní značky cyklu (dva páry symbolů vpravo). Jeho propojení spojnicemi bylo navrženo tak, aby bylo ihned zřejmé, zda se jedná o cyklus s podmínkou na začátku nebo s podmínkou na konci. Oba cykly totiž fungují na principu vyhodnocení svého podmínkového výrazu, na jehož základě algoritmus pokračuje prvním příkazem uvnitř těla cyklu nebo příkazem následujícím (vyznačeno spojnicemi). Právě umístění podmínkového výrazu nahoře nebo dole je pak patrné z jeho propojení i bez přítomnosti textu.



OBR. 29 KONCEPCE EXPANZNÍHO USPOŘÁDÁNÍ SYMBOLŮ

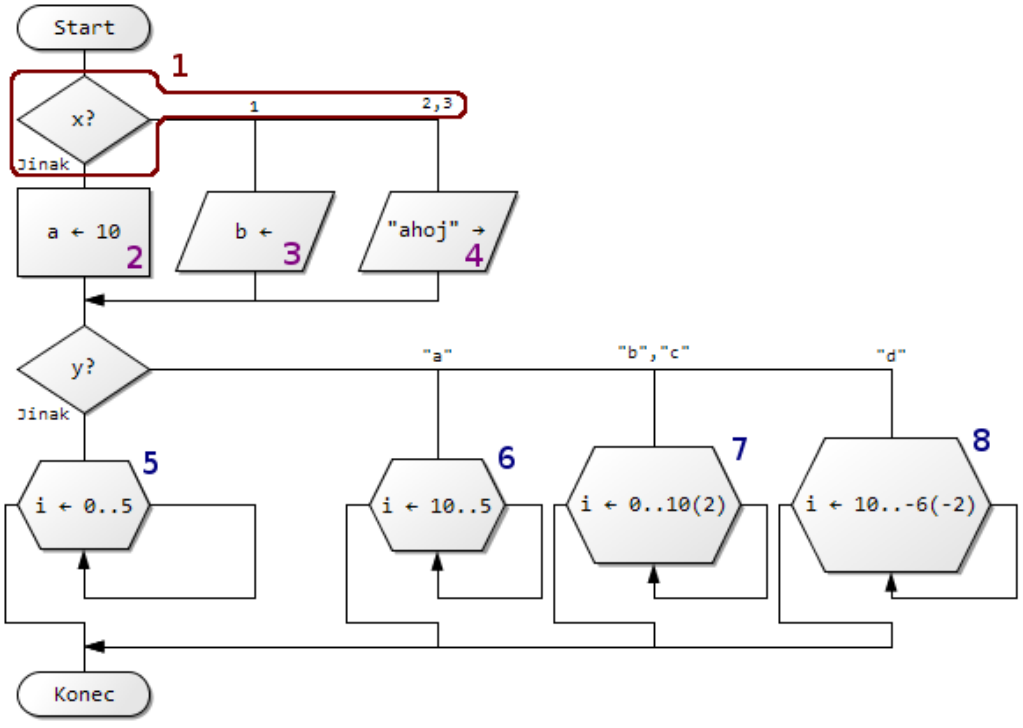
4.1.5 AUTOMATICKÉ GENEROVÁNÍ TEXTU SYMBOLU

Další položka návrhu aplikace spočívala v nalezení způsobu, jak z uživatelem definované funkce symbolu automaticky vygenerovat textovou hodnotu, která by tuto funkci uvnitř jeho grafické značky vizuálně reprezentovala. Uživatel tak bude zproštěn starosti ohledně vyplnění textové hodnoty symbolu a zároveň tak bude vytvořen koncept jednotné příkazové syntaxe.

Pro návrh této funkcionality bylo důležité dbát na to, že by textová hodnota symbolu měla být co nejkratší a co nejvýstižnější. Již na počátku práce bylo panem Mgr. Alešem Janatou zmíněno, že řetězcový znak pro přiřazení hodnoty do proměnné by se neměl stávat z rovnítka („=“), jelikož pak žákům dělá problém této syntaxi porozumět. Nemají-li totiž zkušenosti s programovacím jazykem, znají znak rovnítka pouze z hodin matematiky a zápis příkazu „a = a+1“ se pak v jejich očích jeví jako nesmyslný. Pro reprezentaci příkazu přiřazení hodnoty do proměnné byl tedy zvolen znak šipky („←“). Dále pak budou nahrazovány tyto programové konstrukce:

- Nerovná se „!=“ za „≠“
- Negace „!“ za „¬“
- Větší, menší nebo rovno „>=; <=“ za „≥; ≤“

Na Obr. 30 je zobrazen koncept, který byl pro účel automatického generování textu na základě vyplněné funkce symbolu vytvořen. Níže pak následují podrobnější popisné informace.



OBR. 30 KONCEPT AUTOMATICKY GENEROVANÉHO TEXTU

1. Vícenásobné rozhodování (příkaz switch) – na základě hodnoty proměnné „x“ je tok programu přesunut do první, druhé nebo spodní větve symbolu.
2. Zpracování – do proměnné „a“ je přiřazena hodnota „10“
3. Vstup – do proměnné „b“ je přiřazena uživatelem zadaná hodnota
4. Výstup – je zobrazen dialog s textovou zprávou „ahoj“
5. Cyklus s pevným počtem opakování (cyklus for) – do proměnné cyklu „i“ jsou postupně přiřazeny hodnoty 0,1,2,3,4 a 5.
6. Cyklus s pevným počtem opakování (cyklus for) – do proměnné cyklu „i“ jsou postupně přiřazeny hodnoty 10,9,8,7,6,5
7. Cyklus s pevným počtem opakování (cyklus for) – do proměnné cyklu „i“ jsou postupně přiřazeny hodnoty 0 až 10 s inkrementační konstantou 2.
8. Cyklus s pevným počtem opakování (cyklus for) – do proměnné cyklu „i“ jsou postupně přiřazeny hodnoty 10 až -6 s inkrementační konstantou -2.

4.2 Simulace průchodu

Jedna z nejdůležitějších funkcí aplikace je právě simulace průchodu programu vývojovým diagramem. Její návrh byl proto pro celou práci kritický a byla mu věnována patřičná pozornost.

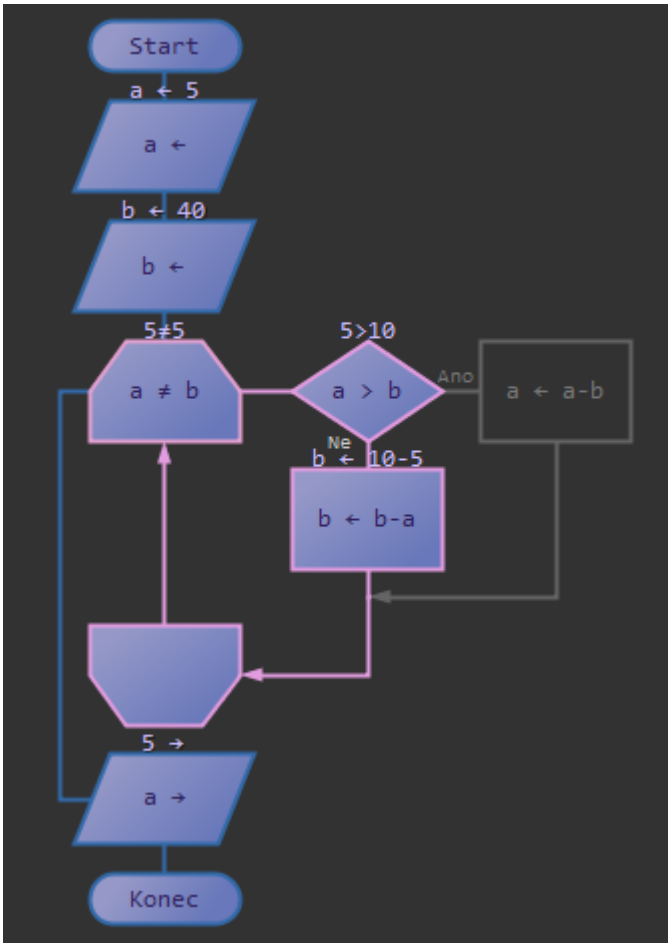
4.2.1 Vizualizace průchodu

Jedna z klíčových částí návrhu simulace průchodu vývojovým diagramem bylo určit metodu, kterou bude průchod vizualizován. Aplikace od Ivo Snozy (*viz kapitola 3.2.2*) poskytovala vizualizaci pouze v podobě průchodové kuličky, zatímco řešení Ing. Ivany Durdilové (*viz kapitola 3.2.1*) již zahrnovalo i ztučňování prošlých spojnic. Po inspiraci druhým z projektů bylo navrženo následující řešení (Obr. 31):

- Při aktivaci simulace průchodu vývojovým diagramem bude plynule změněno pozadí aplikačního plátna na tmavě šedou a bude tak navozeno vhodné prostředí pro výraznější vnímání barev.
- Každý symbol, který průchodem nebyl zatím aktivován, bude vykreslen nenápadně šedou barvou. Jakmile však k jeho aktivaci dojde, dojde i k jeho barevné výplni a bude tak možné jasně

rozeznat provedené symboly od těch, které aktivovány (zatím) nebyly.

- Bude-li spojnice či symbol aktivován více než jednou, bude tato událost indikována změnou barvy spojnice (obrysu symbolu) tak, že daná barva bude po každé další iteraci postupně nabývat světlejší odstín až do limitní bílé. Tímto způsobem pak bude možné přehledně rozeznat i taková místa ve vývojovém diagramu, kudy se algoritmus ubíral více než jednou. Rovněž bude možné odhadnout kolikrát se tak stalo.



OBR. 31 VIZUALIZACE PRŮCHODU VÝVOJOVÝM DIAGRAMEM

4.2.2 METODY PRŮCHODU

Jeden z cílů tohoto návrhu byl určit způsob, jakým bude vývojovým diagramem procházeno. Existující projekty volily průchod animací nebo krokováním, přičemž obě tyto metody byly shledány jako adekvátní a návrh by je proto měl obsahovat.

4.2.2.1 KROKOVÁNÍ

Krokování je nejzákladnějším způsobem, jak jednoduše kontrolovat průchod vývojovým diagramem. Voláním příkazu „krok vpřed“ je provedena funkce toho symbolu, který je aktuálně „na řadě“. Tento symbol by zároveň měl být nějakým způsobem předem označen, aby se uživatel na jeho exekuci mohl připravit. Toto označování tedy bylo přidáno do plánované realizace, načež započala úvaha o tom, zda by bylo možné implementovat i funkci kroku zpět. Taková funkcionalita by byla pro výuku jistě prospěšná, v běžných debuggrech[20] ovšem není přístupná a bylo proto určitou výzvou pokusit se ji realizovat.

Problém implementace funkce kroku zpět je v tom, že v některých situacích neexistuje způsob, jak strojově vyhledat předchozí provedenou instrukci. Pro funkční implementaci kroku zpět je tak nutné následovat některou z těchto dvou metod:

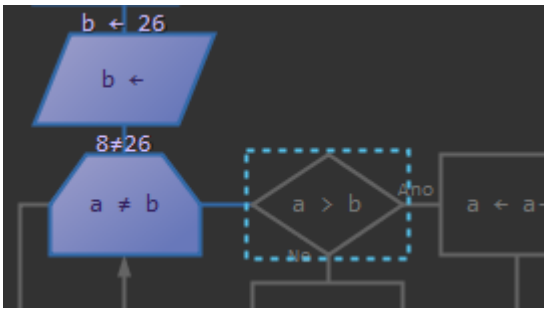
1. Uložit stav paměti před každým provedeným příkazem, přičemž krok zpět pak bude proveden obnovením tohoto záznamu.
2. Spustit program znovu až do místa těsně před provedením aktuálně vykonaného příkazu.

První způsob vyžaduje větší množství operační paměti, protože každým krokem je třeba provést otisk paměti stávající (proměnné, jejich hodnoty, prošlé symboly, spojnice apod.).

Druhý způsob se jeví na první pohled proveditelněji, ovšem předcházel-li by místu aktuálně vykonaného symbolu delší proces výpočtů, byl by uživatel nucen po každém kroku zpět čekat na jejich opětovné provedení. Bylo by navíc nutné uložení všech uživatelských vstupů a vygenerovaných náhodných čísel, aby proces mohl proběhnout přesně tak, jak byl proveden předtím.

[20] Debugger je integrovaná součást programátorského vývojového prostředí, sloužící k nalezení chybových míst v programu [36].

Po zvážení obou z možností, byla pro implementaci funkce kroku zpět zvolena první metoda s tím, že bude ukládáno jen nutné minimum informací (např. ukládání jen těch proměnných, které byly aktuálním krokem nějakým způsobem ovlivněny), aby takových kroků mohl být uložen co nejvyšší počet.

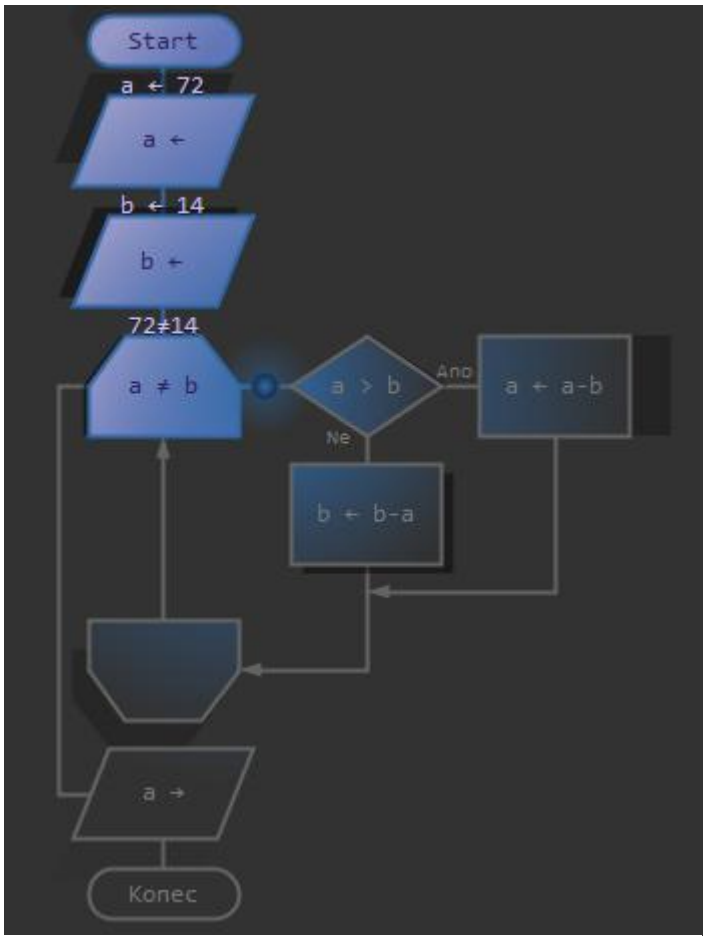


OBR. 32 KROKOVÁNÍ

4.2.2.2 ANIMACE

Animace v existujících projektech trpěla jistými nedostatky v oblasti vytížení procesoru, v její nedostatečné přehlednosti a také zdlouhavosti.

Animace simulace průchodu diagramem by měla působit reprezentativně a měla by tak být realizována efektivně, atraktivně a přehledně. Aby na žáky animace působila přátelsky a mohli se třeba i těšit z jejího spuštění, byla navržena tak, že průchozí kulička bude zářit a osvěcovat tak okolní symboly. Tyto symboly pak navíc budou na základě záře kuličky vrhat stín a animace se tak stane skutečným „představením“. Rychlost kuličky bude navíc realizována sinovou funkcí a uživatel tak nebude muset pozorovat její konstantní pohyb, jako je tomu u existujících projektů. Tato rychlost bude též automaticky upravována na základě délky dráhy, kterou bude muset kulička urazit a nebude tak nikdy po žádné spojnici putovat příliš dlouho.



OBR. 33 ANIMACE

4.2.2.3 FUNKCE SPUSTIT RYCHLE

Jako další možnost simulace průchodu vývojového diagramu byla navržena funkce jeho rychlého spuštění. Tato funkce by měla simulovat spuštění průchodu tak, jak by tomu bylo při spuštění samostatného programu. Vývojový diagram je tak procházen vysokou rychlostí automaticky, dokud není dosažen jeho konec nebo stanovený breakpoint (zarážka). Breakpoint pak bude možné stanovit před spuštěním samotného procesu rychlého procházení.

4.2.2.4 MEZIVÝPOČTY SYMBOLŮ

Během simulace průchodu programu vývojovým diagramem budou rovněž vykresleny mezivýpočty funkcí symbolů. Žák tak přímo z pohledu na konkrétní symbol získá lepší představu o tom, jak byla jeho funkce vykonána a proč byla proměnné přiřazena *právě tato hodnota*.

4.2.3 VOLBA VÝPOČETNÍHO JÁDRA

Pro vyhodnocování funkcí symbolů, které představují předem neznámé operace, bylo nutné najít takové řešení, které by je dokázalo dynamicky zpracovávat. Řešit tuto situaci uvnitř kompilovaného programovacího jazyka[21] by bylo velice komplikované, a proto byl pro tento účel hledán jazyk interpretovaný[22]. Programovací jazyk Java, jež byl zvolen pro vývoj této aplikace, naštěstí disponuje nativní podporou JavaScriptu[15] a nebylo tak nutné pro tuto funkcionalitu použít žádného externího nástroje. JavaScript navíc podporuje práci s poli a složitějšími matematickými operacemi, čehož aplikace bude moci využít.

4.2.4 PŘÍSTUP K PROMĚNNÝM

Dále bylo nutné zvážit, jaký přístup k proměnným bude simulace průchodu zaujímat. Obecně by to mohly být dva různé přístupy, jež jsou popsány níže.

Globální přístup

Při globálním přístupu vytvořené proměnné existují globálně - nezanikají a po jejich vytvoření je s nimi možné pracovat kdykoliv a kdekoliv v rámci celého vývojového diagramu.

Tento přístup je znám například z programovacího jazyka Pascal, kdy jsou proměnné deklarovány v hlavičce zdrojového kódu.

Blokový přístup

Při blokovém přístupu vytvořené proměnné existují jen v rámci svého bloku (větve symbolu) a ve větvích do něj vnořených. Jakmile se tok programu ocitne mimo tento blok (nebo jeho vnořené bloky), proměnná zaniká.

Tento přístup k proměnným známe z většiny moderních (objektových) programovacích jazyků, deklarujeme-li proměnnou uvnitř těla metod (funkcí, procedur).

Závěr

Jelikož má být aplikace určena pro výuku algoritmizace a tím i jakéhokoliv programovacího jazyka, bylo rozhodnuto, že by měly být podporovány oba

[21] Kompilovaný programovací jazyk musí být před spuštěním nejprve kompilátorem převedeny do strojového kódu [37].

[22] Interpretovaný programovací jazyk jsou před spuštěním převedeny do tzv. „vnitřní formy“ v níž jsou pomocí speciálního programu (interpretu) vykonávány (ibid.).

popsané přístupy. Volba přístupu k proměnným pak bude obsažena v nastavení aplikace.

4.3 Příprava vývoje

Před programováním jakékoliv aplikace většího rozsahu (ale i menšího), je vždy přínosné si její vývoj pečlivě naplánovat. V této etapě byl proto vytvořen diagram tříd stěžejních prvků aplikace a harmonogram pro vypracování této práce.

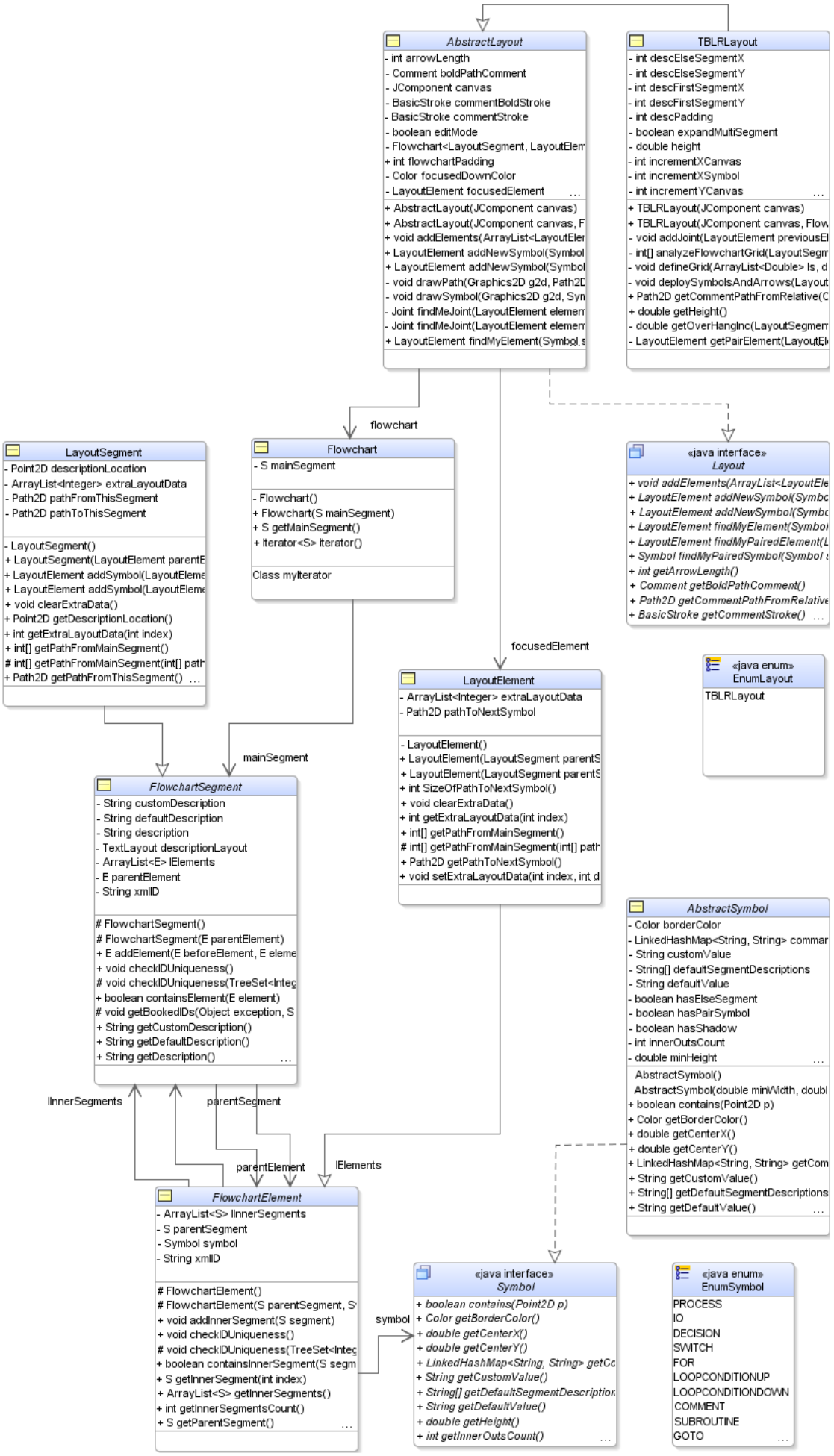
4.3.1 Diagram tříd aplikace

Diagram tříd je jedním z mnoha typů diagramů definovaných grafickým modelovacím jazykem UML [38]. Slouží k zobrazení statické struktury systému prostřednictvím tříd (třídy, rozhraní, výčtové typy) a vztahů mezi nimi. Diagram tříd se pak používá zejména pro dokumentování, zpětnou analýzu a objektový návrh systému. Právě pro objektový návrh byl v této práci použit a bylo tak možné dopředu sestrojit co nejefektivnější základní strukturu aplikace.

Hlavní cíl objektového návrhu aplikace spočíval v oddělení logické struktury vývojového diagramu od jeho grafického znázornění. Tato logická složka by pak byla univerzálně použitelná kterýmkoliv layoutem a zároveň by se tak naskytl způsob, jak diagram úsporně uložit do souboru.

Stručný popis diagramu tříd

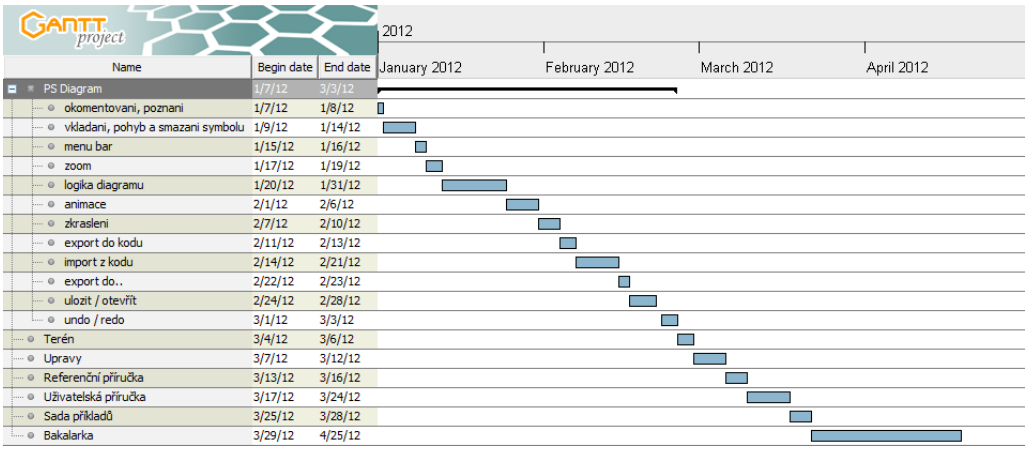
Logická konstrukce vývojového diagramu byla koncipována do stromové struktury (Obr. 34). Abstraktní třída *FlowchartSegment* představuje větev stromu a třída *FlowchartElement* jeho uzel, přičemž třída *Flowchart* tyto dva prvky zapouzdřuje a představuje tak samotný strom. *Element* reprezentuje právě jeden symbol vývojového diagramu, přičemž může obsahovat vnitřní *segmenty* (symboly s vlastní větví jako podmínka, cyklus apod.), které zapouzdřují samotné *elementy*. Třídy *LayoutElement* a *LayoutSegment* rozšiřující třídy *FlowchartElement* a *FlowchartSegment* pak představují prvky, které již obsahují informace o své grafické reprezentaci a jsou modifikovány layoutem. Díky použití tříd výčtových typů (*EnumLayout*, *EnumSymbol*) je pak aplikace velmi snadno rozšiřitelná. Další informace lze najít v technické dokumentaci aplikace (*viz kapitola 1.4 Přílohy práce*).



Obr. 34 Diagram stěžejních tříd aplikace

4.3.2 HARMONOGRAM VÝVOJE

Sestavený návrh aplikace byl poměrně ambiciózní, a bylo proto na místě, v zájmu dodržení termínu odevzdání práce, zhotovit časový harmonogram pro jeho realizaci. Pro tento účel byl zvolen tzv. Ganttův diagram (Obr. 35), kterým bylo možné vhodně zobrazit časové náročnosti a posloupnosti jednotlivých částí projektu.



OBR. 35 HARMONOGRAM VÝVOJE

5 VÝVOJ

Fáze vývoje představuje podle ADDIE modelu samotné vytvoření práce, a to podle plánu a návrhu připraveného v předchozím kroku. Během vývoje byly jednotlivé dílčí verze aplikace průběžně doručovány panu Mgr. Aleši Janatovi, který se (v rámci svých časových dispozic) zabýval jejich testováním a na jeho základě poskytoval zpětnou vazbu o tom, co je případně potřeba změnit.

Vzhledem ke značné rozsáhlosti celého projektu budou v následujících podkapitolách popsány jen takové části jeho vývoje, které byly nějakým způsobem zajímavé.

5.1 IMPLEMENTACE SYMBOLŮ VÝVOJOVÉHO DIAGRAMU

Jak již bylo zmíněno, programovací jazyk Java disponuje prostředky pro pokročilou práci s grafickým kontextem a bylo tak možné veškerý grafický obsah (včetně animace) implementovat vektorově s tím, že jej lze kdykoliv a bezztrátově přiblížit či oddálit.

Při vektorovém návrhu tvarů symbolů bylo pak nutné co nejpřesněji specifikovat jejich poměry stran a úhly, které svírají. Česká státní norma ČSN ISO 5807 však přesné informace o tvarech symbolů vývojového diagramu neposkytuje a po následném ručním měření bylo zjištěno, že jejich tvar je dokonce na různých stránkách odlišný. Pro určení tvarové specifikace symbolů tedy byla použita i norma od společnosti IBM [2], která již symboly zobrazuje uvnitř mřížky.

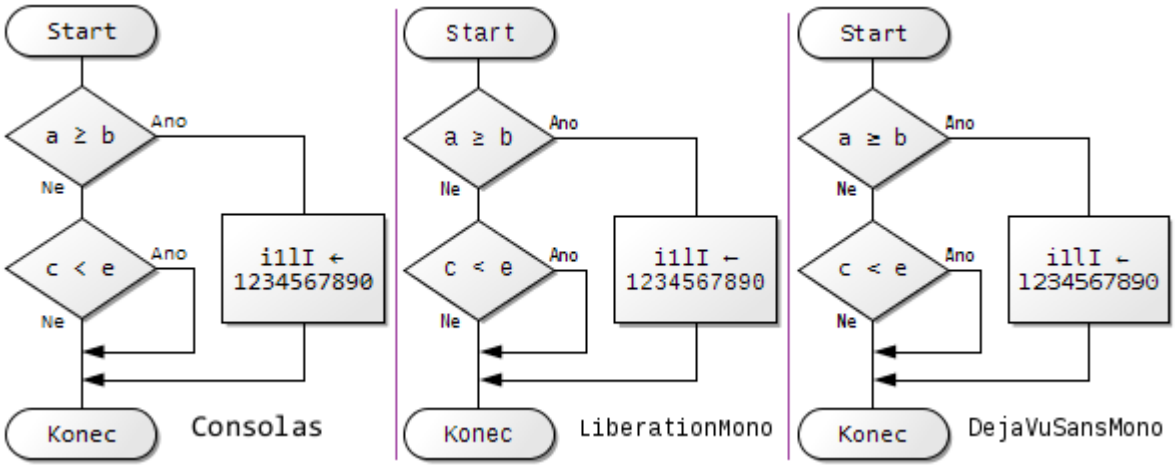
5.1.1 VOLBA STYLU PÍSMA

Jedna z těch méně nápadných, avšak klíčových voleb při vývoji aplikace byl výběr typu písma (tzv. font), který bude použit pro text uvnitř symbolů. Ač se tato problematika může jevit nenáročně, ukázalo se, že kladené nároky jsou velice přísné a volba tak byla poměrně komplikovaná.

Pro text uvnitř symbolů by měla platit stejná kritéria, jako pro text vhodný pro zobrazení zdrojového kódu (např. kódu programovacího jazyka). Písmo by tedy mělo majoritně splňovat tyto požadavky:

1. Neproporcionální
2. Snadno čitelné (i při nižších velikostech)
3. Zobrazování adekvátních mezer mezi jednotlivými znaky (souvisí s předchozím bodem)
4. Dostatečně odlišené podobné znaky (např. „o, 0, O“ a „1, l, I“)
5. Znak nuly s přeškrtnutím nebo tečkou uprostřed (`Ø` nebo `0`)
6. Podpora sady těchto znaků: „←, ≠, ¬, ≥, ≤“
7. Bezplatné

Na téma „*nejlepší font pro zobrazení zdrojového kódu*“ byly již sestaveny takové sady písem, které by těmto kritériím měly alespoň z části vyhovovat [39][40][41]. Na základě požadavků 1 až 5 bylo tedy shromážděno 7 fontů, které byly následně podrobeny testování požadavku šestého. Ukázalo se, že znak šipky činí fontům značnou překážku, protože jej nedokázaly zobrazit 4 z vybraných písem[23]. Zbylá 3 písma se tedy skládala z fontů *DejaVuSansMono*, *LiberationMono* a *Consolas*, a byly na zkoušku použity v aplikaci.



OBR. 36 TEST STYLŮ PÍSMA

Na Obr. 36 si lze všimnout, jak si jednotlivé fonty vedly v zobrazení některých kritických znaků. Problémy činil opět především znak šipky, který se při nulovém přiblížení v *LiberationMono* a *DejaVuSansMono* jevil nevýrazně. Druhý ze jmenovaných pak šipku dokonce zobrazuje tak, že se její spodní část

[23] Byly vyřazeny tyto písma: *Anonymous, AnonymousPro, Inconsolata a BitstreamVeraMono.*

jeví opticky menší. Obě jmenovaná písma oproti fontu *Consolas* trpí kostrbatým vykreslením znaku „menší (nebo rovno)“.

Naprosto jasným vítězem se proto v tomto testu stalo písmo *Consolas*, které všechny znaky dokázalo vykreslit velice čitelně. Písmo *Consolas* je ovšem ve vlastnictví firmy Microsoft a přestože je distribuováno s některými aplikacemi zdarma (Microsoft Office), není ho vzhledem k jeho licenci (MS EULA) možné do aplikace integrovat. Písmo *LiberationMono* pak muselo být vyřazeno taktéž, jelikož je distribuováno pod licencí GNU GPL a výsledná aplikace by tak byla vázána jejími podmínkami.

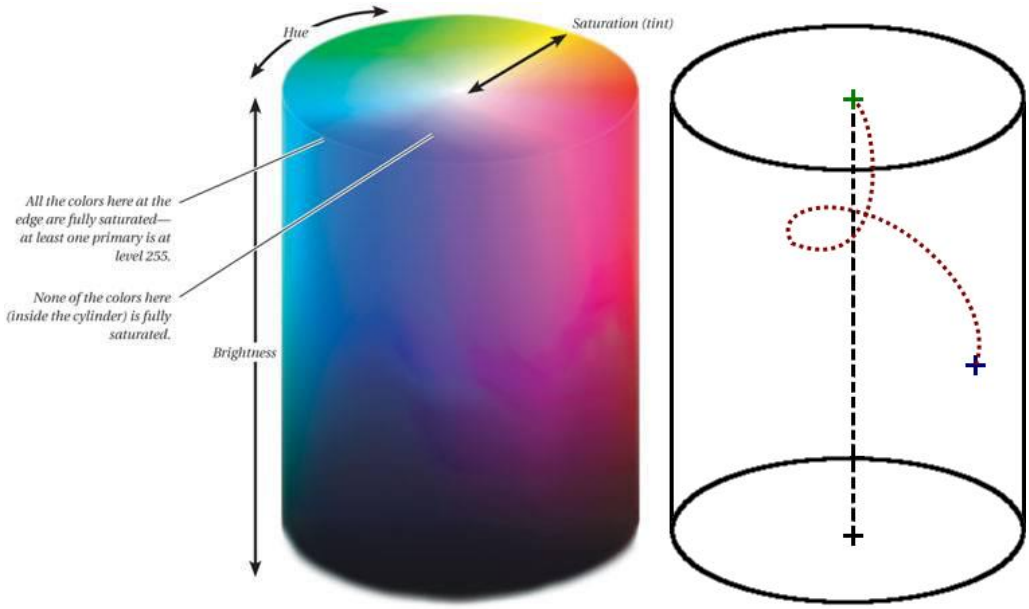
Řešení se tedy stává z algoritmu, který prověří, zda je písmo *Consolas* na cílovém systému k dispozici a pakliže ano (ve většině případů je tomu opravdu tak), bude pro vykreslení textů uvnitř symbolů použito. Není-li naopak toto písmo přítomné, je načten font *DejaVuSansMono*, který byl do aplikace integrován (je dostupný pod licencí Public Domain).

5.2 Implementace barevné vizualizace průchodu

Jeden ze sestavených návrhů vizualizace průchodu vývojovým diagramem se stával z postupných změn barvy obrysů symbolů a spojnic na základě jejich opětovné aktivace s tím, že každou další aktivací se tato barva bude stále více přibližovat limitní bílé. Aby tento efekt byl ještě znatelnější a bylo ho tak možné uživatelem snadněji rozeznat, byla do této barevné modifikace zahrnuta i pozvolná změna barevného odstínu.

Po zanalyzování možností řešení této úlohy bylo jisté, že v tradičním barevném modelu RGB by se úloha řešila jen obtížně. Byl proto použit model HSB (známý také jako HSV), který barvu definuje na základě takových složek, jejichž postupnou modifikací lze kýženého výsledku docílit snadněji. Barevný model HSB definuje barvu na základě tří složek [42]:

- *Hue* - barevný odstín. Měří se úhlem na standardním barevném kole (0° až 360°).
- *Saturation* - sytost barvy. Udává množství barvy v poměru k šedé a měří se v procentech od 0 % (šedá) do 100 % (plně sytá barva bez šedé).
- *Brightness/Value* - jas. Udává relativní světlost nebo tmavost barvy (opět v procentech).



OBR. 37 MODEL HSB A ŘEŠENÍ BAREVÉHO PRŮCHODU

Model HSB je možné zobrazit jako válec (Obr. 37)[24], přičemž cílová bílá barva se nachází v jeho vrchní části uprostřed (zelený křížek na obrázku vpravo). Byla-li tedy určena kterákoliv barva jako barva prvotní aktivace symbolu (modrý křížek na obrázku vpravo), bylo nutné z tohoto místa vést křivku takového tvaru, aby byly rovnoměrně vystřídány všechny možné barevné kombinace až do finální bílé barvy (červená křivka na obrázku vpravo). Tato křivka byla následně vzorkována libovolným počtem barevných odběrů, které již mohly být použity pro kýžený efekt barevného přechodu.

Na následujícím výpisu ze zdrojového kódu aplikace lze pozorovat, jak bylo toto řešení naprogramováno s tím, že cílová barva nebyla použita přímo bílá, neboť se na tmavě šedém pozadí jevila příliš agresivně (konstanty PATHMIN_SATURATION_PERCENTAGE a PATHMAX_BRIGHTNESS_PERCENTAGE určují hodnoty jasu a saturace cílové barvy). Dále bylo nutné první a druhý barevný vzorek od sebe odlišit výrazněji, aby druhá aktivace symbolu (nebo spojnice) byla markantnější (další aktivace jsou již vzorkovány pomocí pravidelného dělení průchodové křivky).

[24] Zdroj obrázku HSB modelu: http://flylib.com/books/en/2.471.1.41/1/

```
pathColors[0] = PATHCOLOR; // přiřazení barvy neaktivované cesty
pathColors[1] = PATHBASECOLOR; // přiřazení barvy prvotní aktivace cesty
float hsbVals[] = Color.RGBtoHSB(PATHBASECOLOR.getRed(),
        PATHBASECOLOR.getGreen(), PATHBASECOLOR.getBlue(), null);

// je třeba 1. a 2. aktivaci od sebe odlišit výrazněji
hsbVals[1] += (PATHMIN_SATURATION_PERCENTAGE - hsbVals[1]) /
        (pathColors.length - 2) * (pathColors.length / 2);
hsbVals[2] += (PATHMAX_BRIGHTNESS_PERCENTAGE - hsbVals[2]) /
        (pathColors.length - 2) * (pathColors.length / 2);

float fractionH = 1.0f / (pathColors.length - 2);
float fractionS = (PATHMIN_SATURATION_PERCENTAGE - hsbVals[1]) /
        (pathColors.length - 2);
float fractionB = (PATHMAX_BRIGHTNESS_PERCENTAGE - hsbVals[2]) /
        (pathColors.length - 2);
for (int i = 1; i + 1 < pathColors.length; i++) {
    pathColors[i + 1] = Color.getHSBColor(fractionH * i + hsbVals[0],
        fractionS * i + hsbVals[1], fractionB * i + hsbVals[2]);
}
```

5.3 Implementace pohybu průchodové kuličky

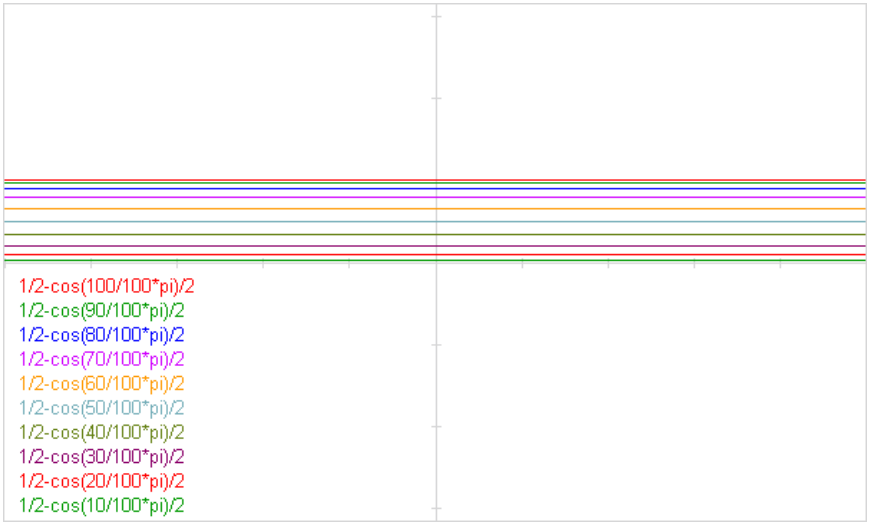
Koncept modulace rychlosti pohybu kuličky byl navržen tak, že bude z počátečního bodu vyslána rychlostí minimální. Student tak bude mít čas zpozorovat, z kterého symbolu je tato kulička vyslána a jakým směrem se bude ubírat. Rychlost průchodové kuličky pak bude postupně stoupat až do momentu, kdy kulička dosáhne poloviny cílové vzdálenosti a tím bude zároveň urychlena ta část její dráhy, která nepředstavuje nic zajímavého. Následně bude podobným způsobem její rychlost klesat, dokud nedorazí na místo určení. Studentovi tak bude opět ponechán čas navíc, aby se připravil na aktivaci funkce dalšího symbolu.

Aby tento koncept mohl být realizován, bylo nutné sestavit takovou matematickou funkci, která by popsanému chování odpovídala a zároveň poskytovala výstup použitelný pro další zpracování. Výsledná funkce má následující podobu:

$$f(x) = \frac{1}{2} - \frac{\cos\left(\frac{část}{celek} * \pi\right)}{2}$$

Do rychlosti pohybu kuličky je započítán i čas, za který musí být dráha překonána (určený na základě délky spojnice). Proměnná *část* tedy představuje hodnotu, kolik času již od počátku pohybu uplynulo a proměnná *celek* kolik

času bylo pro překonání vzdálenosti určeno. Výstup funkce (Obr. 38) má pak podobu čísla v rozmezí 0 až 1, který udává procento délky dráhy, v kterém se kulička má v daném okamžiku nacházet.



OBR. 38 FUNKCE PRO VÝPOČET CÍLOVÉ POZICE PRŮCHODOVÉ KULIČKY

Do algoritmu pohybu průchodové kuličky byla navíc přidána proměnná v podobě aktuální hodnoty jezdce z grafického rozhraní aplikace, která má na starosti celý proces pohybu kuličky urychlovat či zpomalovat. Uživateli bylo dovoleno i dynamicky měnit počet snímků za sekundu, jímž se animace vykresluje a bylo nutné zajistit i kompletní optimalizaci animace tak, aby ji bylo možné co nejplynuleji pozorovat i na starších počítačích. Výsledný algoritmus, který řídí pohyb kuličky je tedy rozsáhlejší a v případě zájmu je k nahlédnutí ve zdrojovém kódu aplikace na přiloženém CD (viz metoda *myActionPerformed* ve vnitřní třídě *PlayTimer* třídy *Animator* v balíčku *diagram.animation*).

5.4 IMPLEMENTACE FUNKCE UNDO/REDO

Cílem vývoje systému pro sestavování vývojového diagramu pomocí layoutu bylo zajistit uživateli při této činnosti co nevyšší komfort. Tohoto cíle bylo dosahováno nejen samotným layoutem, ale i umožněním kopírování / vyjmutí / vkládání symbolů a zajištěním provozu funkcionality kroku zpět (tzv. Undo) a znovu (tzv. Redo) při jejich editaci (např. při nechtěném smazání symbolu je možné akci vrátit). Právě funkce Undo a Redo byla řešena originálním způsobem, jemuž budou věnovány následující odstavce.

Pro podporu Undo a Redo funkce Java poskytuje přeprogramovanou třídu *UndoManager*, která zajišťuje základní očekávané chování této činnosti (zejména manipulace se zásobníkem návratových akcí). Programátorovi poté již stačí jen implementovat rozhraní *UndoableEdit*, které představuje právě jednu návratovou akci, která v sobě zapouzdřuje i metody pro své konkrétní provedení Undo a Redo funkcionality. Obecně se poté uplatňuje takový postup, kdy je pro každou jednotlivou návratu-možnou akci vytvořena právě jedna třída [43]. Měl-li by tedy být umožněn návrat např. akcí vložit symbol, smazat symbol a upravení textu symbolu, musely by být vytvořeny právě tři třídy implementující rozhraní *UndoableEdit*. Každá taková třída by pak zvlášť řešila provoz své návratnosti a obnovitelnosti.

Po naprogramování dvou takových návratových akcí bylo však zjištěno, že přestože byly naprogramovány s ohledem na úsporu paměti, kdy bylo uloženo jen minimum potřebných objektů pro provedení jejich funkce, představují relativně vysokou paměťovou náročnost a nebylo by tak možné uložit vyšší množství takovýchto kroků. Byl proto vyvinut jiný, univerzální a paměťově méně náročný způsob, jak docílit stejné funkcionality bez nutnosti vytváření tříd pro každou novou návratovou akci.

Řešení využívá přeložení vývojového diagramu do XML, které obsahuje jen takové nezbytné množství informací, aby mohl být diagram obnoven[25]. Těsně před tím, než je provedena jakákoliv změna ve vývojovém diagramu, která by ovlivnila strukturu jeho XML podoby, je proveden jeho XML otisk. Jakmile je daná změna provedena, je uložený XML otisk porovnán se stávajícím a jsou vyhledána místa, která byla touto změnou ovlivněna. Následuje uložení jen takových řetězců, které byly při porovnání otisků odlišné, a tím je této metodě zajištěna velice nízká paměťová náročnost[26].

Pro obnovení nebo vrácení provedené akce je pak pomocí uložených informací a aktuálně vygenerovaného XML otisku obnoven XML otisk původní, který je následně načten aplikací a je tak docíleno úspěšného provedení této činnosti.

[25] Neobsahuje žádné údaje o spojnicích mezi symboly, o umístění symbolů a dalších prvcích, které je možné znovu vypočítat layoutem.

[26] Další snížení paměťové náročnosti by mohlo být docíleno aplikováním komprimačního algoritmu

Výhodou tohoto přístupu je jeho univerzálnost a nižší paměťové nároky, díky kterým bylo možné určit limit návratových akcí na 1000 [27]. Nevýhoda pak spočívá v tom, že je nutné zaznamenat opravdu všechny možné změny, které mají jakýkoliv dopad na XML podobu vývojového diagramu, neboť by v opačném případě došlo k desynchronizaci a obnova XML otisku by neproběhla úspěšně (jsou k dispozici jen části otisku, které jsou vázány na jeho původní podobu). Pro další informace viz třída *UniversalEdit* v balíčku *diagram.gui.managers.undoableEdits*.

5.5 Implementace syntaktického analyzátoru

Jedna z opravdu nadstandardních funkcí, která byla stanovena i v zadání práce, sestává z možnosti vygenerování vývojového diagramu ze zdrojového kódu programovacího jazyka a zároveň naopak možnosti vývojový diagram do některého z podporovaných jazyků exportovat[28]. Import ze zdrojového kódu by pak byl dobře využitelný např. učiteli v případě, kdy by rádi prezentovali žákům algoritmus, který již mají vytvořený v některém z podporovaných programovacích jazyků. Export pak mohou využít např. zvídaví žáci ke zkoumání výsledku své práce interpretované ve skutečném programovacím jazyce.

Pro to, aby bylo možné docílit takové funkcionality, bylo nutné vytvořit vlastní syntaktický analyzátor (známý také pod pojmem parser), který umožňuje zdrojový kód umístěný na jeho vstup zpracovat a sestavit podle něj objekt odpovídajícího a funkčního vývojového diagramu. Podobným způsobem pak dokáže vývojový diagram přeložit do programovacího jazyka.

Při návrhu parseru byla použita referenční příručka pro programovací jazyk Pascal [14], nicméně se ukázalo, že nepopisuje jeho detaily příliš podrobně, a pro jejich analýzu byl proto používán přímo kompilátor tohoto jazyka [44]. Zejména při exportu vývojového diagramu jsou pak např. korektně použity středníky na koncích řádků zdrojového kódu, jež nejsou v Pascalu používány pro ukončení příkazů, jako je tomu u většiny programovacích jazyků, nýbrž k jejich oddělování [45].

[27] Jedna návratová akce obsahuje nejčastěji nosnou informaci do velikosti 700 bytů.

[28] Jako podporovaný programovací jazyk byl zvolen Pascal (viz kapitola 3.3.1), nabídku podporovaných jazyků je však možné jednoduše rozšířit přidáním záznamu do patřičné enumerační třídy.

Pro realizaci syntaktického analyzátoru bylo hojně využito regulárních výrazů, jejichž funkce mají ovšem své hranice, a tak pro rozlišení některých struktur, jako například několikanásobně vnořené komentáře ve zdrojovém kódu, muselo být naprogramováno vlastní řešení. Výsledný analyzátor je tak z vloženého zdrojového kódu schopen nejen vygenerovat vývojový diagram včetně korektního zacházení se všemi běžnými příkazy, ale i zpracovat všechny obsažené komentáře.

Import

Import vývojového diagramu ze zdrojového kódu byl vytvořen tak, že je-li ve vloženém kódu syntaktická chyba, která zapříčiňuje nemožnost vytvoření diagramu, je o této skutečnosti uživatel obeznámen dialogem s tím, že zpráva, která je mu zobrazena, obsahuje i výpis prvku, který chybu způsobuje a uživatel ji tak následně může snadno opravit.

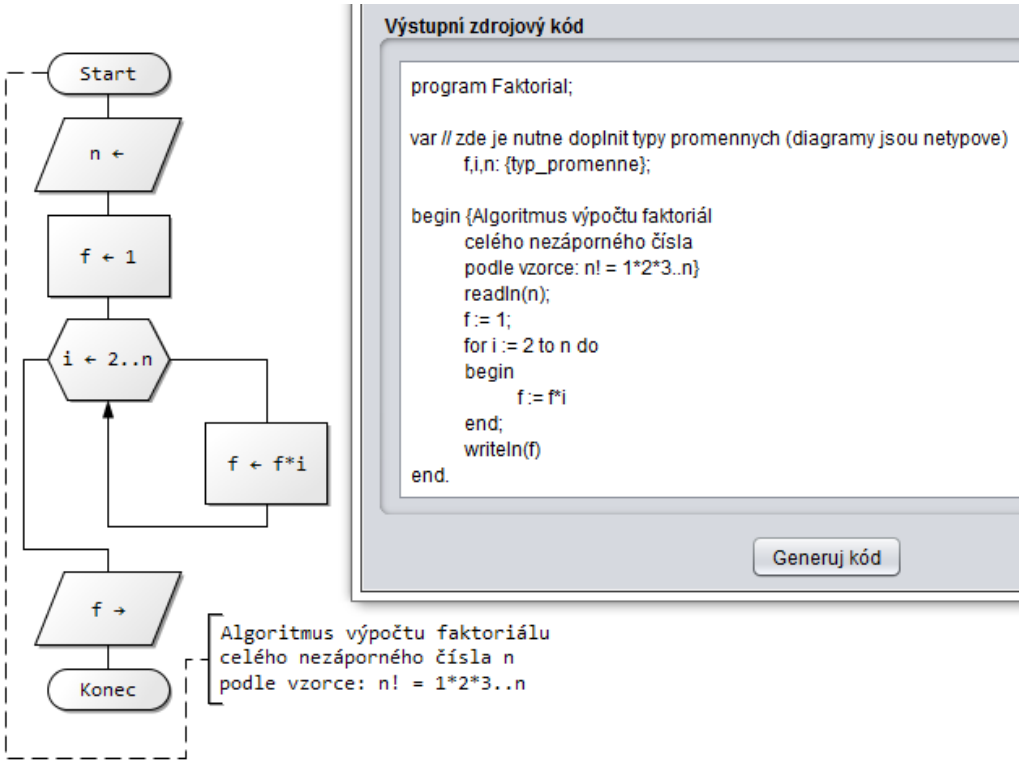
Pakliže se v kódu vyskytuje prvek, který aplikace nedokáže ve vývojovém diagramu interpretovat, je diagram přesto vytvořen. Na místě prvku, který nebylo možné přeložit, je pak zobrazen symbol anotace obsahující upozornění a textovou formu tohoto prvku. Uživatel tak má opět jasnou představu, který prvek se nepodařilo importovat.

Export

Export vývojového diagramu prostřednictvím zdrojového kódu byl navržen tak, aby jeho výstup byl uživatelsky co nejčitelnější, a je proto implementován včetně korektního použití odřádkování a tabulátorů.

Samotný vývojový diagram byl ve výsledné aplikaci koncipován tak, aby proměnné, které uživatel používá, nemusely být dopředu deklarovány. Vytvoří-li tedy uživatel symbol zpracování a definuje-li v jeho funkci identifikátor proměnné a její hodnotu k přiřazení, daná proměnná je vytvořena, aniž by muselo být specifikováno, o jaký její datový typ se jedná. Právě tato informace o datovém typu je však pro téměř jakýkoliv programovací jazyk klíčová.

Jelikož určení datového typu takto zadaných hodnot není nijak univerzálně a jednoduše proveditelné, je nutné, aby jednotlivé datové typy určil již sám uživatel. Pro usnadnění tohoto úkolu byl však vytvořen takový algoritmus, který všechny proměnné obsažené ve vývojovém diagramu vyhledá a s informací o nutnosti tohoto doplnění je zapíše na správné místo do výstupního zdrojového kódu (Obr. 39).



OBR. 39 EXPORT VÝVOJOVÉHO DIAGRAMU

6 OVĚŘENÍ APLIKACE V PRAXI

Jedním z cílů této práce bylo výslednou aplikaci rovněž otestovat v praktickém provozu na některé škole, aby tak byly odhaleny její případné nedostatky před samotnou fází implementace[29].

Pro tento účel byla již od počátku domluvena přednáška na Trutnovské Střední průmyslové škole, odkud na této práci spolupracoval prostřednictvím testování aplikace pan učitel Mgr. Aleš Janata. Pan učitel však bohužel těsně před termínem domluvené návštěvy na delší dobu onemocněl a přednáška tak nemohla být realizována. Ověření aplikace v praxi se tedy konalo na Střední odborné škole elektrotechnické v Hluboké nad Vltavou, jejíž vedení s otestováním aplikace rádo souhlasilo.

6.1 PŘÍPRAVA

Ihned po mém ranním příchodu mne uvítal laskavý úsměv pana Ing. Josefa Turka, tamějšího učitele informatiky, se kterým jsem měl domluvenu svou přednášku. Po mém představení panu zástupci ředitele Mgr. Jiřímu Mrázkovi, jsme se s panem učitelem odebrali do jeho kabinetu, kde jsme započali debatu na téma vývojové diagramy. Bylo mi řečeno, že vývojové diagramy jsou povinnou součástí studentské ročníkové práce z předmětu programování v Pascalu, kdy mají sloužit jako podklad k řešení zadané úlohy a že k jejich sestavování v hodině nepoužívají žádný specializovaný software.

Poté co byla na všechny počítače v učebně s pomocí správce sítě pana Bc. Jana Krejsy nainstalována aktuální verze běhového prostředí Javy, jejíž přítomnost je pro běh programu nezbytná, byla panu učiteli Josefu Turkovi aplikace představena. Jelikož na něj učinila ohromný dojem, vyburcoval následně dalšího svého kolegu, aby se přišel na plánovanou přednášku i s jeho žáky také podívat.

6.2 PRŮBĚH

V počítačové učebně byl nakonec spolu se dvěma třídami a učiteli přítomen i pan zástupce ředitele a přednáška tak získala neočekávaně početné publikum.

[29] Fáze implementace je podle ADDIE modelu „stěžejní krok, v němž je vytvořená práce nasazena a spuštěna cílovému publiku“ [8].

Všichni přítomní žáci aktuálně studovali 3. ročník oboru Elektronické počítačové systémy, po optání jsem však byl seznámen s informací, že neabsolvovali žádnou výuku programování, protože je tento předmět dostupný jen jako jeden z povinně volitelných a v daném ročníku o něj nebyl zájem. Přednáška tak byla poněkud zkomplikována, neboť přítomní žáci byli zaměřeni na síťové technologie nebo počítačovou grafiku a bylo je nutno seznámit se základy algoritmizace.

V úvodu přednášky byli tedy žáci seznámeni s pojmy algoritmus a vývojový diagram spolu s popisem jeho základních symbolů. Pro demonstraci funkcí jednotlivých symbolů byla použita simulace průchodu vývojovým diagramem pomocí prezentované aplikace. Při této činnosti žáci poznávali i způsob práce s aplikací a koncepci umisťování symbolů pomocí vkládacích bodů. Následně byl ve spolupráci s přítomnými žáky vytvořen jednoduchý algoritmus, který na základě náhodně vygenerovaného čísla nechal uživatele opakovaně hádat jeho hodnotu. Bylo-li tipované číslo menší nebo větší než náhodné, byla uživateli o této skutečnosti zobrazována odpovídající zpráva, dokud hodnota náhodného čísla nebyla uhodnuta.

Ve druhé části přednášky byl pak žákům zadán úkol v podobě sestrojení algoritmu pro výpočet faktoriálu, který měli sestavit každý samostatně na svém počítači. Příjemným překvapením se pak stalo zjištění, že si žáci při práci s aplikací počínali velice obratně a neměli sebemenší problémy s jejím ovládáním ani logikou. Ukázalo se však, že obtížnost zadaného úkolu byla zřejmě zvolena příliš vysoká, neboť si většina žáků (patrně pro jejich zaměření) nevěděla s úkolem příliš rady. Po drobné pomoci však někteří žáci správné řešení přeci jen dokázali sestavit.

Na závěr hodiny bylo všem žákům představeno správné řešení zadané úlohy, a to opět metodou simulace průchodu vývojovým diagramem, s jejíž pomocí byly jednotlivé kroky algoritmu podrobně popsány a vysvětleny.

Na otázku jak se žákům aplikace líbí a zda se jim s ní dobře pracovalo, odpovídali jednoznačně kladně. Oceňovali především layout a systém vkládání symbolů.

6.3 Shrnutí

Na základě pozorování žákovské zručnosti při manipulaci s aplikací lze konstatovat, že uživatelské rozhraní aplikace (po předchozím krátkém úvodu)

působí z hlediska ovladatelnosti zcela intuitivně. Dále bylo přednáškou prokázáno, že je aplikace vhodná nejen pro prezentaci předem vytvořených algoritmických struktur, ale i pro jejich vytváření přímo během vyučovací hodiny.

Z psychologického aspektu na žáky přednáška působila rovněž kladně. Práce s aplikací je dle jejich slov bavila, někteří žáci dokonce z vlastní vůle setrvali v učebně přes část přestávky, aby zde řešený úkol v aplikaci dokončili.

V oblasti technického aspektu aplikace byly zprvu drobné obavy o plynulost animace na místním pomalejším učitelském počítači, ta však byla nakonec vykreslována bez sebemenších problémů.

6.3.1 Odhalené nedostatky

Během praktického používání aplikace byly odhaleny 3 její nedostatky, všechny jsou však řešitelné:

1. Spuštění aplikace trvá na starších počítačích poměrně dlouhou dobu a uživatel tak neví, zda na ikonu aplikace poklepal správně.
2. Během simulace je chybně zobrazována diakritika proměnných řetězcového typu.
3. Je-li symbolu cyklu s pevným počtem opakování zadána hodnota proměnné cyklu existující proměnnou a stejná proměnná je použita i pro počáteční hodnotu počítadla tohoto cyklu, je hodnota této existující proměnné smazána a cyklus se tak stane nekonečným, což neodpovídá běžnému chování této konstrukce (*for (a=a; a<=10; a++)*).

První odhalený nedostatek se běžně objevuje u všech složitějších aplikací a jeho řešení spočívá v použití tzv. „splash screen“, což je obrázek, který je uživateli zobrazen ihned při spouštění aplikace takovou dobu, než je plně načtena a zobrazena.

Druhý problém je způsoben tím, že funkce JavaScriptu *uneval()*, která je pro vyhodnocování funkcí symbolů používána, překládá veškeré znaky, které nejsou obsaženy v základní ASCII tabulce, do tzv. unicode kódů (např. slovo „dům“ je pak přeloženo jako „d\u016fm“). Funkce *uneval()* bohužel nepřijímá žádné parametry, jež by toto chování mohly ovlivnit, a tak jedno z možných řešení spočívá ve vyhledání a následném nahrazení těchto kódů jejich pro člověka čitelnými podobami.

Třetí problém spočívá v chybné implementaci chování inicializace cyklu s pevným počtem opakování, kdy je jako signál pro nutnost inicializace proměnné cyklu použito vymazání případné existující proměnné stejného identifikátoru. Jako řešení se pak nabízí použití jakéhokoliv jiného postupu určení signálu pro inicializaci proměnné cyklu (např. volání metody s odpovídajícím parametrem).

7 VÝSLEDKY PRÁCE

Výsledkem práce je poměrně robustní aplikace[30] pojmenovaná PS Diagram, jejíž kompletní popis a charakteristiky lze najít přímo v uživatelské příručce (*viz Příloha A*). Pro aplikaci byla rovněž zhotovena technická dokumentace a sbírka 7 základních příkladů vývojových diagramů, které lze aplikací otevřít a následně realizovat jejich simulaci průchodu (*viz kapitola 1.4 Přílohy práce*).

7.1 DIDAKTICKY PŘÍNOSNÉ PRVKY

Do aplikace se podařilo integrovat tyto didakticky přínosné prvky:

- Během sestavování vývojového diagramu jsou jeho symboly automaticky zarovnávány a korektně propojovány spojnicemi, díky čemuž je sestavování diagramu časově nenáročné a aplikace je tak vhodná i pro jejich vytváření přímo během výuky.
- Jelikož se uživatel nemusí nijak starat o umisťování jednotlivých symbolů, vzniká tak žákovi rovněž prostor pro plnou koncentraci na nalezení správného řešení zadané úlohy.
- Textové hodnoty symbolů vývojového diagramu jsou generovány automaticky (s ohledem na co nejvyšší výstupní uživatelskou srozumitelnost) na základě jeho vyplněné funkce a příkaz přiřazení hodnoty do proměnné je tak zobrazen pomocí znaku šipky („←“), což působí srozumitelně i pro žáky kteří nemají zkušenosti s programováním (*viz kapitola 4.1.5*).
- Průchod algoritmického průběhu vývojového diagramu je možné realizovat více způsoby, přičemž při krokování je při průchodu poskytnuta i nadstandardní funkce kroku zpět (*viz kapitola 4.2.2.1*).
- Během průchodu vývojového diagramu je zobrazován jak výpis aktuálních proměnných, tak mezivýpočty funkcí symbolů. Celá vizualizace průchodu je navíc obohacena o barevné rozlišení symbolů a spojnic na základě toho, zda jimi tok algoritmu prošel, prošel jednou či prošel vícekrát. Je možné i zpětně odhadnout, o kolik průchodů se jednalo.

[30] Výsledná aplikace je tvořena přesně 84 třídami a obsahuje 23 565 řádků zdrojového kódu spolu s celkem 989 042 znaky.

7.2 Možné rozšíření

Aplikace byla již od počátku vyvíjena s maximálním ohledem na její možnost pozdějšího rozšiřování.

Po implementaci patřičného rozhraní je tedy možné rozšířit nejen sadu symbolů vývojového diagramu a dostupné layouty, ale taktéž podporované programovací jazyky pro funkcionalitu jejich exportu a importu z/do vývojového diagramu. Pro usnadnění implementace byly rovněž vytvořeny patřičné abstraktní třídy a po umístění záznamu o novém rozšíření do náležité třídy enumeračního typu je tento prvek automaticky zařazen do grafického rozhraní aplikace.

7.2.1 Plánované rozšíření

Výsledná aplikace bude dále rozšiřována, přičemž jsou vedle opravení stávajících nedostatků naplánovány tyto vylepšení:

- automatické aktualizace;
- zakomponování uživatelské příručky a příkladů vývojových diagramů přímo do aplikace;
- vícenásobné označování symbolů pro kopírování, mazání a vyjímání;
- přibližování plátna k aktuální pozici kurzoru;
- přidání možnosti obnovení výchozího nastavení aplikace;

Projeví-li pak školy zájem, je možné aplikaci rozšířit i o podporu volání funkcí a procedur.

7.3 Licence

Aplikaci prozatím nestanovuji žádnou konkrétní licenci, podléhá tak standardnímu autorskému právu [31] . Rozhodl jsem se tak na základě plánovaných vylepšení, aby aplikace nebyla šířena bez možnosti automatické aktualizace.

[31] Jakékoliv dotazy pak směřujte na email: miroslavbartyzal@gmail.com.

8 Závěr

V rámci této bakalářské práce vznikla aplikace, která umožňuje rychlé a efektivní sestavení vývojového diagramu programu s následnou možností přehledné vizualizace jeho algoritmického průchodu.

Během práce jsem navrhl a realizoval několik originálních konceptů, které mají za úkol urychlit proces sestavování vývojového diagramu. Vytvářené symboly vývojového diagramu jsou automaticky umisťovány a korektně propojovány spojnicemi a žák je tak zproštěn jakékoliv starosti s jeho formátováním. Při sestavování diagramu tím žákovi vzniká prostor pro plnou koncentraci na nalezení správného řešení zadané úlohy. Vkládání symbolů a definování jejich funkcí je navíc díky metodě tzv. *párového označování* a automaticky generovaného textu symbolů velice svižné a po předchozím ověření aplikace v praxi na střední škole je možné konstatovat, že je aplikace vhodná nejen pro prezentaci předpřipravených algoritmických úloh, ale i k jejich vytváření přímo během vyučovací hodiny.

Následná vizualizace průchodu vývojovým diagramem byla naprogramována tak, aby poskytovala co možná nejširší možnosti pozorování chování procházeného algoritmu. Žákovi tak byl poskytnut přehled o aktuálních proměnných a jejich hodnotách, mezivýpočtech funkcí symbolů a o počtu aktivací jednotlivých symbolů. Samotnou vizualizaci pak lze provést nejen animací, ale i krokováním (včetně kroku zpět) nebo rychlým spuštěním, při němž byla poskytnuta možnost vložení zarážek.

Za nejkomplikovanější oblast během programování aplikace považuji implementaci layoutu, při níž bylo dosti náročné pokrýt všechny alternativy návazností symbolů tak, aby jejich umístění, propojení spojnicemi a proložení vkládacími body bylo vykresleno korektně.

Pro aplikaci hodlám vytvořit vlastní internetové stránky a po opravení nalezených nedostatků a implementaci plánovaných vylepšení bych ji rád nabídl českým školám pro nasazení do výuky.

LITERATURA

[1] ČSN ISO 5807. *Zpracování informací - dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy programu a diagramy zdrojů systému*. Česká republika : Český normalizační institut, Leden 1996. 28 s.

[2] GC20-8152-1. *IBM Data Processing Techniques*: *Flowcharting Techniques*. New York: International Business Machines Corporation, 1970. Dostupné z: http://www.fh-jena.de/~kleine/history/software/IBM-FlowchartingTechniques-GC20-8152-1.pdf

[3] TAUFER, I., HRUBINA, J., TAUFER, J.:. *Algoritmy a algoritmizace: vývojové diagramy, sbírka řešených příkladů. Pardubice: Univerzita Pardubice, 2001*. Kopp, České Budějovice, 1997. ISBN 80-86148-49-1.

[4] TAUFER, Ivan. *Algoritmy a algoritmizace - vývojové diagramy*. Vyd. 1. Pardubice: Univerzita Pardubice, 2009, 92 s. ISBN 978-80-7395-182-5.

[5] PŠENČÍKOVÁ, Jana. *Algoritmizace*. Vyd. 2. Kralice na Hané: Computer Media, 2009, 128 s. ISBN 978-80-7402-034-6.

[6] ADDIE Model. *Learning-Theories.com* [online]. © 2008-2012 [cit. 2012-04-15]. Dostupné z: http://www.learning-theories.com/addie-model.html

[7] JANOCH, Petr. *Kvalitativní výzkum e-learningu v prostředí současné firmy* [online]. Brno, 2008 [cit. 2012-04-15]. Dostupné z: http://is.muni.cz/th/74595/ff_m/. Diplomová práce. Masarykova univerzita, Filozofická fakulta. Vedoucí práce doc. Mgr. Jiří Zounek, Ph.D.

[8] SEDLÁČKOVÁ, Barbora. *E-learningový kurz knihovnické angličtiny 2 (e-LKA2)* [online]. Brno, 2011 [cit. 2012-04-15]. Dostupné z: http://is.muni.cz/th/217841/ff_m/. Diplomová práce. Masarykova univerzita, Filozofická fakulta. Vedoucí práce PhDr. Tamara Váňová.

[9] DOHNAL, Pavel. *Programování a programovací jazyky ve vzdělávání na ZŠ* [online]. Brno, 2009 [cit. 2012-04-15]. Dostupné z: http://is.muni.cz/th/72886/pedf_m/. Diplomová práce. Masarykova univerzita, Pedagogická fakulta. Vedoucí práce Ing. Martin Dosedla.

[10] KLEMENT, Milan a Jan LAVRINČÍK. *Úvod do MS Visual Basic 2010*: *Interaktivní studijní opora* [online]. první vydání. Olomouc, 2011 [cit. 2012-04-15]. ISBN 978-80-87557-07-5. Dostupné z: http://www.pros.upol.cz/files/others/vzdelavaci-moduly/modul_1.pdf

[11] SNOZA, Ivo. *Interaktivní dynamické vývojové diagramy* [online]. Pardubice, 2009 [cit. 2012-04-15]. Dostupné z: http://dspace.upce.cz/handle/10195/34891. Bakalářská práce. Univerzita Pardubice, Katedra softwarových technologií. Vedoucí práce Ing. Soňa Neradová.

[12] DURDILOVÁ, Ivana; SUCHÁNKOVÁ, Lenka. *Animace uživateli vytvořených interaktivních vývojových diagramů pro výuku algoritmizace* [online]. 2005, 2006 [cit. 2012-04-18]. Dostupné z WWW: <http://lab.uzlabina.cz/~projekty/index.htm>.

[13] HANÁK, Jan. *C++/CLI - Začínáme programovat* [online]. první. Artax, 2009 [cit. 2012-04-15]. ISBN 978-80-87017-04-3. Dostupné z: http://download.microsoft.com/download/8/0/E/80E9ED8D-F63F-4912-A5B8-A1CFB186571D/Zaciname_programovat_v_Cpp_CLI.pdf

[14] VAN CANNEYT. *Free Pascal: Reference guide.*: *Reference guide for Free Pascal* [online]. 2011 [cit. 2012-04-15]. Document version 2.6. ISBN neuvedeno. Dostupné z: ftp://ftp.freepascal.org/fpc/docs-pdf/ref.pdf

[15] *PortableApps.com* [online]. 2004 [cit. 2012-04-14]. What is a portable app?. Dostupné z WWW: <http://portableapps.com/about/what_is_a_portable_app>.

[16] Vítejte u NetBeans a na stránkách www.netbeans.org. ORACLE CORPORATION. *NetBeans* [online]. © 2012 [cit. 2012-04-14]. Dostupné z: http://netbeans.org/index_cs.html

[17] Algoritmus. *Algoritmy.net* [online]. © 2008-2012 [cit. 2012-04-15]. Dostupné z: http://www.algoritmy.net/article/1240/Algoritmus

[18] VANĚK, Tomáš. Algoritmus a jeho vlastnosi. *Tomáš Vaněk* [online]. datum vydání neuveden [cit. 2012-04-15]. Dostupné z: http://dupeto.cz/~tomik/index.php?vybrano=alg

[19] PODHRÁZSKÝ, Milan. Pseudokód. *Algoritmy a datové struktury v Javě* [online]. datum vydání neuveden, 20.7.2002 [cit. 2012-04-15]. Dostupné z: http://www.alg.webzdarma.cz/diplomka/kap2/pseudokod.html

[20] Procesy a vlákna. BENEŠ, Miroslav. *Fakulta elektrotechniky a informatiky, VŠB-TUO*: *Katedra informatiky* [online]. datum neuvedeno [cit. 2012-04-16]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/pte/texty/vlakna/ch01s01.html

[21] ADDIE Instructional Design Model. *About e-Learning* [online]. © 2007-2012 [cit. 2012-04-17]. Dostupné z: http://www.about-elearning.com/addie-instructional-design-model.html

[22] 7. Ukazatele, pole a retezce. SALOUN, Petr. *Programování v jazyce C* [online]. 1996 [cit. 2012-04-18]. Dostupné z: http://melkor.dnp.fmph.uniba.sk/~zenis/prirucky/c/kap07.htm

[23] Drag-and-drop. *Webopedia* [online]. c2012 [cit. 2012-04-19]. Dostupné z: http://www.webopedia.com/TERM/D/drag_and_drop.html

[24] Visio 2010. *Microsoft Office* [online]. c2012 [cit. 2012-04-19]. Dostupné z: http://office.microsoft.com/cs-cz/visio/

[25] *SmartDraw* [online]. ©2012 [cit. 2012-04-19]. Dostupné z: http://www.smartdraw.com/

[26] YEd - Graph Editor. *YWorks* [online]. © 2012 [cit. 2012-04-19]. Dostupné z: http://www.yworks.com/en/products_yed_about.html

[27] *Lucidchart* [online]. © 2012 [cit. 2012-04-19]. Dostupné z: https://www.lucidchart.com/

[28] *Gliffy* [online]. © 2012 [cit. 2012-04-19]. Dostupné z: http://www.gliffy.com/

[29] ČERNÝ, Michal. UML a Flowchart ve výuce programování na gymnáziu. *Metodický portál: Články* [online]. 30. 09. 2011, [cit. 2012-04-19]. Dostupný z WWW: <http://clanky.rvp.cz/clanek/c/G/13721/UML-A-FLOWCHART-VE-VYUCE-PROGRAMOVANI-NA-GYMNAZIU.html>. ISSN 1802-4785.

[30] UML - Unified Modeling Language. KUČEROVÁ, Helena. *Vyšší odborná škola informačních služeb* [online]. 2011, 10.12.2011 [cit. 2012-04-19]. Dostupné z: http://web.sks.cz/users/ku/pri/uml.htm

[31] UML, alea iacta est!. BENEŠOVSKÝ, Miroslav a Karel RICHTA. *Katedra Softwarového Inženýrství MFF UK* [online]. Brno, 18-21.10.2008 [cit. 2012-04-19]. Dostupné z: http://www.ksi.mff.cuni.cz/~richta/publications/TutorialUML.pdf

[32] UML 2 Activity Diagram. *Sparx Systems* [online]. © 2000-2012 [cit. 2012-04-20]. Dostupné z: http://www.sparxsystems.com/resources/uml2_tutorial/uml2_activitydiagram.html

[33] Slovník. *Stargen* [online]. © 2000-2011 [cit. 2012-04-20]. Dostupné z: http://www.stargen.cz/slovnik/XML/

[34] JavaScript v Mozille Úvod. *Mozilla developer network* [online]. 2007 [cit. 2012-04-20]. Dostupné z: https://developer.mozilla.org/cs/JavaScript_v_Mozille/Úvod

[35] MORKES, David. Java Applet krok za krokem. *Interval.cz* [online]. 27. 09. 2002 [cit. 2012-04-20]. Dostupné z: http://interval.cz/clanky/java-applet-krok-za-krokem/

[36] Pojem debugger. *ABZ slovník cizích slov* [online]. © 2005-2006 [cit. 2012-04-21]. Dostupné z: http://slovnik-cizich-slov.abz.cz/web.php/slovo/debugger

[37] Konstrukce překladače. *Algoritmy.net* [online]. c 2008-2012 [cit. 2012-04-21]. Dostupné z: http://www.algoritmy.net/article/100/Konstrukce-prekladace

[38] UML: Diagramy tříd. *Miloš Němec* [online]. 15. 5. 2010 [cit. 2012-04-22]. Dostupné z: http://www.milosnemec.cz/clanek.php?id=199

[39] Top 10 Programming Fonts. BENJAMIN, Dan. *Hivelogic* [online]. [17 May 2009] [cit. 2012-04-22]. Dostupné z: http://hivelogic.com/articles/top-10-programming-fonts/

[40] Anonymous Pro: a programming font with style. BENJAMIN, Dan. *Hivelogic* [online]. [12 June 2009] [cit. 2012-04-22]. Dostupné z: http://hivelogic.com/articles/anonymous-pro-programming-monospace-font/

[41] Font Survey: 42 of the Best Monospaced Programming Fonts. DIETRICH, Hans. *CodeProject* [online]. [18 Aug 2010] [cit. 2012-04-22]. Dostupné z: http://www.codeproject.com/Articles/30040/Font-Survey-42-of-the-Best-Monospaced-Programming

[42] Barevný model HSB (HSV). PIHAN, Roman. *FotoRoman* [online]. © 2011 [cit. 2012-04-23]. Dostupné z: http://fotoroman.cz/glossary2/3_hsb.htm

[43] Add Undo/Redo functionality to a Java app. *ProcessWork Blog* [online]. [4 August 2009] [cit. 2012-04-23]. Dostupné z: http://www.processworks.de/blog/2009/08/add-undoredo-functionality-to-a-java-app/

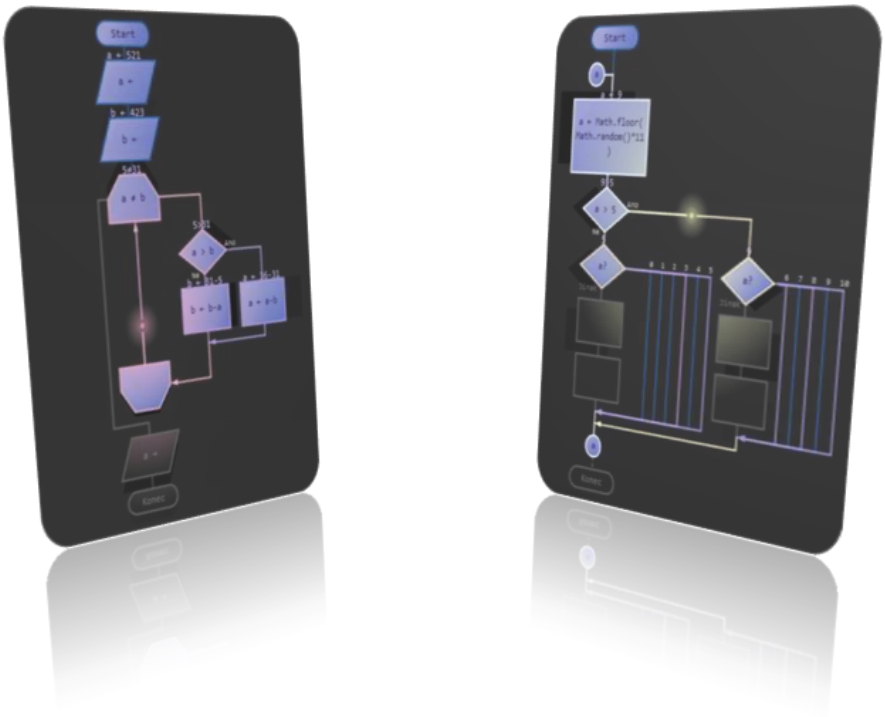
[44] Online Compiler (.net) PASCAL. *Online Compiler* [online]. rok vydani neuveden [cit. 2012-04-24]. Dostupné z: http://www.onlinecompiler.net/pascal

[45] Pascal descriptions and it's functions. *Function . name* [online]. rok vydání neuveden [cit. 2012-04-24]. Dostupné z: http://function.name/in/Pascal

PŘÍLOHA A

UŽIVATELSKÁ PŘÍRUČKA K APLIKACI PS DIAGRAM

Příručku lze taktéž nalézt na přiloženém CD



UŽIVATELSKÁ PŘÍRUČKA K APLIKACI PS DIAGRAM

autor: Miroslav Bartyzal
kontakt: miroslavbartyzal@gmail.com

OBSAH

1 Úvod

Tato uživatelská příručka slouží jako manuál a nápověda k aplikaci PS Diagram. Aplikace PS Diagram slouží k vytváření vývojových diagramů a vizualizaci jejich algoritmického průběhu pro výuku algoritmizace.

Díky zabudovanému layoutu aplikace poskytuje velmi rychlý způsob, jak vývojový diagram sestavit, přičemž nabízí širokou škálu použitelných symbolů z oblasti vývojového diagramu programu, navrženou v souladu s českou státní normou ČSN ISO 5807. Atraktivní a přehledný způsob vizualizace jejich průchodu pak do výuky přináší vhodný nástroj pro procházení algoritmů.

Poznámka:
Ke spuštění aplikace je nutné mít na počítači nainstalovanou aktuální verzi Java SE Runtime Environment (JRE), která je zdarma ke stažení na oficiálních stránkách Oracle zde.

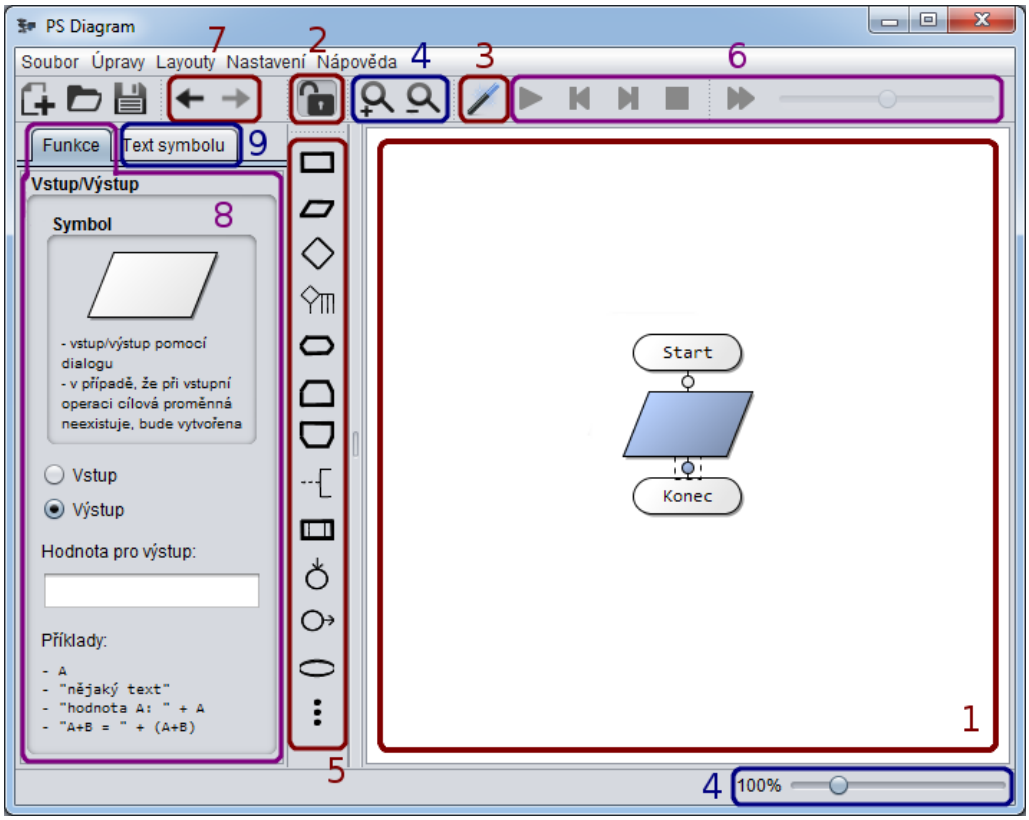
1.1 Charakteristika aplikace

- ✓ Vývojový diagram lze díky jeho vektorovému formátu kdykoliv přiblížit či oddálit, aniž bychom přišli o kvalitu jeho vykreslení (*kapitola 2.2*).
- ✓ Aplikace je přehledně rozdělena do třech oddělených režimů: režim náhledu (*kapitola 2*), editační režim pro samotné vytváření vývojových diagramů (*kapitola 3*) a režim animační, zajišťující vizualizaci jejich algoritmického průchodu (*kapitola 4*).
- ✓ Aplikace disponuje inteligentním zarovnáváním vkládaných symbolů vývojového diagramu do zvoleného layoutu. Symboly jsou zároveň automaticky korektně propojovány spojnicemi a uživatel je tak zproštěn jakýchkoli starostí s formátováním diagramu (*kapitola 3.1*).
- ✓ Přidávání a editace funkce či textu symbolů vývojového diagramu je díky *vkládacím bodům* a *párovému označování* velice svižné a efektivní (*kapitola 3.1.2*).

- ✓ Textové hodnoty symbolů vývojového diagramu jsou generovány automaticky (s ohledem na co nejvyšší výstupní uživatelskou srozumitelnost) na základě jeho vyplněné funkce (*kapitola 3.4.1*).
- ✓ Průchod algoritmického průběhu vývojového diagramu je možné realizovat více způsoby, přičemž při *krokování* je při průchodu poskytnuta i nadstandardní funkce kroku zpět (*kapitola 4.2*).
- ✓ Během průchodu vývojového diagramu je zobrazován jak výpis aktuálních proměnných, tak mezivýpočty funkcí symbolů. Celá vizualizace průchodu je navíc obohacena o barevné rozlišení symbolů a spojnic na základě toho, zda jimi tok algoritmu prošel, prošel jednou či prošel vícekrát. Je možné i zpětně odhadnout, o kolik průchodů se jednalo (*kapitola 4.1*).
- ✓ Vývojový diagram je možné importovat (vygenerovat) z vloženého zdrojového kódu některého z podporovaných programovacích jazyků (*kapitola 7*) nebo vývojový diagram naopak do zdrojového kódu exportovat (*kapitola 8*).

1.2 GRAFICKÉ ROZHRANÍ APLIKACE

Na obrázku Obr. 1 je zobrazeno hlavní okno aplikace. Čísly jsou označeny jeho hlavní ovládací prvky, jejichž vysvětlivky následují níže.



OBR. 1 GRAFICKÉ ROZHRANÍ APLIKACE

1. Plátno s vývojovým diagramem
2. Přepínání mezi aplikačním režimem editace / náhledu
3. Přepínání mezi aplikačním režimem animace / náhledu
4. Přibližování / oddalování vývojového diagramu
5. Dostupné symboly vývojového diagramu programu
6. Panel pro ovládání animačního režimu aplikace
7. Tlačítka zpřístupňující funkce Undo (vrátit akci) a Redo (obnovit akci)
8. Záložka editace funkce vybraného symbolu
9. Záložka editace textu vybraného symbolu

1.3 REŽIMY APLIKACE

Aplikace obsahuje celkem 3 různé režimy zobrazení vývojového diagramu. Je to režim náhledu, editační režim a režim animační. Diagram lze v jednu chvíli zobrazovat vždy jen v jednom z těchto režimů.

Jednotlivými režimy se v této příručce budeme podrobněji zabývat v samostatných kapitolách, které následují.

2 REŽIM NÁHLEDU

V režimu náhledu se nacházíme automaticky, jsou-li editační a animační režimy ve vypnutém stavu (Obr. 2, popisek 1 a 2).

Vývojový diagram je v tomto režimu zobrazován v prosté formě tak, jak ho lze následně exportovat (viz kapitola 6). Tato prostá forma zobrazení neumožňuje jakoukoliv editaci vývojového diagramu, poskytuje však uživateli přehledný způsob, jak vývojový diagram zkontrolovat vůči případným logickým chybám apod.

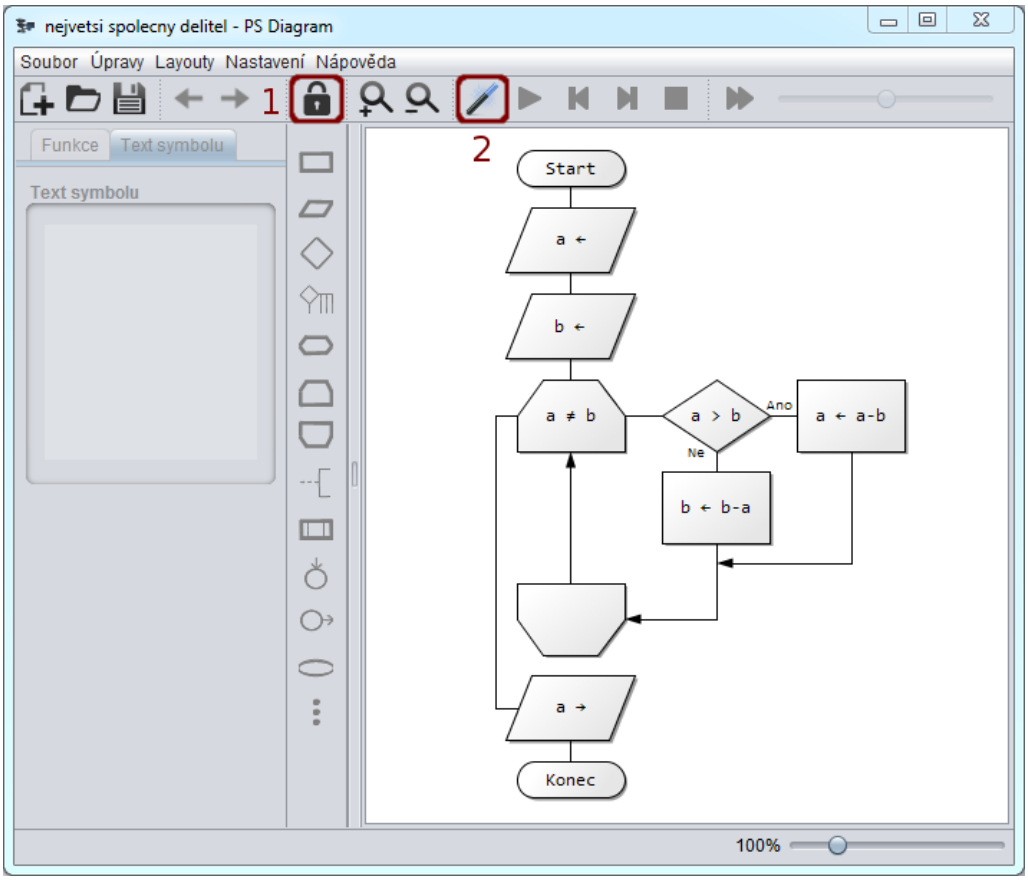
2.1 MANIPULACE S VÝVOJOVÝM DIAGRAMEM

Režim náhledu zároveň umožňuje vývojovým diagramem intuitivně manipulovat.

Použijeme-li kolečko myši, můžeme diagram rolovat nahoru a dolů, tak jak jsme zvyklí z ostatních aplikací. Přidržíme-li navíc klávesu Alt, docílíme rolování doleva a doprava.

Pohodlnější způsob jak manipulovat s vývojovým diagramem je přidržením levého tlačítka myši nad plátnem (Obr. 1, popisek 1). Pohybujeme-li pak myší do různých směrů, kurzor myši se nám změní v ručičku a pozice diagramu kopíruje její pohyb.

Plátno pro vývojový diagram je nekonečné, nejsme tak omezeni jeho velikostí a můžeme si dovolit vytvořit libovolně složité algoritmické struktury. Při manipulaci s nekonečným plátnem nám však nehrozí, že ztratíme přehled o pozici našeho diagramu. Vývojovým diagramem lze totiž pohybovat vždy maximálně v okolí zobrazeného plátna + velikost diagramu, diagram se tak nachází vždy maximálně o jeden pixel za hranící viditelného plátna.



OBR. 2 REŽIM NÁHLEDU

2.2 PŘIBLIŽOVÁNÍ

Vývojový diagram lze kdykoliv přiblížit či oddálit. Veškerá grafika, která je vykreslena na plátně, je realizována vektorově. Při přibližování tak nepřicházíme o kvalitu vykreslení, jak by tomu bylo u grafiky bitmapové.

Přiblížení nebo oddálení vývojového diagramu lze ovládat třemi způsoby. První dva způsoby ilustruje Obr. 1 (popisek 4). Třetí způsob spočívá v přidržení klávesy Ctrl + použití kolečka myši nad plátnem vývojového diagramu (Obr. 1, popisek 1).

3 EDITAČNÍ REŽIM

Do editačního režimu vstoupíme sepnutím tlačítka zámku (Obr. 1, popisek 2) v hlavním panelu aplikace.

Tento režim nám zpřístupní panel dostupných symbolů (Obr. 1, popisek 5) a my tak můžeme vývojové diagramy vytvářet. K vytváření diagramů využijeme i záložku Funkce symbolu (Obr. 1, popisek 8) a Text symbolu (Obr. 1, popisek 9), jejichž účelem se budeme zabývat později.

K tomu, abychom porozuměli systému vkládání symbolů, si v následující kapitole představíme layout.

Poznámka:
Editační režim spolupracuje s režimem náhledu, s vývojovým diagramem tak lze nadále manipulovat obvyklým způsobem.

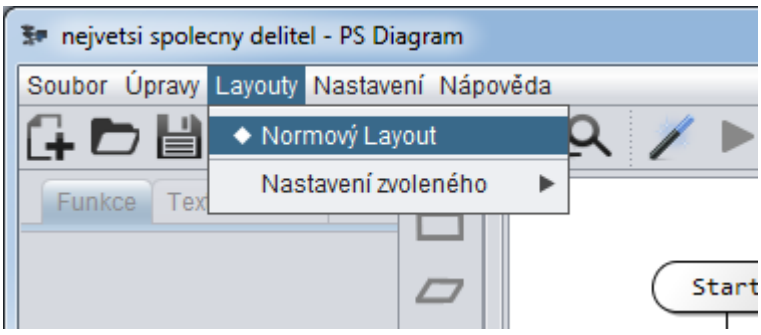
3.1 LAYOUT

Aplikace PS Diagram disponuje inteligentním zarovnáváním vkládaných symbolů do zvoleného layoutu. Symboly jsou zároveň automaticky korektně propojovány spojnicemi a uživatel je tak zproštěn jakýchkoli starostí s formátováním diagramu.

Tím, že při vytváření vývojového diagramu odpadají starosti s jeho formátováním, vzniká uživateli prostor pro koncentraci na samotný vytvářený algoritmus a nalezení správného řešení dané úlohy. Vytváření diagramů se tímto zároveň stává otázkou okamžiku.

3.1.1 DOSTUPNÉ LAYOUTY

Požadovaný layout lze zvolit v hlavní nabídce aplikace pod tlačítkem Layouty (Obr. 3). V dolní části této nabídky lze navíc nalézt případná další nastavení právě zvoleného layoutu.



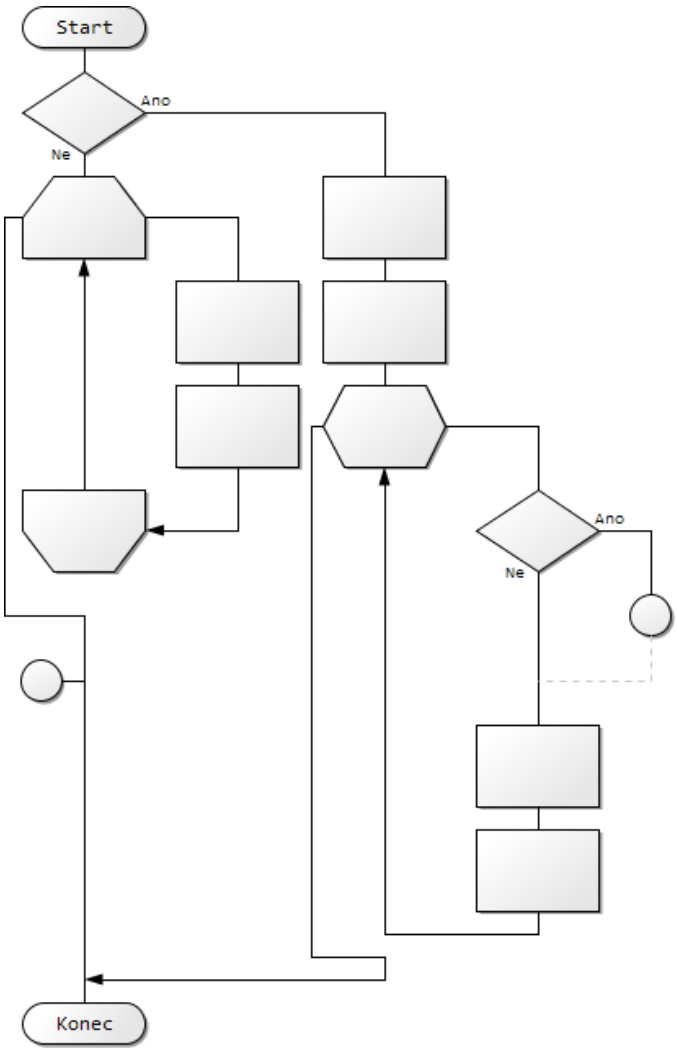
OBR. 3 VOLBA LAYOUTU

Poznámka:
Aplikace zatím obsahuje pouze layout Normový, v případě zájmu je však s rozšířením nabídky layoutů počítáno.

3.1.1.1 NORMOVÝ LAYOUT

Normový layout (Obr. 4) představuje výchozí layout aplikace. Byl navržen tak, aby co nejvíce odpovídal předpisům české státní normy ČSN ISO 5807 týkající se vývojových diagramů.

Směr toku tohoto layoutu směřuje zleva doprava a shora dolů, přičemž spojnice do symbolů vstupují zleva nebo shora a vystupují vpravo nebo dole. Ke spojnicím jsou automaticky doplňovány směrové šipky, směřují-li vzhledem k počátku jinam než do IV. kvadrantu.



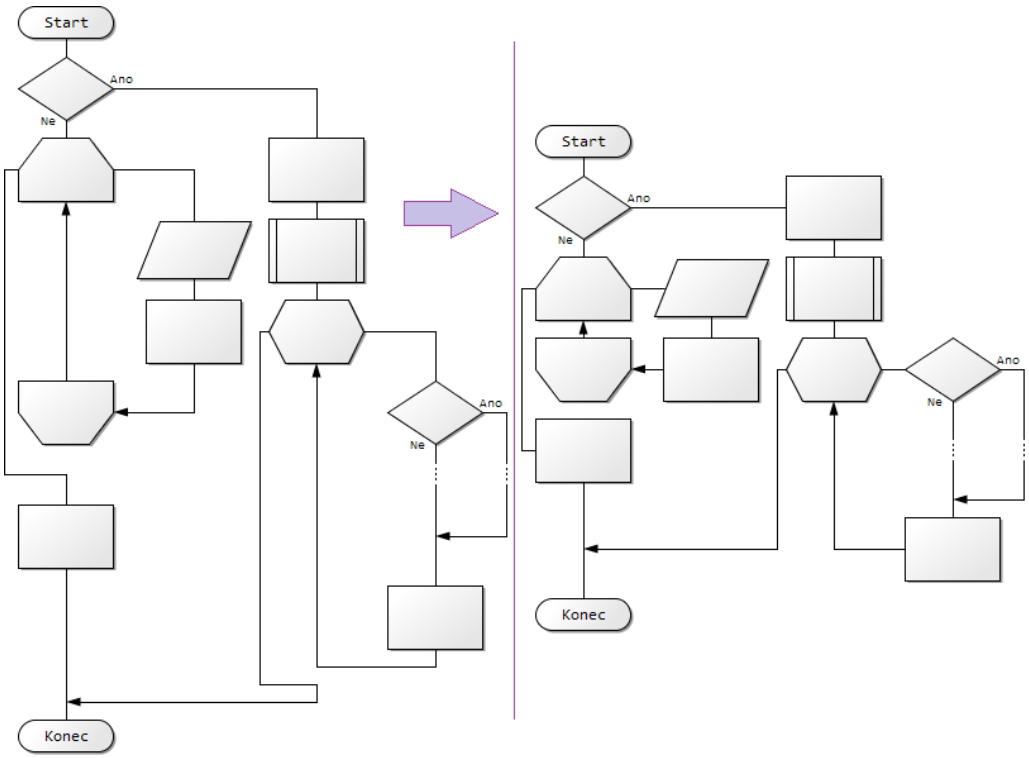
OBR. 4 NORMOVÝ LAYOUT

3.1.1.1.1 MOŽNÁ NASTAVENÍ

Normový layout poskytuje dvě možná přizpůsobení.

3.1.1.1.1.1 SMRŠŤOVÁNÍ

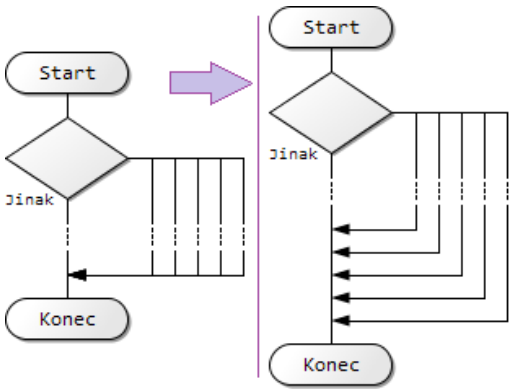
Možnost „*Smršťování*“ nám umožní symboly v sloupcích skládat vedle sebe a navazovat postranní šipky cyklů na následující symboly zleva (Obr. 5). Tuto možnost využijeme zejména tehdy, chceme-li např. vývojový diagram vytisknout a potřebovali bychom zmenšit jeho velikost. Při zapnutém smršťování lze vývojový diagram rovněž editovat, nicméně tento režim je méně přehledný, než editace při vypnutém smršťování.



OBR. 5 SMRŠŤOVÁNÍ

3.1.1.1.1.2 EXPANZE SWITCH SYMBOLU

Druhé přizpůsobení nese název „*Expanze switch symbolu*“. Switch symbol, označovaný také jako „vícenásobné rozhodování“, se od ostatních symbolů odlišuje vyšším počtem spojnic, které ze symbolu vycházejí. Obecně existuje více způsobů, jak tyto spojnice zobrazovat - PS Diagram spojnice defaultně vykresluje paralelně. Povolením této volby se konce těchto spojnic navrací jednotlivě, tak jak je to možné pozorovat na obrázku (Obr. 6).

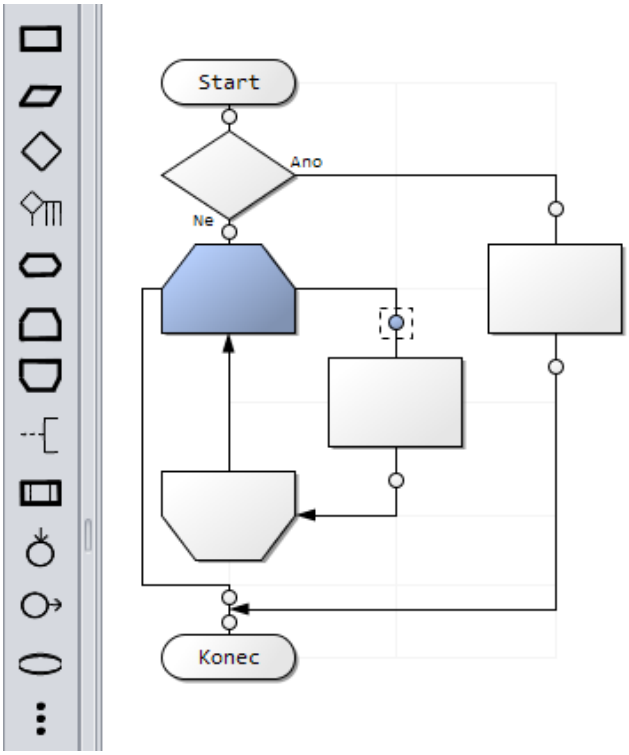


OBR. 6 EXPANZE SWITCH SYMBOLU

3.1.2 PŘIDÁVÁNÍ SYMBOLŮ

Přidávání symbolů se v aplikaci PS Diagram realizuje velmi specificky.

Nacházíme-li se v Editačním režimu aplikace, na místech mezi symboly jsou vykresleny tzv. *vkládací body* (Obr. 7). Vkládací body nám označují místa, kam lze potencionálně vložit nový symbol. Označíme-li si požadovaný vkládací bod a klikneme-li poté v panelu dostupných symbolů (Obr. 7, panel vlevo) na požadovaný symbol, dojde k jeho automatickému začlenění do vývojového diagramu.



OBR. 7 VKLÁDACÍ BODY

3.1.2.1 PÁROVÉ OZNAČOVÁNÍ

Všimněte si, že v jednom okamžiku je vždy označen pár symbol-vkládací bod (označení značí modrá barva). Tato funkcionalita zajišťuje svižnost při vkládání a editaci symbolů, neboť umožňuje tyto dvě činnosti provádět naráz, bez dalšího označování. Po vložení symbolu tak můžeme ihned editovat např. jeho text či funkci, přičemž je aplikace stále připravena na přidání symbolu následujícího.

V systému označování figuruje i označení rámované. Toto označení určuje místo, kam bude vložen případný přidávaný symbol komentáře.

3.2 ÚPRAVY

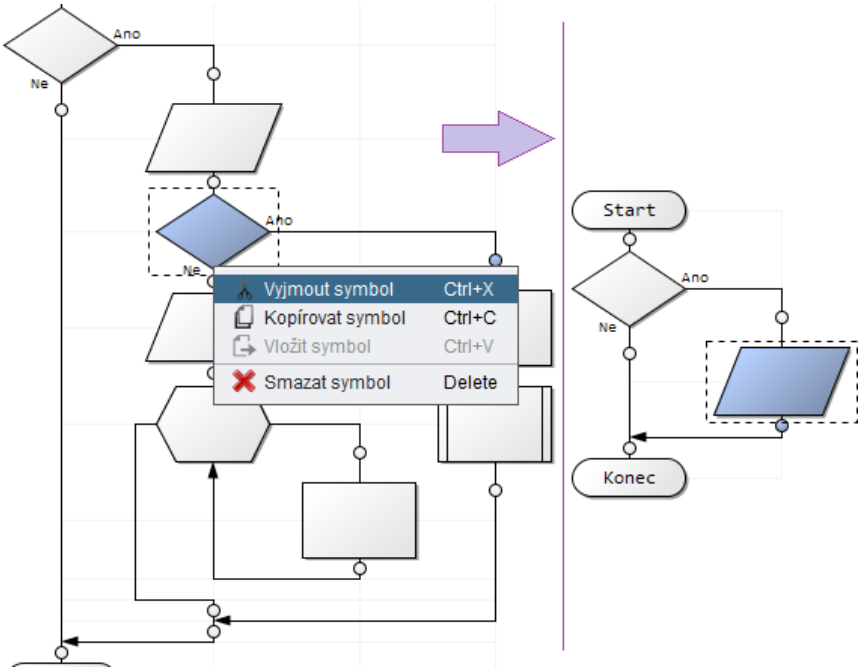
Editační režim aplikace podporuje většinu zaběhlých příkazů úprav, které bychom pro komfort editace vývojového diagramu mohli očekávat.

Menu úprav můžeme při zapnutém režimu editace vyvolat buď v hlavním menu pod položkou s názvem „*Úpravy*“ nebo vyvoláním kontextové nabídky po kliknutí na některý symbol či vkládací bod vykreslený na plátně.

3.2.1 KOPÍROVÁNÍ, VYJMUTÍ, VLOŽENÍ, MAZÁNÍ SYMBOLŮ

Při použití funkce kopírování, vyjmutí, vložení či smazání symbolu, aplikace bere zřetel na logickou strukturu našeho vývojového diagramu.

Chceme-li tedy vyjmout např. symbol podmínky, je po aktivaci této funkce vyjmut nejen samotný symbol této podmínky, ale i jeho vnitřní větve se všemi symboly vnořenými (Obr. 8).



OBR. 8 VYJMUTÍ SYMBOLU

Pro vložení symbolu pak již stačí jen označit požadovaný vkládací bod a stejným způsobem zvolit funkci „*Vložit symbol*“.

Tip:
Pro urychlení práce při úpravách vývojového diagramu použijte klávesové zkratky Ctrl+c pro kopírování, Ctrl+x pro vyjmutí, Ctrl+v pro vložení, resp. klávesu Delete pro smazání symbolu.

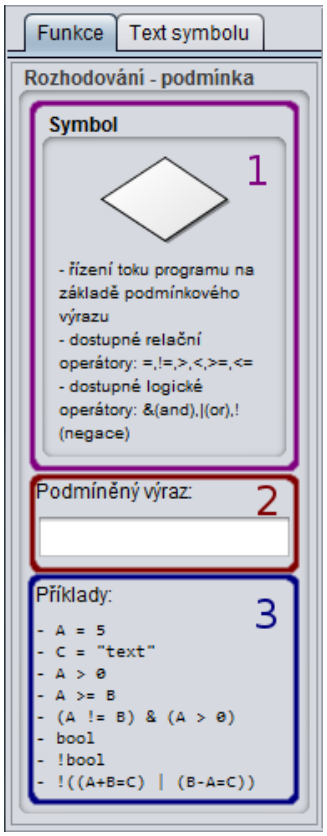
3.2.2 UNDO/REDO FUNKCE

Aplikace rovněž podporuje funkce „vrátit akci“ a „obnovit akci“. Jejich aktivaci vyvoláme buď příslušnými tlačítky v hlavní liště aplikace (Obr. 1, popisek 7) nebo v nabídce „*Úpravy*“. Můžeme využít i klávesových zkratek.

3.3 FUNKCE SYMBOLU

Záložka Funkce (Obr. 1, popisek 8) slouží k nastavení funkcionality zvoleného symbolu. Každý symbol má svou vlastní, specifickou funkcionalitu. Jednotlivým funkcím symbolů se budeme věnovat v kapitole *3.5 Dostupné symboly*.

Formulář nastavení funkce symbolu se skládá ze tří hlavních částí (Obr. 9):



OBR. 9 FORMULÁŘ FUNKCE SYMBOLU

1. Panel s grafickým znázorněním symbolu, kterého se nastavení týká spolu se stručným popisem jeho funkce
2. Komponenty pro nastavení samotné funkce symbolu
3. Příklady, popisující několik tipů ohledně možného nastavení funkce symbolu

Tip:
Zdají-li se vám textová pole pro vyplnění funkce symbolu příliš krátká, můžete celý

panel Funkce rozšířit. Pro jeho rozšíření slouží lišta mezi plátnem vývojového diagramu (Obr. 1, popisek 1) a panelem dostupných symbolů (Obr. 1, popisek 5).

Poznámka:
Nastavené funkce jsou vyhodnocovány pomocí JavaScriptu Rhino od společnosti Mozilla. Je tak možné přímo využívat balíčků, které tento interpretovaný jazyk nativně obsahuje. Velice užitečná je pak například třída Math, která nám zprostředkovává některé složitější matematické operace (reference zde).

3.3.1 Proměnné a jejich hodnoty

V aplikaci PS Diagram se proměnné, pokud již neexistují, deklarují při přiřazování hodnot automaticky. Proměnným nedefinujeme ani jejich typ, jediné, co je tedy nutné do formuláře funkce symbolu vyplnit, je název proměnné, případně její hodnota k přiřazení (dovoluje-li to daný symbol).

Chceme-li jako hodnotu proměnné přiřadit textový řetězec, je nutné ho ohraničit dvojitými uvozovkami (*"textová hodnota"*).

Poznámka:
Proměnné jsou v aplikaci PS Diagram case-sensitive – citlivé na malá a velká písmena. Znamená to, že když přiřadíme proměnné s názvem „a" hodnotu 1 a proměnné „A" hodnotu 2, ve výsledku budeme mít v paměti uloženy proměnné dvě: „a" s hodnotou 1 a „A" s hodnotou 2. Identifikátor „a" se totiž nerovná identifikátoru „A".

3.3.1.1 Pole

Výjimku v automatické deklaraci proměnných tvoří odkaz na index pole, které ještě nebylo vytvořeno. Pole je vždy nutné před vložením jeho *n*-tého prvku nejprve obsadit alespoň žádným prvkem (hodnota vyplněna jako „[]" či [5,6,7, …] popřípadě „["první", "druhý", …]").

Velikost pole není fixní, můžeme tak do pole přidávat libovolné množství dalších prvků, aniž bychom museli deklarovat jeho velikost. Prvky pole jsou indexovány od *nuly*.

Aplikace podporuje i pole vícerozměrná. Stačí jako hodnotu proměnné přiřadit pole, v němž jako jednotlivé prvky specifikujeme libovolné množství polí vnořených (hodnota vyplněna jako „[[5,3],[7,1],[3,5], …]"). Přejeme-li si pak přečíst hodnotu nultého prvku prvního pole (číslo 7), zápis bude vypadat takto: „*název_pole*[1][0]".

3.3.2 Syntaktické filtry

Textová pole pro nastavení funkcí symbolů (Obr. 9, popisek 2) obsahují tzv. syntaktické filtry. Tyto filtry nám zaručují, že do textových polí lze zadat pouze takové příkazy a identifikátory, které jsou považovány za validní.

Téměř ve všech programovacích jazycích identifikátor proměnné nesmí začínat číslicí. Této konvence se drží i PS Diagram a název proměnné tak může obsahovat pouze tyto znaky:

- číslice (v pořadí > 1. znak)
- písmena bez diakritiky
- znak podtržítka („_“)
- znak dolaru („$“)

Vždy, když by přidáním konkrétního znaku do textového pole funkce symbolu vznikla syntaktická chyba, je vložení tohoto znaku zablokováno a v levé dolní části aplikace je o této skutečnosti zobrazeno textové upozornění.

Poznámka:
Syntaktické filtry je možné v nastavení aplikace vypnout, je však silně doporučeno nechat tuto volbu povolenu.

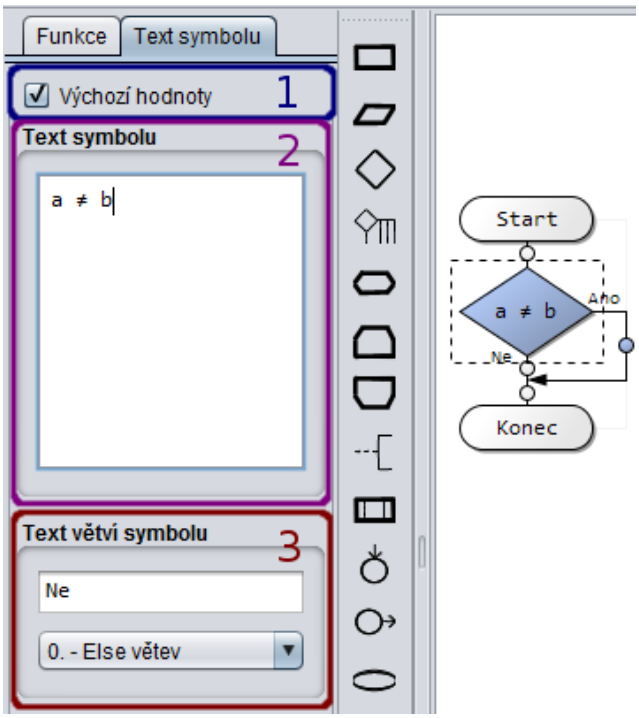
3.4 Text symbolu

Záložka Text symbolu (Obr. 1, popisek 9) slouží k nastavení textové hodnoty právě označeného symbolu. Tyto hodnoty jsou rozděleny do dvou kategorií: výchozí hodnoty a hodnoty uživatelské.

3.4.1 Výchozí hodnoty

Je-li kompletně vyplněna funkce symbolu (Obr. 9, popisek 2), textová hodnota je pro tento symbol vygenerována automaticky. Automatické generování textových hodnot bylo navrženo s ohledem na co nejvyšší výstupní srozumitelnost, některé znaky jsou proto konvertovány do čitelnější podoby. Jedná se především o tyto znaky:

- „nerovná se“
- „větší nebo rovno“, „menší nebo rovno“
- negace
- znaménko přiřazení hodnoty do proměnné



OBR. 10 FORMULÁŘ TEXTU SYMBOLU

3.4.2 UŽIVATELSKÉ HODNOTY

Text symbolu, ať už je vygenerovaný nebo není, můžeme ručně upravovat. K tomuto účelu je v tomto formuláři připravena textová oblast (Obr. 10, popisek 2). Jakékoliv upravené hodnoty, které neodpovídají případným hodnotám vygenerovaným, jsou považovány za uživatelský text.

Je-li automaticky vygenerovaný text přístupný, v horní části formuláře pro editaci textu symbolu se nám zobrazí zatrhávací políčko (Obr. 10, popisek 1) pro přepínání vygenerovaného textu, mezi textem uživatelským. Úpravou vygenerovaného textu o tuto hodnotu tedy nepřicházíme a můžeme se k ní kdykoliv navrátit.

Umožňuje-li symbol editaci textu svých výstupních větví, v dolní části formuláře je pro jejich úpravy zobrazeno textové pole s možností výběru konkrétní větve k editaci (Obr. 10, popisek 3).

Tip:
Je-li automaticky vygenerovaný text příliš dlouhý a symbol tak přesahuje přípustné rozměry, můžeme použít editaci textu symbolu pro jeho optimální odřádkování.

3.5 Dostupné symboly

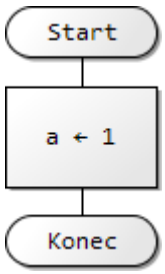
Aplikace nabízí širokou škálu použitelných symbolů. V rámci normy vývojového diagramu programu pokrývá všechny popsané symboly, krom symbolu paralelního zpracování.

Jednotlivé symboly si popíšeme v následujících podkapitolách této příručky.

3.5.1 Zpracování

Symbol zpracování nám umožňuje přiřazení dané hodnoty do cílové proměnné.

Znaménko přiřazení ve vygenerovaném textu symbolu má pro přehlednost podobu šipky.

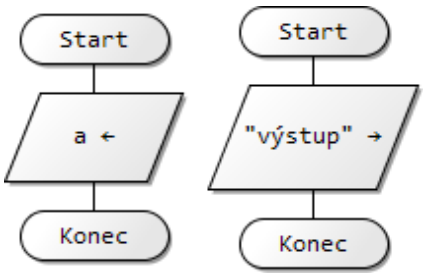


Obr. 11 Symbol zpracování

3.5.2 Vstup/Výstup

Tento symbol zajišťuje vstupně-výstupní operace pomocí dialogu (v animačním režimu aplikace). Vstupem je myšleno ruční zadání hodnoty (z klávesnice), výstupní operace zobrazí definovaný řetězec na výstupní zařízení (monitor).

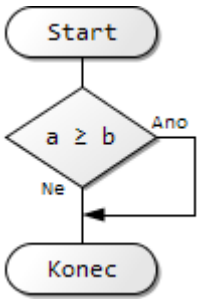
Vygenerovaný text pro rozlišení vstupu a výstupu používá opačně orientované šipky, kdy šipka směřující dovnitř symbolu značí vstupní a šipka ven výstupní operaci.



Obr. 12 Symbol vstup/výstup

3.5.3 Rozhodování – podmínka

Symbol rozhodování neboli podmínka řídí tok programu na základě zadaného podmínkového výrazu. Je-li tento výraz vyhodnocen jako logická pravda, algoritmus pokračuje výstupní větví označenou jako „Ano“. V opačném případě, kdy je výraz vyhodnocen jako logická nepravda, se tok ubírá větví pod popiskem „Ne“ (else větev).



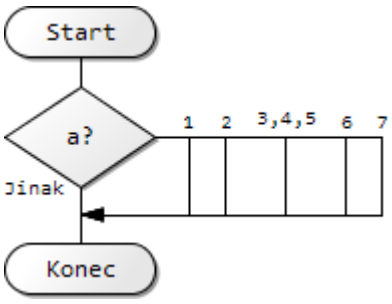
Obr. 13 Symbol rozhodování - podmínka

3.5.4 Rozhodování – vícenásobné

Aplikace implementuje i rozhodování vícenásobné, které je v programování reprezentováno příkazem „switch“ či „case“.

Tento příkaz vyhodnotí cílovou proměnnou vůči zadaným konstantám. Konstanty mohou představovat číselné hodnoty nebo textové hodnoty ohraničené uvozovkami. Konstant je možno jedné větvi určit více, v takovém případě je nutné je oddělit čárkami.

Odpovídá-li cílová proměnná některé ze zadaných konstant, tok programu bude nasměrován k té větvi, které konstanta náleží. Neodpovídá-li cílová proměnná žádné z konstant, tok programu bude pokračovat větví označenou jako „Jinak“ (default větev).



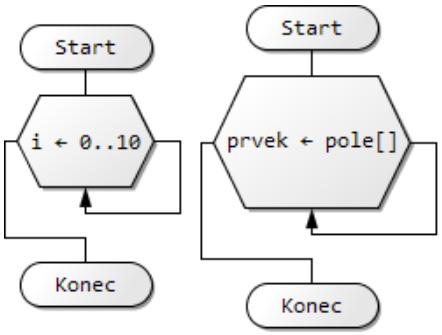
Obr. 14 Symbol rozhodování - vícenásobné

3.5.5 Příprava – cyklus pevným počtem opakování

Symbol přípravy v aplikaci představuje dva specifické cykly s pevným počtem opakování.

První cyklus známe z programování jako klasický „for cyklus“, kdy je proměnné cyklu přiřazována číselná hodnota z definovaného „počítadla“ (Obr. 15, vlevo).

Druhý typ cyklu je z programování znám jako „for-each cyklus“, kdy jsou proměnné cyklu postupně přiřazeny všechny prvky cílového pole (Obr. 15, vpravo).

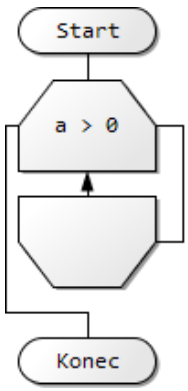


Obr. 15 Symbol přípravy- cyklus s pevným počtem opakování

3.5.6 Cyklus s podmínkou na začátku

Symbol cyklu s podmínkou na začátku je definován mezními značkami. Horní mez cyklu řídí tok programu podobně jako symbol rozhodování.

Je-li podmínkový výraz symbolu vyhodnocen jako logická pravda, algoritmus pokračuje větví těla cyklu. Pokud je dosaženo dolní meze cyklu, tok programu je přesunut zpět k jeho horní hranici, aby zde byl znovu vyhodnocen podmínkový výraz. Je-li podmínkový výraz vyhodnocen jako nepravda, cyklus je přerušen a následuje příkaz pod tímto cyklem.

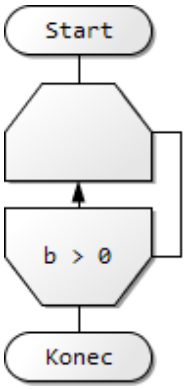


Obr. 16 Symbol cyklus s podmínkou na začátku

Poznámka:
Všimněte si, že popsané chování cyklu je patrné i ze způsobu, jak jsou hranice cyklu propojeny spojnicemi.

3.5.7 Cyklus s podmínkou na konci

Symbol cyklu s podmínkou na konci je rovněž definován mezními značkami. Na rozdíl od cyklu s podmínkou na začátku, je však v tomto cyklu řídící podmínkový výraz umístěn v jeho dolní hranici. Toto umístění podmínkového výrazu nám zaručí, že tělo cyklu bude vykonáno minimálně jednou.



Obr. 17 Symbol cyklus s podmínkou na konci

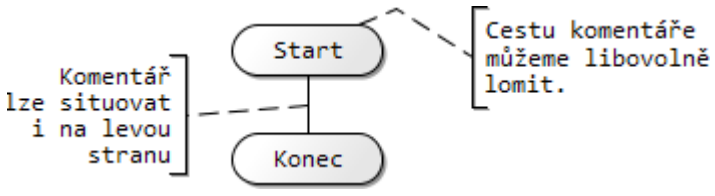
Poznámka:
Všimněte rozdílů ve spojnicovém propojení hranic cyklů s podmínkou nahoře a s podmínkou dole.

3.5.8 Komentář

Do vývojového diagramu lze umístit i symboly komentářů.

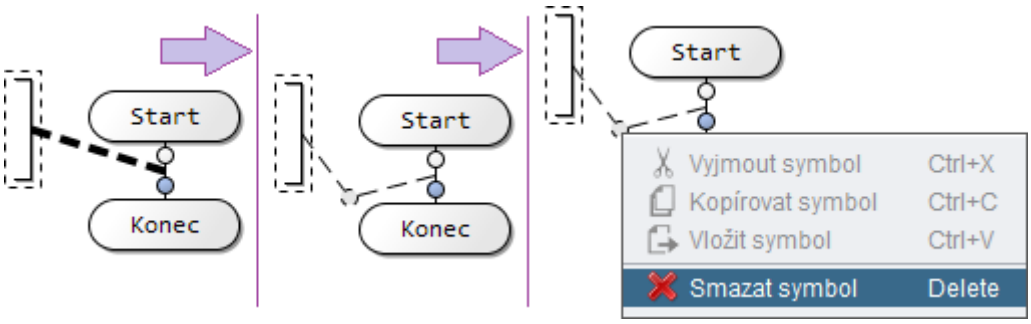
Komentář je obecně možné přiřadit buď na vlastní pozici uvnitř toku vývojového diagramu, nebo jej přiřadit ke kterémukoliv jinému symbolu.

Symbolem komentáře lze v editačním režimu aplikace dále manipulovat. Komentář můžeme umístit na libovolnou pozici na plátně, přičemž je nám umožněno i modulování jeho spojnice.



Obr. 18 Symbol komentář

Úprava spojnice komentáře (Obr. 19) je realizována intuitivní cestou, kdy stačí myší najet nad požadovanou spojnici k editaci. Tato spojnice je pak zvýrazněna, a nám již stačí metodou „táhni a pusť“ (drag&drop) spojnici zlomit podle našich představ. Chceme-li později tento zlom upravit, stačí opět myší najet nad tento spoj. Podobným způsobem je možné spoj odstranit, kdy po najetí myši nad zlom spojnice vyvoláme kontextovou nabídku, v níž vybereme možnost jeho odstranění.

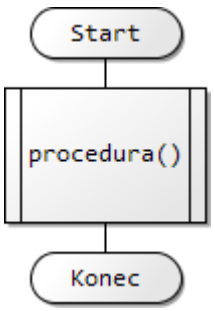


OBR. 19 ÚPRAVA SPOJNICE KOMENTÁŘE

3.5.9 PŘEDDEFINOVANÉ ZPRACOVÁNÍ

Tento symbol představuje pojmenované zpracování, které se skládá z jedné nebo více operací, jež jsou specifikovány jinde. Jedná se tedy např. o volání funkcí či procedur.

Aplikace PS Diagram (zatím) procedury a funkce nepodporuje, tento symbol tedy nedisponuje nastavením své funkce.



OBR. 20 SYMBOL PŘEDDEFINOVANÉ ZPRACOVÁNÍ

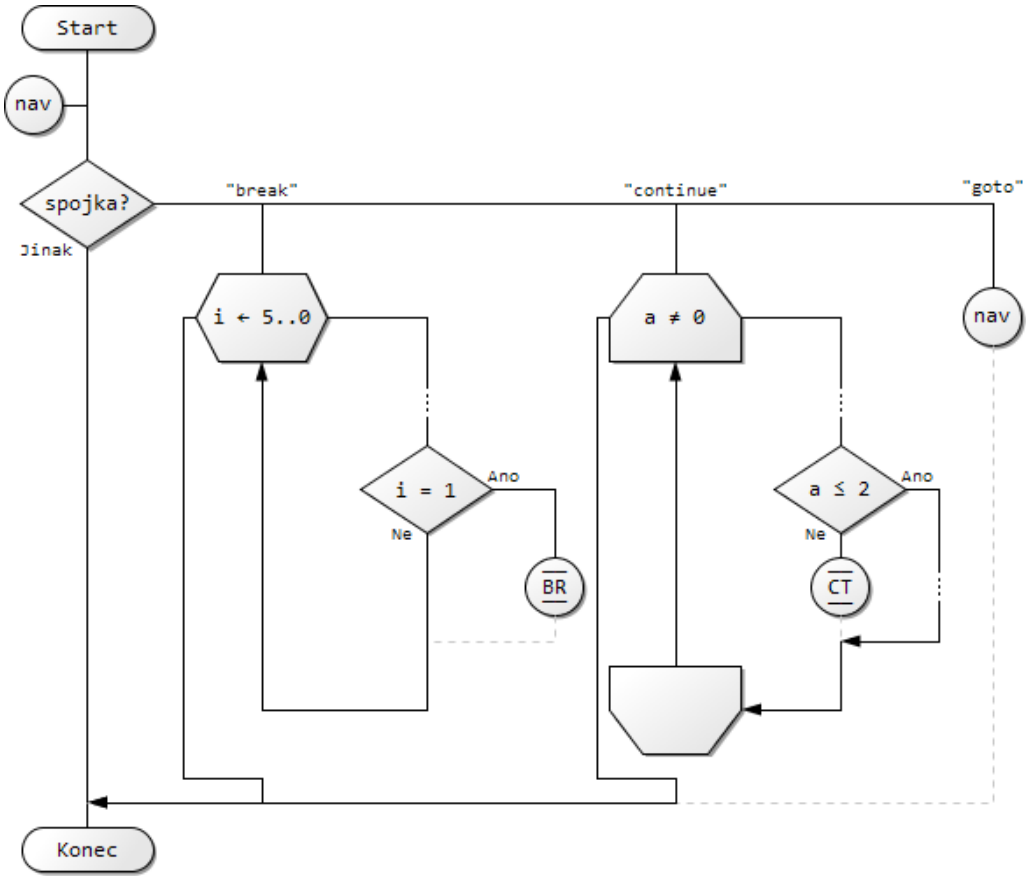
3.5.10 SPOJKA – BREAK, CONTINUE, GOTO

Tento symbol může v aplikaci nabývat třech různých funkcí.

První funkce nese název „break“, stejnojmenného příkazu známého z programování. Příkaz break slouží k okamžitému opuštění cyklu tak, že je vykonán následující příkaz za cyklem. Symbolu je zároveň vygenerován automatický text.

Druhá funkce symbolu pod názvem „continue“ opět vykonává příkaz známý z programování, kdy je aktuální smyčka cyklu přerušena a tok programu je předán řídícímu symbolu cyklu. Symbolu je rovněž vygenerován automatický text.

Třetí, poslední funkcí tohoto symbolu je příkaz s názvem „goto“. Tento příkaz způsobí, že je tok programu přesměrován na místo označené symbolem návěští (kapitola 3.5.11) stejného identifikátoru. Je tedy nutné vyplnit text symbolu, který bude zároveň sloužit jako identifikátor.



OBR. 21 SYMBOL SPOJKA

Poznámka:
Všimněte si, že spojnice vedoucí od tohoto symbolu jsou zobrazeny čárkovaně. Takto jsou označena místa, kde se tok programu nemůže nikdy vyskytnout.

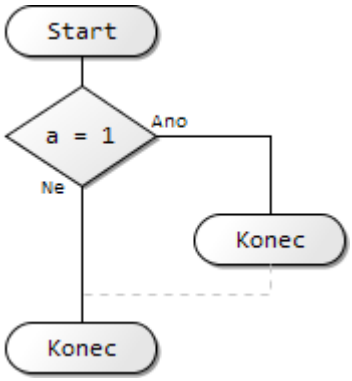
3.5.11 SPOJKA – NÁVĚŠTÍ

Symbol spojky v podobě návěští, představuje cíl pro přesměrování toku programu, je-li vstoupeno do symbolu s příkazem „goto“ (kapitola 3.5.10). Grafickou reprezentaci tohoto symbolu lze zpozorovat na Obr. 21 (druhý symbol shora).

3.5.12 MEZNÍ ZNAČKA

Mezní značka představuje začátek nebo konec programu. Každý algoritmus by měl být ohraničen právě těmito značkami, proto jsou v aplikaci automatickou součástí každého nově vytvořeného vývojového diagramu.

Tento symbol lze do vývojového diagramu přidat i vícekrát, v takovém případě vždy představuje konec programu.

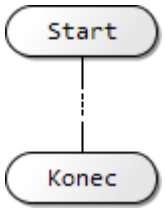


OBR. 22 SYMBOL MEZNÍ ZNAČKA

Poznámka:
Všimněte si, že spojnice vedoucí od tohoto symbolu jsou zobrazeny čárkovaně. Takto jsou označena místa, kde se tok programu nemůže nikdy vyskytnout.

3.5.13 VÝPUSTKA

Výpustku využijeme tehdy, chceme-li znázornit, že dochází k vypuštění symbolu nebo skupiny symbolů, kde ani druh ani jejich počet nemusí být definován.



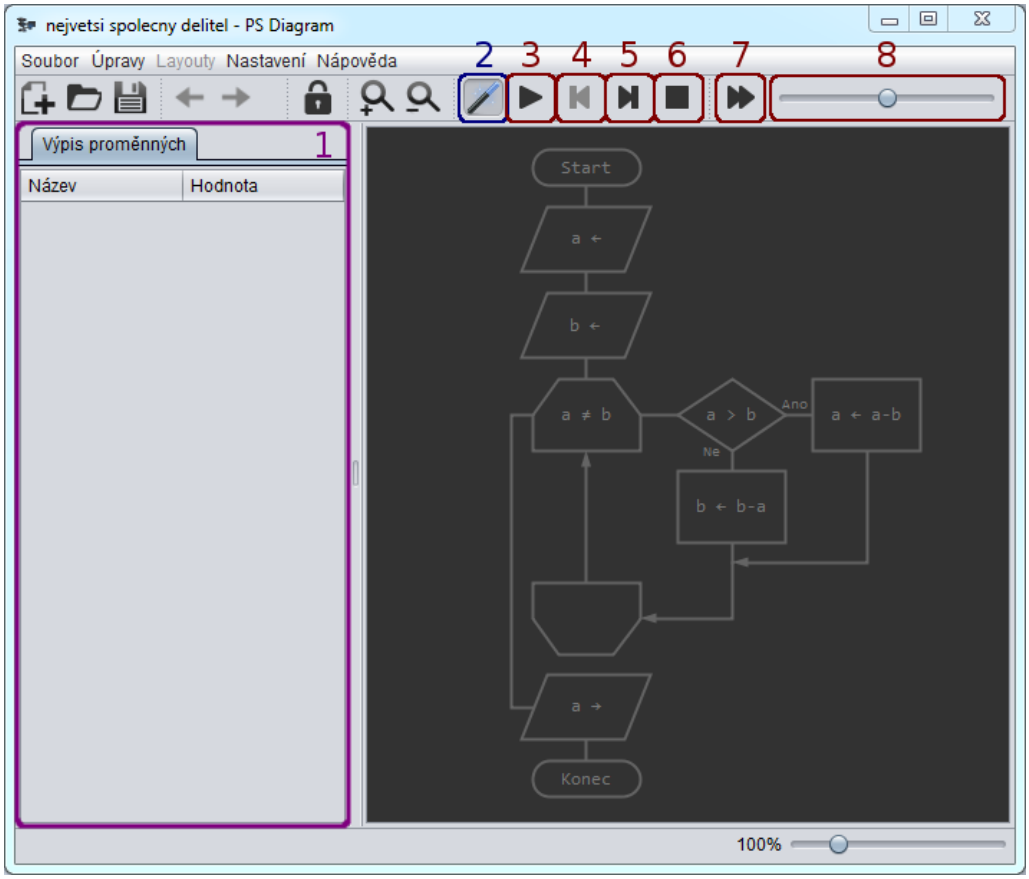
OBR. 23 SYMBOL VÝPUSTKA

4 Animační režim

Do animačního režimu aplikace vstoupíme sepnutím tlačítka magické hůlky (Obr. 24, popisek 2) v hlavním panelu aplikace.

Animační režim zprostředkovává vizualizaci algoritmického průběhu námi vytvořeného vývojového diagramu. Díky němu je pak možné pozorovat, jak se např. mění parametry cyklu, jakým způsobem algoritmus rozhoduje a jak se mění proměnné.

Vstupem do tohoto režimu se nám zpřístupní panel pro jeho ovládání (Obr. 24, popisky 3-8) a panel výpisu proměnných (Obr. 24, popisek 1). Panel s výpisem proměnných se nám bude hodit po spuštění samotného průchodu, kdy v něm budou přehledně zobrazeny aktuálně existující proměnné (kapitola 4.1.2).



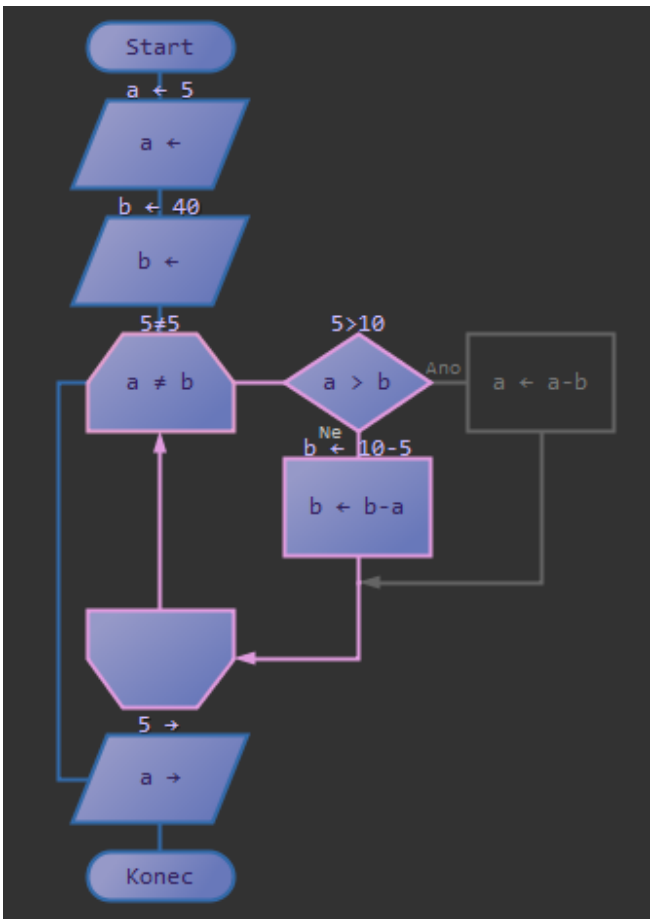
Obr. 24 Animační režim

Poznámka:
Animační režim spolupracuje s režimem náhledu, s vývojovým diagramem tak lze nadále manipulovat obvyklým způsobem.

4.1 Vizualizace průchodu vývojovým diagramem

Jistě jste postřehli, že vývojový diagram je v animačním režimu vykreslen jiným způsobem, než je tomu u ostatních režimů aplikace. Symboly s jejich spojnicemi jsou vykresleny prozatím nenápadně, to se ale mění po jejich aktivaci.

Vykoná-li symbol svou funkci, změní svou barvu tak, že lze tuto skutečnost jasně rozeznat. Uživatel tak zřetelně pozná, kudy se algoritmus vytvořeného vývojového diagramu ubírá či ubíral (Obr. 25).



Obr. 25 Vizualizace průchodu vývojovým diagramem

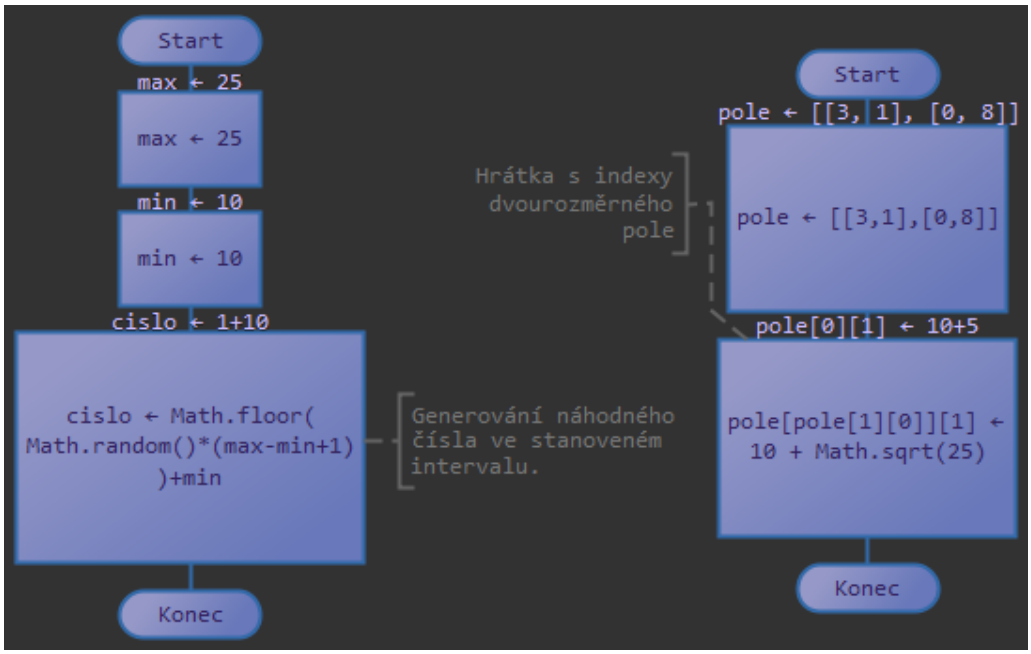
Z Obr. 25 můžeme navíc vypozorovat, že některé symboly provedly svou funkci více než jednou. Vykoná-li totiž symbol svou přiřazenou funkci, je barva jeho okraje zbarvena do odlišné barvy, než tomu bylo předtím. Stejně reagují i spojnice mezi symboly. Tato barva je upravována až do limitní bílé, která ve výchozím stavu odpovídá 25. aktivaci.

Tip:
Chcete-li s procházením vývojového diagramu začít od začátku, použijte tlačítko s popiskem „*Reset*“ (Obr. 24, popisek 6).

4.1.1 Mezivýpočty funkcí symbolů

Na Obr. 25 je možné pozorovat i zobrazení mezivýpočtů funkcí symbolů. Tyto mezivýpočty dávají uživateli jasnou představu o tom, jak bylo výsledku funkce daného symbolu dosaženo.

Mezivýpočty zobrazují vždy výsledky dílčích příkazů či hodnoty daných proměnných. Tento způsob zobrazení nám často může rozuzlit i jinak komplikované příkazové struktury (Obr. 26).



Obr. 26 Mezivýpočty složitějších struktur

4.1.2 Výpis proměnných

V panelu s výpisem proměnných se zobrazují aktuálně existující proměnné a jejich hodnoty (Obr. 27).

Proměnná, která byla posledním vykonáním funkce symbolu přidána nebo upravena, je vyobrazena tučným písmem. Je tak pro uživatele snazší se ve výpisu orientovat. Proměnné jsou navíc vypsány v jejich abecedním pořadí.

Speciálnímu nakládání se těší proměnné polí, kdy jsou jejich prvky na jednotlivých indexech zobrazeny ve stromové struktuře.

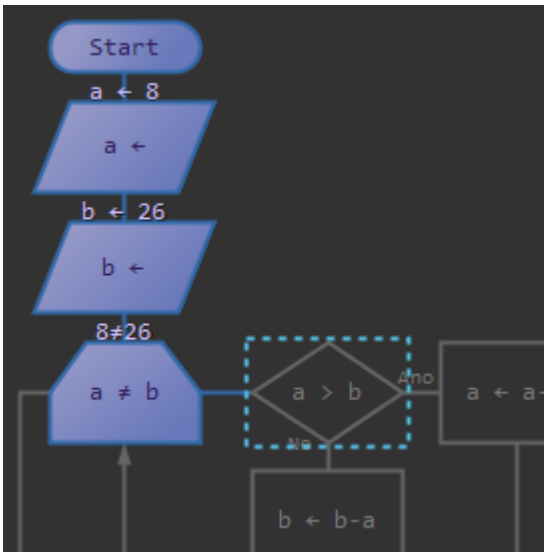
Název	Hodnota
cislo	5
novacek	**1**
▼ pole	[5, 2, 8]
[0]	5
[1]	2
[2]	8
prom	"moje hodnota"

OBR. 27 VÝPIS PROMĚNNÝCH

4.2 KROKOVÁNÍ

Procházení algoritmu vývojového diagramu můžeme ovládat několika způsoby. Jeden z těchto způsobů je právě funkce krokování.

Stiskem tlačítka krok vpřed (Obr. 24, popisek 5) aktivujeme symbol čekající na zpracování. Symbol čekající na zpracování je vždy modře ohraničen a my se tak vždy můžeme připravit na pozorování výsledku vykonání jeho funkce (Obr. 28).



OBR. 28 SYMBOL ČEKAJÍCÍ NA ZPRACOVÁNÍ

Aplikace PS Diagram disponuje i nadstandardní funkcí kroku zpět (Obr. 24, popisek 4), kdy je celý algoritmus po jeho (částečném) vykonání možné vrátit do požadovaného dřívějšího stavu.

Tip:
Pro urychlení krokování vývojovým diagramem využijte klávesových zkratek popsaných v nápovědách konkrétních tlačítek (zobrazí se po najetí myši).

4.3 Funkce spustit rychle

Dalším ze způsobů procházení algoritmu vývojového diagramu, je funkce jeho rychlého spuštění.

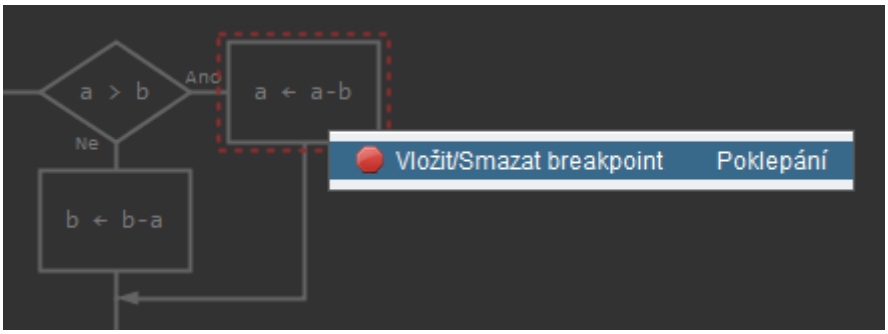
Po stisknutí odpovídajícího tlačítka (Obr. 24, popisek 7) je celý diagram procházen automaticky co nejvyšší možnou rychlostí. Tato funkce tedy simuluje chování skutečné aplikace, kdy není umožněno její krokování a konečný výsledek je zobrazen téměř okamžitě.

V případě, že bychom chtěli tento spuštěný proces pozastavit, můžeme tak učinit buď tlačítkem „*Pozastavit animaci*“ (Obr. 30, popisek 1) nebo využít funkce breakpoint, popsanou v následující podkapitole.

4.3.1 Funkce breakpoint

Breakpoint neboli zarážka je označení místa, kde dojde k zastavení spuštěného rychlého procházení diagramu. Používá se pro automatické pozastavení procházení v místě, kde chceme provést algoritmickou inspekci.

Pro vložení breakpointu vyvolejte v animačním režimu kontextovou nabídku nad některým z požadovaných symbolů, který bude touto zarážkou obklopen (Obr. 29). Breakpointů je možné definovat libovolné množství.



Obr. 29 Vložení breakpointu

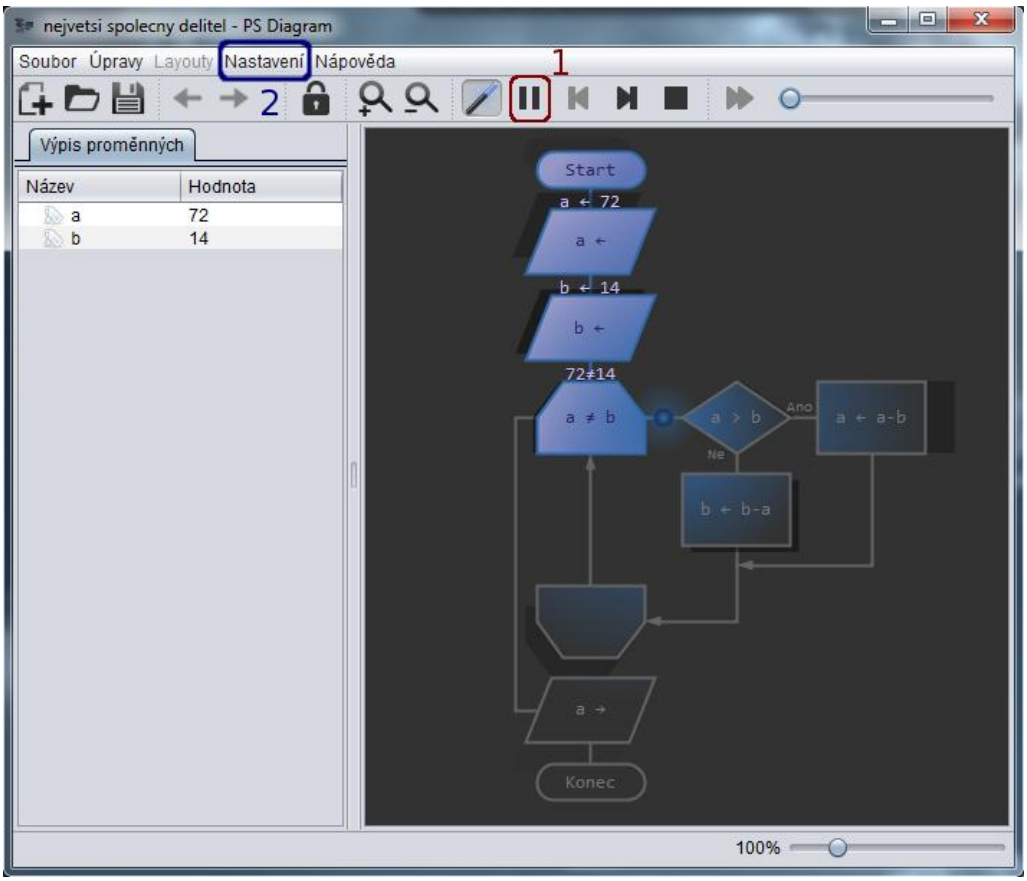
Tip:
Pro rychlé vložení breakpointu jednoduše poklepejte na požadovaný symbol.

4.4 ANIMACE

Animace je nejatraktivnější způsob průchodu vývojovým diagramem.

Aktivujeme-li ji tlačítkem s popiskem „*Spustit animaci*“, v místě symbolu aktuálně čekajícího na zpracování se zobrazí tzv. *průchodová kulička* (Obr. 30). Tato zářící kulička nás pak plynule provází celým průchodem, dávajíce nám jasně najevo, kudy se algoritmus našeho vývojového diagramu ubírá.

Animaci průchodu vývojovým diagramem můžeme kdykoliv pozastavit tlačítkem s popiskem „*Pozastavit animaci*“ (Obr. 30, popisek 1). Animaci lze zároveň popohnat, a to stiskem tlačítka pro krok vpřed (Obr. 24, popisek 5). Jak již jeho název napovídá, způsobí okamžitý přesun průchodové kuličky k symbolu, jež čeká na zpracování.



OBR. 30 ANIMACE

Tip:
Pracujeme-li s animací, je obzvlášť výhodné použití klávesových zkratek pro její ovládání. Využijte klávesu mezerník k jejímu spuštění/pozastavení.

4.4.1 Pohyb průběhové kuličky

Průběhová kulička se nepohybuje konstantní rychlostí. Její rychlost se mění v závislosti na prošlé trase vůči její délce a je tak docíleno vyšší záživnosti celé animace.

Rychlost, kterou se průběhová kulička pohybuje, je možno dále dynamicky upravovat. K tomuto účelu použijte posuvník v pravém horním rohu aplikace (Obr. 24, popisek 8).

4.5 Možnosti nastavení

Animační režim aplikace disponuje několika nastavitelnými položkami, které mají vliv na jeho chování. Tyto položky lze upravit v nastavení aplikace (Obr. 30, popisek 2) pod záložkou „*Animační režim*“.

V nastavení animačního režimu pak lze např. změnit přístup k proměnným, určit dosah záře průběhové kuličky či ji zcela vypnout. Nastavit můžeme i počet snímků za sekundu, kterým se animace bude vykreslovat.

4.5.1 Přístup k proměnným

Aplikace umožňuje k proměnným přistupovat dvěma způsoby, jež si popíšeme v následujících podkapitolách.

4.5.1.1 Globální přístup

Při globálním přístupu vytvořené proměnné existují globálně - nezanikají a po jejich vytvoření je s nimi možné pracovat kdykoliv a kdekoliv v rámci celého vývojového diagramu.

Tento přístup je znám například z programovacího jazyka Pascal, kdy jsou proměnné deklarovány v hlavičce zdrojového kódu.

4.5.1.2 Blokový přístup

Při blokovém přístupu vytvořené proměnné existují jen v rámci svého bloku (větve symbolu) a ve větvích do něj vnořených. Jakmile se tok programu ocitne mimo tento blok (nebo jeho vnořené bloky), proměnná zaniká.

Tento přístup k proměnným známe z většiny moderních (objektových) programovacích jazyků, deklarujeme-li proměnnou uvnitř těla metod (funkcí, procedur).

5 UKLÁDÁNÍ A OTEVÍRÁNÍ

Vytvořené vývojové diagramy lze standardně ukládat a otevírat ve formátu XML. K tomuto účelu můžeme využít tlačítek umístěných v menu „*Soubor*“ v horním levém rohu aplikace nebo tlačítek v hlavní programové liště pod touto nabídkou.

Využít lze i zaběhlých klávesových zkratek nebo technologie „táhni a pusť“ (drag&drop). V tomto případě stačí soubor, který chceme otevřít, myší přetáhnout do aplikačního plátna (Obr. 1, popisek 1) a vývojový diagram je pak automaticky otevřen.

6 GRAFICKÝ EXPORT

Grafickou podobu vývojového diagramu lze kdykoliv exportovat do některého z podporovaných formátů.

V možnostech nastavení aplikace navíc nalezneme volby týkající se transparentnosti a velikosti okraje výstupního vývojového diagramu.

6.1 OBRÁZEK

Jednou z možností exportu vývojového diagramu je právě export do bitmapového obrázku. Formulář pro tuto funkci vyvoláme stiskem tlačítka „*Export do obrázku*“, které se nalézá v menu „*Soubor*“ v levém horním rohu aplikace.

6.2 PDF

Další možností, jak vývojový diagram exportovat, je jeho uložení do formátu PDF. V tomto případě je vývojový diagram uložen vektorově a my tak na rozdíl od grafiky bitmapové (export do obrázku), při jeho přibližování nepřicházíme o jeho obrazovou kvalitu.

7 IMPORT ZE ZDROJOVÉHO KÓDU

Zajímavou funkcí, kterou PS Diagram disponuje, je možnost reverzního inženýrství.

Tato funkcionalita nám dovoluje vývojový diagram vygenerovat z vloženého zdrojového kódu některého z podporovaných programovacích jazyků[1].

Formulář pro tuto funkci vyvoláme tlačítkem „*Import ze zdroj. kódu*“, umístěném v menu „*Soubor*“ v levém horním rohu aplikace.

Tip:
Import ze zdrojového kódu se stává vysoce užitečným zejména v případech, kdy bychom rádi prezentovali algoritmus, který již máme napsán v některém z podporovaných programovacích jazyků.

Poznámka:
Podporován je prozatím jen programovací jazyk Pascal.

8 EXPORT DO ZDROJOVÉHO KÓDU

Další nadstandardní funkcí, kterou PS Diagram disponuje, je možnost vývojový diagram naopak do zdrojového kódu exportovat.

Zhotovený vývojový diagram tak lze okamžitě převést do funkční aplikace některého z podporovaných programovacích jazyků[1].

Poznámka:
Podporován je prozatím jen programovací jazyk Pascal.

[1] Správnost vygenerovaného vývojového diagramu (nebo zdrojového kódu) není vždy stoprocentní, je proto doporučeno výstup ještě překontrolovat, případně poopravit.

PŘÍLOHA B

DOTAZNÍK
VÝZKUM – VÝUKA ALGORITMŮ

Dotazník lze taktéž nalézt na přiloženém CD

Výzkum - výuka algoritmů

Předmět mé bakalářské práce, je vytvořit aplikaci umožňující simulaci průchodu programu vývojovým diagramem pro účel výuky algoritmizace. Sestavení vývojového diagramu bude umožněno dvěma způsoby. Uživatel si bude moci vývojový diagram sestavit sám, pomocí vestavěného editoru, nebo využít funkci automatického vygenerování diagramu z vloženého zdrojového kódu programovacího jazyka. Podporovány budou základní příkazy strukturovaného programování, jako např. proměnné, pole, rozhodování, cykly, vstupy a výstupy. Aplikace bude následně umožňovat simulaci průchodu takto vytvořeným vývojovým diagramem tak, že bude zřejmé, jak jím program prochází, jak se mění parametry cyklu, jakým způsobem rozhoduje a jak se mění proměnné. Výsledný diagram bude možné uložit opět ve formě zdrojového kódu.
Pro základní představu ohledně vizualizace průchodu algoritmem, přikládám jednoduchý obrázek.
https://lh4.googleusercontent.com/_i9OaTrLscNs/TbaARzjC4NI/AAAAAAAAAE0/0RkgXe-nr4Y/s800/simulace_pruchodu.png

Otázky jsou nepovinné, můžete tedy vyplnit jen ty, na které opravdu znáte odpověď. Neznáte-li na některou otázku odpověď, prosím o přeposlání formuláře odpovídající osobě, která by tyto odpovědi doplnila.

Děkuji,
Miroslav Bartyzal (miroslavbartyzal@centrum.cz)

Jaký software je pro účel vyobrazení algoritmů (vývojových diagramů) používán na Vaší škole?

může se jednat o pouhé zobrazení vývojových diagramů, případně i o pokročilejší software s procházením

- ☐ žádný (nepoužíváme vývojové diagramy)
- ☐ žádný (používáme tužku a papír s tabulí)
- ☐ Malování (součást OS MS Windows)
- ☐ MS Word (funkce Tvary)
- ☐ Visual Paradigm
- ☐ Jiné:

Jaké programovací jazyky se ve Vaší škole vyučují?

- [ ] Pascal
- [ ] Delphi
- [ ] C#
- [ ] C
- [ ] C++
- [ ] Java
- [ ] VB 6
- [ ] VB.NET
- [ ] PHP
- [ ] Jiné:

Měli by jste zájem o aplikaci, která by byla speciálně určena pro tvorbu vývojových diagramů a zároveň poskytovala možnost vizualizace jejich průchodu?

jinými slovy o předmět mé bakalářské práce

- ○ Ano
- ○ Ne
- ○ Jiné:

Název a sídlo Vaší školy?

Pokud chcete být kontaktován po dokončení mé bakalářské práce s nabídkou ke stažení hotové aplikace, zadejte prosím emailovou adresu. Na tuto adresu bude nabídka zaslána.

Odeslat