

# **Knihovna pro vizualizaci grafů v prostředí .NET**

The Library for a Graph Visualization within .NET Framework

Bc. Jaromír Uhřík

---

Diplomová práce  
2013



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

akademický rok: 2012/2013

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jaromír Uhřík**  
Osobní číslo: **A11415**  
Studijní program: **N3902 Inženýrská informatika**  
Studijní obor: **Informační technologie**  
Forma studia: **prezenční**

Téma práce: **Knihovna pro vizualizaci grafů v prostředí .NET**

Zásady pro vypracování:

1. Zpracujte stručný přehled řešené problematiky včetně v současnosti používaných řešení.
2. Naprogramujte knihovnu pro vizualizaci grafů v prostředí .NET Framework. Knihovna bude schopna dynamického vykreslování ze zadaného XML formátu.
3. Sestavte přehledný manuál pro práci s výslednou knihovnou.
4. Implementujte editační nástroj využívající vytvořenou knihovnu.
5. Vypracujte sadu ukázek praktického použití knihovny.
6. Využijte tuto knihovnu pro vizualizaci algoritmu Analytického Programování.
7. Výsledek práce vhodně prezentujte v prostředí Internetu.

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. CLARK, John. A first look at graph theory. Singapore: World Scientific, 1996, xiv, 330 s. ISBN 98-102-0490-6.
2. ESPOSITO, Dino. XML: efektivní programování pro .NET. 1. vyd. Překlad Jaroslav Černý. Praha: Grada, 2004, 596 s. ISBN 80-247-0775-6.
3. LIBERTY, Jesse. Programming .NET 3.5. Sebastopol: O'Reilly, 2008, xvii, 455 s. ISBN 978-0-596-52756-3.
4. NASH-WILLIAMS, W.T. Tutte; foreword by Crispin St. J.A. Graph theory. Transferred to digital print. Cambridge: Cambridge University Press, 2001. ISBN 05-217-9489-7.
5. C-Sharp Graphics Programming. Hoboken: John Wiley, 2010. ISBN 978-111-8035-504.

Vedoucí diplomové práce:

**Ing. Bc. Pavel Vařacha, Ph.D.**

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

**22. února 2013**

Termín odevzdání diplomové práce:

**22. května 2013**

Ve Zlíně dne 22. února 2013

prof. Ing. Vladimír Vašek, CSc.

*děkan*



doc. Mgr. Roman Jašek, Ph.D.

*ředitel ústavu*

## **ABSTRAKT**

Diplomová práce se zabývá vizualizací grafů v prostředí .NET. Teoretická část uvádí stručný přehled existujících řešení, související teorii a charakterizuje technologie, které jsou dále využity při implementaci reálně použitelné knihovny a z ní vycházející editační aplikace. Praktická část popisuje implementaci některých vybraných částí knihovny a provádí seznámení s možnostmi, které jsou výsledkem její implementace.

Klíčová slova: vizualizace grafů, .NET, dynamická knihovna, editor grafů

## **ABSTRACT**

The thesis deals with graph visualization in .NET Framework. In the theoretical part, there is brief review of existing programs followed by related theory and description of tools and technologies. The knowledge base described in these chapters is utilized for developing applicable library and editable tool based on this library. The practical part reveals punctual implementation of selected parts of the library and possible communication ways resulting from its implementation.

Keywords: graph visualization, .NET, dynamic library, graph editor

### **Poděkování**

Tímto bych rád poděkoval svému vedoucímu panu Ing. Bc. Pavlu Vařachovi, Ph.D. za vedení diplomové práce a čas strávený při konzultacích.

**Prohlašuji, že**

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

**Prohlašuji,**

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

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>10</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>11</b>
<b>1 ÚVOD DO GRAFŮ</b> .....	<b>12</b>
1.1 HRANA .....	12
1.1.1 Základní rozdělení hran.....	13
1.2 VRCHOL .....	13
1.3 ORIENTACE .....	14
1.4 CESTA .....	14
1.5 ÚPLNÝ GRAF.....	15
1.6 STROM.....	15
1.7 REPREZENTACE GRAFŮ.....	16
1.7.1 Matice sousednosti .....	16
1.7.2 Matice incidence .....	16
1.7.3 Grafická reprezentace.....	17
<b>2 EXISTUJÍCÍ SOFTWAREOVÁ ŘEŠENÍ</b> .....	<b>18</b>
2.1 YFILES .NET .....	18
2.1.1 Varianty produktu.....	18
2.1.2 Ceny .....	19
2.1.3 Výhody a nevýhody.....	20
2.2 GEPHI .....	21
<b>3 DEFINICE UŽIVATELSKÝCH POŽADAVKŮ</b> .....	<b>23</b>
<b>4 POPIS POUŽITÝCH TECHNOLOGIÍ</b> .....	<b>24</b>
4.1 MICROSOFT .NET FRAMEWORK.....	24
4.1.1 Jazyky.....	25
4.1.2 Verze .....	25
4.1.3 Výkon.....	26
4.1.4 Bezpečnost .....	26
4.2 MICROSOFT VISUAL STUDIO.....	26
<b>5 FYZIKÁLNÍ ZÁKLADY</b> .....	<b>29</b>
5.1 PRUŽNOST .....	29
5.2 HOOKŮV ZÁKON.....	29
5.3 ELASTICKÁ HYSTEREZE .....	29
<b>II PRAKTICKÁ ČÁST</b> .....	<b>30</b>
<b>6 ZÁKLADNÍ NÁVRH</b> .....	<b>31</b>
<b>7 KNIHOVNA PRO VIZUALIZACI GRAFŮ</b> .....	<b>32</b>

7.1	VNITŘNÍ REPREZENTACE VRCHOLŮ .....	32
7.2	VNITŘNÍ REPREZENTACE HRAN.....	33
7.2.1	Typy orientací hran.....	33
7.3	UCHOVÁVÁNÍ POZIC .....	34
7.4	MOŽNOSTI VSTUPU .....	34
7.4.1	Textový řetězec .....	34
7.4.2	XML vstup .....	36
7.5	STYLOVÁNÍ .....	38
7.5.1	Dynamické dědění grafických tříd .....	40
7.5.2	Podporované vlastnosti .....	41
7.6	ORGANIZAČNÍ ALGORITMUS .....	42
7.6.1	Princip organizace .....	43
7.7	KRUHOVÉ USPOŘÁDÁNÍ.....	45
<b>8</b>	<b>MANUÁL PRO PRÁCI S KNIHOVNOU .....</b>	<b>47</b>
8.1	PŘIDÁNÍ KNIHOVNY DO PROJEKTU .....	47
8.2	DEKLARACE PROMĚNNÉ .....	47
8.3	INICIALIZACE.....	47
8.4	PLOCHA PRO VYKRESLOVÁNÍ .....	48
8.5	NASTAVENÍ GRAFU .....	48
8.5.1	Určení středu .....	48
8.5.2	Délka hran .....	49
8.5.3	Gravitace .....	49
8.5.4	Změna měřítka.....	50
8.6	REŽIM VYKRESLOVÁNÍ .....	50
8.7	NAČTENÍ GRAFU .....	50
8.7.1	Načtení z textové definice .....	50
8.7.2	Načtení dle XML.....	51
8.8	PŘIDÁNÍ VRCHOLU.....	51
8.9	PŘIDÁNÍ HRANY.....	52
8.10	DALŠÍ DOSTUPNÉ MOŽNOSTI.....	52
<b>9</b>	<b>EDITAČNÍ NÁSTROJ .....</b>	<b>55</b>
9.1	IMPLEMENTAČNÍ NÁVRH.....	55
9.2	PRACOVNÍ PLOCHA.....	56
9.2.1	Editační okno .....	56
9.2.2	Editor vlastnosti .....	57
<b>10</b>	<b>PŘÍKLADY POUŽITÍ.....</b>	<b>59</b>

---

10.1	VYTVORENÍ GRAFU Z XML .....	59
10.2	PŘÍKLAD POUŽITÍ KNIHOVNY V JAZYCE C# .....	61
10.3	PŘÍKLAD POUŽITÍ KNIHOVNY V JAZYCE VISUAL BASIC .....	62
10.4	INTERAKTIVNÍ ZNÁZORNĚNÍ TOPOLOGIE SÍTĚ .....	64
<b>11</b>	<b>VYUŽITÍ KNIHOVNY PRO VIZUALIZACI ALGORITMU ANALYTICKÉHO PROGRAMOVÁNÍ .....</b>	<b>66</b>
<b>12</b>	<b>PREZENTACE V PROSTŘEDÍ INTERNETU .....</b>	<b>69</b>
	<b>ZÁVĚR .....</b>	<b>70</b>
	<b>ZÁVĚR V ANGLIČTINĚ .....</b>	<b>72</b>
	<b>SEZNAM POUŽITÉ LITERATURY .....</b>	<b>74</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK .....</b>	<b>76</b>
	<b>SEZNAM OBRÁZKŮ .....</b>	<b>77</b>
	<b>SEZNAM TABULEK .....</b>	<b>78</b>
	<b>SEZNAM PŘÍLOH .....</b>	<b>79</b>

## ÚVOD

Hlavním záměrem této diplomové práce bude navržení a následná implementace knihovny pro vizualizaci grafů v prostředí Microsoft .NET Framework. Konkrétně se bude jednat o grafy tvořené vrcholy a hranami, nepůjde tedy o znázornění grafů výšečových, sloupcových, spojnicových a jiných, které známe například ze statistiky nebo oblasti financí.

Důležitou součástí knihovny by měl být algoritmus schopný rozložit vrcholy grafu po ploše s ohledem na jejich vzájemné vazby tak, aby výsledná vizualizace byla co možná nejpřehlednější a nejnázornější. Pozornost bude věnována také možnosti individualizace vzhledu vrcholů a hran výsledného grafu.

Jako praktická ukázka demonstrace schopností knihovny by měla sloužit samostatná aplikace zpřístupňující všechny funkcionality knihovny prostřednictvím intuitivního grafického uživatelského prostředí.

V neposlední řadě bude prostor věnován také propojení knihovny s algoritmem analytického programování, a to za účelem vizualizace jeho výsledků v reálném čase formou názorného grafu.

## **I. TEORETICKÁ ČÁST**

## 1 ÚVOD DO GRAFŮ

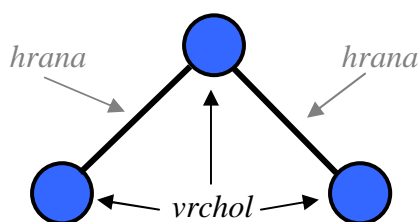
Grafy, kterými se budeme zabývat v rámci této diplomové práce, jsou součástí teorie grafů, což je odvětví věnující se zkoumání struktur grafů. Nejedná se tedy o grafy používané například pro vizualizaci nejrůznějších finančních, statistických a podobných výsledků (grafy výšečové, sloupcové, spojnicové, aj.).

Zjednodušeně lze graf považovat za uspořádanou dvojici množin  $V$  a  $E$ , přičemž  $V$  je množina vrcholů (vertex) a  $E$  je množina hran (edge). [1]

$$G = (V, E) \quad (1)$$

Využitím grafů je možné provést reprezentaci nejrůznějších problémů napříč mnoha obory. Lze zachytit vztahy mezi definovanými objekty (vrcholy), kde vždy konkrétní vztahy mezi nimi jsou hranami.

Mnohdy může jít o znázornění reálných problémů ve zjednodušené formě, která díky reprezentaci v podobě grafu vede k zásadně snazšímu pochopení problému jako celku. [1]



Obr. 1. Vrcholy a hrany

V následujícím textu se stručně seznámíme alespoň se základními pojmy, které souvisí s teorií grafů a jejich znázorňováním.

### 1.1 Hrana

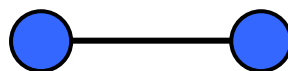
Jde obecně o  $k$ -tici (uspořádaná nebo neuspořádaná dvojice) vrcholů grafu. Nejčastěji je hrana znázorněna jako přímka spojující dva vrcholy. [2]

### 1.1.1 Základní rozdělení hran

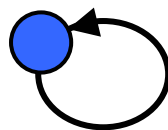
- **Orientovaná hrana** – tímto typem hran lze procházet pouze ve směru, který je naznačen; jde o uspořádanou dvojici vrcholů



- **Neorientovaná hrana** – zde jde o neuspořádanou dvojici vrcholů; hrana je průchozí v obou směrech



- **Smyčka** – hrana spojuje tentýž vrchol; začátek hrany a její konec je umístěn ve shodném vrcholu



- **Násobné hrany** – jde o více hran, které spojují tentýž vrchol



S typem hran souvisí také označení grafů. Grafy lze pak dle výskytu jednotlivých typů hran nazývat následovně:

- **Jednoduchý graf** – jde o graf bez výskytu násobných hran a smyček
- **Multigraf** – takový graf obsahuje násobné hrany
- **Pseudograf** – vyznačuje se výskytem smyček [3] [4]

## 1.2 Vrchol

Vrchol grafu, někdy nazýván také jako uzel, je prvek neprázdné množiny vrcholů  $V$ . Propojení dílčích vrcholů je realizováno pomocí hran, které mohou, ale nemusí být orientované (viz kapitola 1.1). [5]

Graficky se vrchol často znázorňuje jako kruh, reprezentace však může být libovolná.

### 1.3 Orientace

Orientace grafu souvisí s tím, zda jsou v rámci grafu využívány orientované hrany (viz kapitola 1.1).

#### Definice neorientovaného grafu

Jedná se o dvojici  $G = \langle V, E \rangle$ , kde  $V$  je neprázdná množina vrcholů a  $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$  je množina dvouprvkových množin vrcholů (neorientovaných hran). [5]

#### Definice orientovaného grafu

Jedná se o dvojici  $G = \langle V, E \rangle$ , kde  $V$  je neprázdná množina vrcholů a  $E \subseteq V \times V$  je množina uspořádaných dvojic vrcholů (orientovaných hran). [5]

### 1.4 Cesta

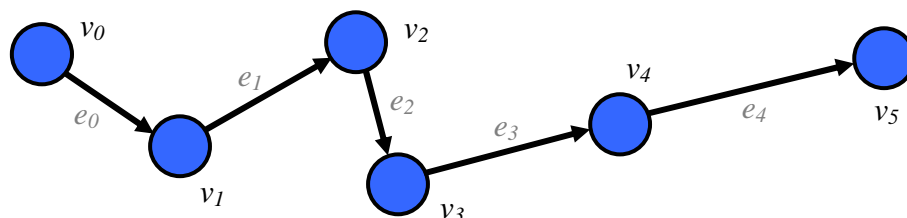
Cestou je v grafu  $G = \langle V, E \rangle$  označována posloupnost  $P = (v_0, e_1, v_1, \dots, e_n, v_n)$ ,

kde pro neorientovaný graf platí:  $e_i = \{v_{i-1}, v_i\}$ ,

pro orientovaný graf platí:  $e_i = (v_{i-1}, v_i)$

a zároveň je splněno  $v_i \neq v_j$  pro  $i \neq j$ . [6]

Jinými slovy jde o posloupnost vrcholů spojených hranami, kdy se v cestě žádný z vrcholů nevyskytuje více než jednou.



Obr. 2. Vrcholy a hrany tvořící cestu

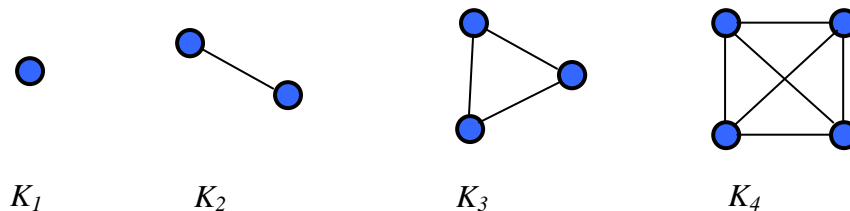
Jako typický příklad z běžného života lze uvést klasickou trasu při průjezdu několika městy napříč republikou. Jednotlivá města jsou vrcholy grafu a cesty zase jeho hranami.

Jako **délka cesty** je označován, dle konkrétního případu, počet vrcholů nebo případně počet hran.

Pokud platí  $v_0 = v_n$ , tak se již z formálního pohledu nejedná o cestu, ale o **kružnici**.

## 1.5 Úplný graf

Jedná se o neorientovaný graf, v němž jsou každé dva vrcholy spojeny hranou. Takový graf je často označován jako  $K_n$ , kde  $n$  je počet vrcholů úplného grafu.



Obr. 3. Příklady úplných grafů

Úplný graf, který má  $n$  vrcholů, má přesně  $\frac{n \cdot (n-1)}{2}$  hran. [7]

## 1.6 Strom

Stromem je označován neorientovaný graf, takový, který neobsahuje žádnou smyčku a je souvislý. Strom může obsahovat jeden kořen (hlavní vrchol), směrem od kořene pak vedou hrany připojující další vrcholy.

U stromů se můžeme setkat s následujícími názvoslovím:

- **Potomek** – potomkem konkrétního vrcholu můžeme nazvat všechny vrcholy, ke kterým vede hrana z tohoto původního vrcholu (rodič)
- **List** – jde o vrchol, který nemá žádné další potomky
- **Větev** – jedná se o přesně a jednoznačně určenou cestu od kořene k listu

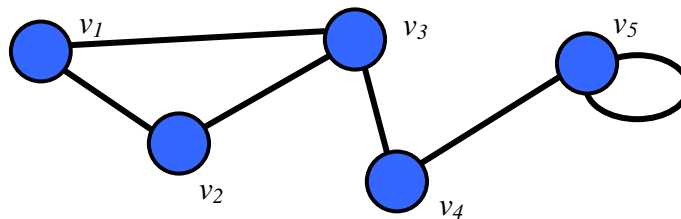
## 1.7 Reprezentace grafů

Grafy lze reprezentovat a zaznamenat několika způsoby, nejčastěji se využívá maticového zápisu, který sice není ideální pro názorné pochopení, ale pro jednoznačnost zápisu je ideální. [8]

### 1.7.1 Matice sousednosti

Jedná se o zápis pomocí čtvercové matice  $n \times n$ , která je typu  $\{0,1\}^{n \times n}$ . V případě neorientovaných grafů lze využít pouze trojúhelníkovou matici, přičemž je jedno, zda se použije horní, či dolní. [9]

Zjednodušeně pak můžeme konkrétní řádky/sloupce matice sousednosti chápat jako vrcholy a výskyt hodnoty 1 na odpovídající pozici značí, že mezi vrcholy existuje hrana. Konkrétně první řádek uvedené matice  $(0 \ 1 \ 1 \ 0 \ 0)$  nám říká, že vrchol  $v_1$  je propojen (sousedí) s vrcholy  $v_2$  a  $v_3$ .



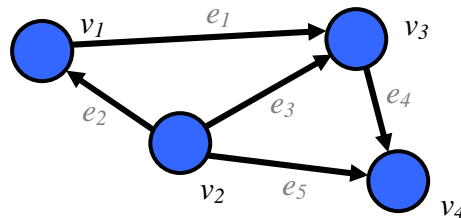
$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \text{ nebo také } \begin{pmatrix} 0 & & & & \\ 1 & 0 & & & \\ 1 & 1 & 0 & & \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Graf zapsaný výše uvedenou první maticí lze zapsat také pouze trojúhelníkovou maticí, která popis grafu zachycuje stejně podrobně.

### 1.7.2 Matice incidence

Opět se jedná o maticový zápis. Pokud je počet vrcholů označen jako  $n$  a počet hran jako  $m$ , tak jde o matici typu  $\{-1,0,1\}^{n \times m}$ .

Řádky matice znázorňují vrcholy a sloupce naopak hrany. Hrana je vždy zapsána tak, že u orientovaných grafů má v počátečním vrcholu 1 a u cílového -1. Pro neorientovaný graf se vždy uvádí 1 pro počáteční i cílový vrchol.



$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ -1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 \end{pmatrix} \text{ pro neorientovaný graf: } \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

### 1.7.3 Grafická reprezentace

Narozdíl od maticových zápisů je grafická reprezentace pro člověka mnohem názornější a více pochopitelná. Jde o klasické grafické zakreslení vrcholů a hran grafu do jednoduchého obrázku.

Tímto typem reprezentace se budeme dále zabývat v praktické části, která se bude věnovat automatickému (programovému) generování grafických výstupů na základě různých typů vstupních zápisů.

## 2 EXISTUJÍCÍ SOFTWAREVÁ ŘEŠENÍ

V současné době existuje celá řada nástrojů pro vizualizaci grafů, pouze však některé nabízejí přímé propojení s uživatelskou .NET aplikací, například formou DLL knihovny. U ostatních je třeba propojení řešit separátně.

Konkrétně se podíváme na dva různé produkty, přičemž prvním bude placený produkt yFiles .NET a druhým pak open-source projekt Gephi.

### 2.1 yFiles .NET

Produkt yFiles ve variantě .NET nabízí řešení podporující vytváření Windows Form, Windows Presentation Foundation (WPF) a Silverlight aplikací.

Mimo jiné přímo podporuje techniky pro vizualizaci a analýzu grafů. Dostupné funkce jsou závislé na distribuční variantě tohoto produktu.

#### 2.1.1 Varianty produktu

Dostupné jsou konkrétně 3 varianty:

##### Layout

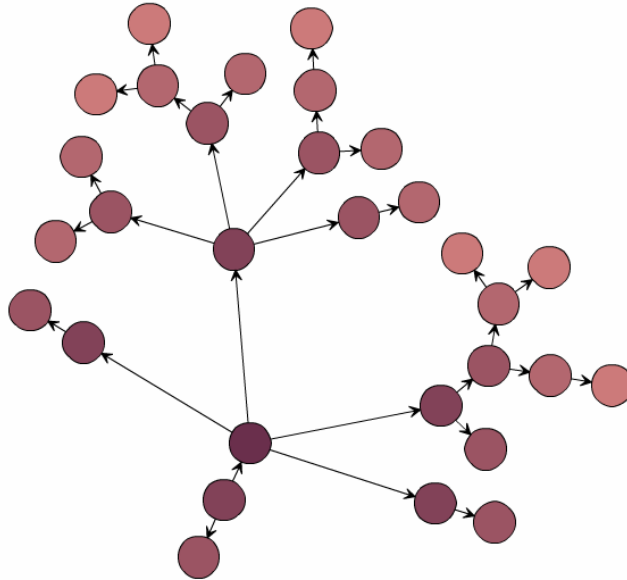
- Datové struktury pro efektivní práci s grafy
- Velké množství algoritmů pro tvorbu grafů
- Algoritmy pro automatické rozložení grafů a diagramů
- Algoritmy pro automatické rozložení spojů v diagramech
- Automatické rozmísťování popisků v diagramu

##### Viewer

- Vizualní reprezentace grafů
- Nástroj pro interaktivní prohlížení grafů
- Podpora vnořených grafových struktur
- Podpora mnoha vstupních formátů, export pouze v obrazových formátech
- Podpora tisku

## Complete

- Jedná se o kompletní produkt, který obsahuje všechny funkce dostupné ve variantách Layout a Viewer.



Obr. 4 - Náhled grafu vytvořeného v aplikaci yFiles

### 2.1.2 Ceny

Cenové podmínky jsou závislé na variantě produktu a způsobu použití. Jak již bylo výše uvedeno, celkem jsou dostupné tři varianty Layout, Viewer a Complete. První dvě jsou ve shodné cenové hladině, třetí (tvořená kombinací dvou předchozích) není dvojnásobně dražší, ale je na ni poskytována sleva ve výši přibližně 30%.

Pro plnohodnotné používání je vhodné mít k dispozici jak funkce ve variantě Layout, tak i Viewer, nabízí se tedy rovnou využití třetího a nejdražšího produktu Complete.

Cena se dále odvíjí dle způsobu využití, celkem jsou nabízeny čtyři různé typy:

- **Licence pro jednoho vývojáře** – licence je platná pro jednoho individuálního vývojáře, který je uveden v objednávce
- **Licence pro projekt** – licence určená pro konkrétní jeden softwarový produkt, který je specifikován v objednávce; licence dále umožňuje současné využívání až třemi vývojáři

- **Licence pro webový portál** – licence pro přesně specifikovaný webový portál s možností přístupu více vývojářů a dalších webových služeb
- **Zdrojové kódy** – licence na nebinární variantu produktu

Tab. 1. Cenový přehled yFiles .NET [10]

	yFiles.NET Layout	yFiles.NET Viewer	yFiles.NET Complete
<b>Jeden vývojář</b>	€ 3 900	€ 3 900	€ 5 300
<i>roční předplatné</i>	€ 1 170	€ 1 170	€ 1 590
<b>Projekt</b>	€ 7 800	€ 7 800	€ 10 600
<i>roční předplatné</i>	€ 2 340	€ 2 340	€ 3 180
<b>Portál</b>	€ 15 600	€ 15 600	€ 21 200
<i>roční předplatné</i>	€ 4 680	€ 4 680	€ 6 360
<b>Zdrojové kódy</b>	€ 31 200	€ 31 200	€ 42 400
<i>roční předplatné</i>	€ 9 360	€ 9 360	€ 12 720

U všech těchto zmíněných variant a typů licencí má zákazník možnost využít jednorázového nákupu licence nebo ročního předplatného, kde se jedná přibližně o třetinovou částku. Výhodou předplatného jsou zejména bezplatné průběžné aktualizace vybraného produktu.

Při nákupu licence pro akademické účely je poskytována sleva ve výši 30%. [10]

### 2.1.3 Výhody a nevýhody

Za velkou výhodu produktu yFiles lze určitě považovat množství algoritmů pro sestavování grafů a také mnoho vizuálních stylů pro grafické znázornění. Nechybí prostor pro vlastní individualizaci, a to jak jednotlivých grafů, tak také samotného prostředí aplikace. Jedná se o velmi komplexní nástroj nejen pro vizualizaci grafů, ale také pro jejich podrobnou analýzu a výzkum.

Velmi dobrá je také podpora různých platforem, v nabídce jsou tyto následující produktové varianty:

- yFiles for Java
- yFiles for .NET, dále se ještě dělí na: .NET, WPF, Silverlight
- yFiles for HTML
- yFiles for the Web

- yFiles for Android

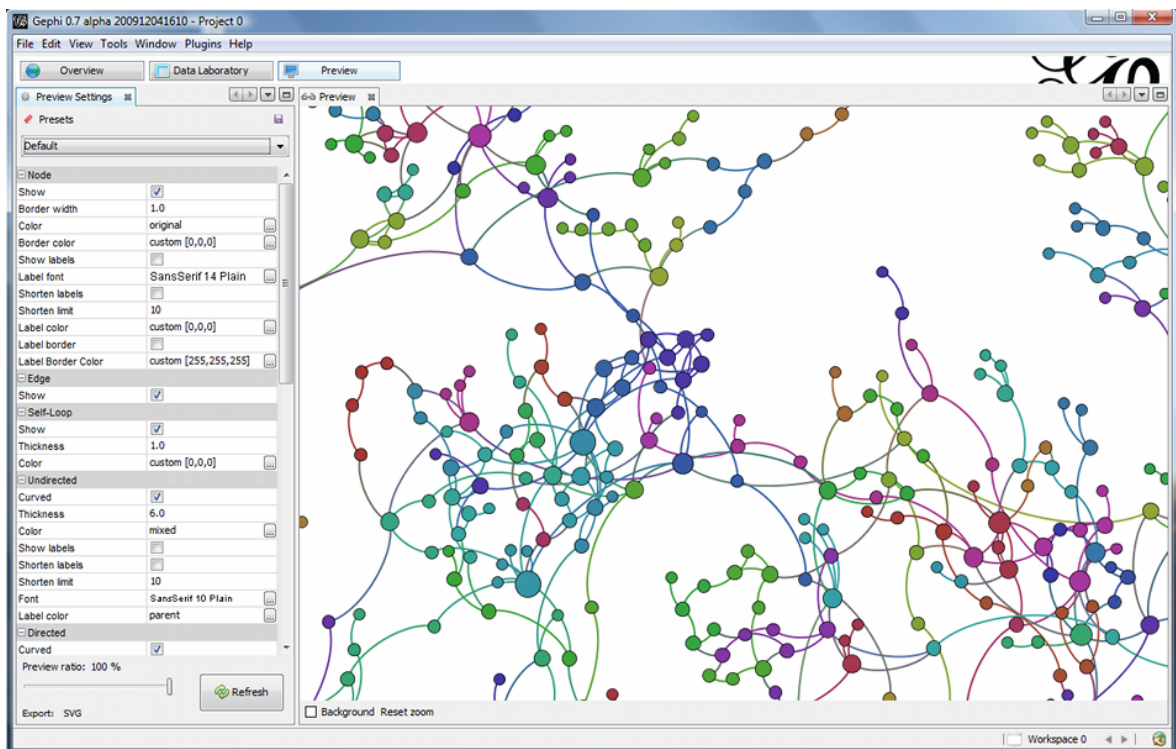
Hlavní nevýhodou je bezesporu vysoká cena produktu, která u kompletní varianty začíná na částce 5 300 Eur (přibližně 140tis. Kč) pro jednoho vývojáře a může se vyšplhat až k 42 400 Eur u nebinární varianty.

Navíc je nutné pro každou platformu kupovat samostatnou licence, a to i v případě, že jde třeba o .NET a WPF variantu.

## 2.2 Gephi

Jedná se o interaktivní platformu pro vizualizaci a prohlížení nejrůznějších typů grafů. Velkou výhodou je fakt, že celá platforma je open-source a zcela zdarma.

Nabízí celou řadu funkcí a možností. Gephi lze určitě považovat za zajímavou alternativu k produktům řady yFiles.



Obr. 5. Náhled aplikace Gephi [11]

Gephi vyniká hlavně v těchto oblastech:

- **Zkoumání a analýza dat** – analýza dat v reálném čase orientovaná na intuitivní ovládání a procházení
- **Analýza vazeb** – zkoumání vztahů mezi objekty a jejich vzájemných propojení
- **Orientace na sociální sítě** – vytváření vizualizací postavených na datech ze sociálních sítí
- **Biologická analýza** – znázornění vazeb z oblasti biologie

Jako hlavní nevýhodu v našem případě je třeba zmínit, že je Gephi napsáno v jazyce Java a nelze jej tedy využít přímo ve vlastní .NET aplikaci formou externí dynamické knihovny. Díky tomuto faktu se rozsah využití znatelně omezuje, nicméně pro běžné použití lze Gephi ve spolupráci s .NET aplikací použít.

Odlišnost platformy uživatelské (.NET) a vizualizační (Java) aplikace s sebou nese také problém týkající se distribuce a využívání výsledného produktu, kdy je potřeba zajistit vhodné běhové prostředí pro obě aplikace.

Dalším limitujícím faktorem by mohly být hardwarové nároky na grafickou kartu, neboť pro vykreslování se využívá OpenGL 3D, které nemusí být na starších grafických kartách a některých systémech podporováno.

### 3 DEFINICE UŽIVATELSKÝCH POŽADAVKŮ

Jedním z hlavních problémů, který bude řešen dále v praktické části, bude návrh a následná implementace algoritmu určeného k automatickému rozmístění vrcholů grafu tak, aby byla výsledná vizualizace co možná neoptimalnější, bez zbytečných překrývajících se prvků a na první pohled názorná.

Výstupem algoritmu by mělo být grafické znázornění grafu, který bude zadán v určeném formátu (textově, případně maticově).

Samotný algoritmus bude součástí dynamické knihovny (DLL), jež bude kromě zmíněného algoritmu nabízet i další možnosti sloužící k individualizaci výsledné vizualizace. Samozřejmostí by měla být vhodná rozhraní pro vstupní a výstupní komunikaci s knihovnou.

Jako názorná ukázka použití výsledné knihovny bude vytvořen obecný editor nabízející kompletní práci s grafy v grafickém rozhraní aplikace. Za pomoci něj by mělo jít vytvářet nové grafy (vrcholy a spojnice mezi nimi), upravovat grafickou podobu jednotlivých prvků, ukládat a samozřejmě zpětně načítat již vytvořené projekty.

Samotná knihovna bude také použita v samostatném ukázkovém projektu, kde bude v reálném čase vizualizovat výsledky algoritmu analytického programování.

Požadavky na výstup této práce lze tedy shrnout do následujících hlavních bodů:

- **Knihovna pro vizualizaci grafů** vhodná pro použití ve vlastním projektu (aplikaci)
- **Editor grafů** využívající vytvořenou knihovnu, schopný samostatného fungování a nabízející intuitivní uživatelské rozhraní
- **Demonstrace možností knihovny** na konkrétním projektu, jež se zabývá analytickým programováním

Další neodmyslitelnou částí by měly být doprovodné informace sloužící k snazšímu pochopení fungování a jednoduchému použití, jak samotné knihovny, tak také zmíněného editoru.

## 4 POPIS POUŽITÝCH TECHNOLOGIÍ

Jelikož by měla být výstupem této práce dynamická knihovna pro použití v prostředí .NET, bude k vytvoření využita výhradně technologie Microsoft .NET Framework.

### 4.1 Microsoft .NET Framework

Obecně .NET je souhrnné označení pro řadu technologií dostupných na několika platformách, ke kterým patří Web, Windows a Pocket PC.

Specifikace jádra .NET je standardizovaná a nazývá se Common Language Infrastructure, zkráceně CLI. Jde o otevřenou specifikaci popisující vlastnosti proveditelného kódu a také o specifikaci prostředí pro jeho běh. Tyto vlastnosti tvoří základní jádro Microsoft .NET Framework. Specifikace CLI byla vyvinuta nejen společností Microsoft.

Jde zejména o specifikaci prostředí umožňujícího využívat více vysokoúrovňových programovacích jazyků bez omezení na konkrétní počítačovou platformu. Odpadá tak nutnost individualizace překladače pro odlišné vlastnosti jednotlivých platforem. [12]

Implementací specifikace CLI je Common Language Runtime, zkráceně CLR.

Do specifikace CLI mimo jiné patří následující:

- **The Common Type System (CTS)** – jde o sadu datových typů a operací, využívá se v mnoha programovacích jazycích
- **Metadata** – zachycení struktury programu bez závislosti na konkrétní jazyk
- **Common Language Specification (CLS)** – základní pravidla společná pro každý jazyk odpovídající specifikaci CLI
- **Virtual Execution System (VES)** – zavádí a provádí programy dle CLI

Dostupné kompatibilní jazyky jsou překládány do „mezijazyka“, který je nezávislý na hardwarové platformě, nazývá se Common Intermediate Language, zkráceně CIL. Ke kompilaci do strojového kódu, pro konkrétní hardwarovou platformu, dochází právě na základě CIL. [14] [15]

Framework Microsoft .NET existuje v několika obměnách. Nejrozšířenější variantou je standardní Microsoft .NET Framework určený pro osobní počítače s verzí operačního systému Windows XP a vyšší. [16]

Redukovanou verzí frameworku je .NET Compact Framework dostupný na zařízeních se systémem Windows CE a Windows Mobile, tedy zejména na smartphonech. Pro přístroje s opravdu minimálními výpočetními možnostmi je určena varianta .NET Micro Framework, která je ještě více omezena co do funkčnosti.

Zcela nezávislá je pak open source implementace .NET nazvaná Mono. Jde o variantu určenou pro operační systémy typu Unix, konkrétně Linux, OS X, atp. [17]

#### 4.1.1 Jazyky

Pro vývoj aplikací není předepsán žádný konkrétní jazyk. Jak již bylo výše uvedeno, výsledná aplikace je vždy přeložena do „mezijazyka“ CLI.

Nejčastěji jsou k vývoji .NET aplikací využívány jazyky C#, Visual Basic .NET nebo také Delphi. C# je velmi podobný jazykům C a Java. Visual Basic .NET, často zkráceně označován jako VB.NET, vychází z původního Visual Basic, je hojně využíván pro svou jednoduchost a jistou podobnost s člověku blízkým zápisem syntaxe.

Kromě těchto tří zmíněných jazyků je k dispozici ještě řada dalších, zde alespoň jmenovitě: Managed C++, F#, J#, IronPython nebo Boo.

#### 4.1.2 Verze

Samotný Microsoft .NET Framework je dostupný v několika verzích. S každou verzí se většinou pojí i nová verze vývojového prostředí.

- **.NET 1.0** – vydáno v roce 2002, vývojové prostředí Visual Studio .NET
- **.NET 1.1** – vydáno v roce 2003, vývojové prostředí Visual Studio .NET 2003
- **.NET 2.0** – vydáno v roce 2005, vývojové prostředí Visual Studio .NET 2005
- **.NET 3.0** – vydáno v roce 2006, zveřejněno prostředí Expression Blend
- **.NET 3.5** – vydáno v roce 2007, vývojové prostředí Visual Studio .NET 2008
- **.NET 4.0** – vydáno v roce 2010, vývojové prostředí Visual Studio .NET 2010
- **.NET 4.5** – vydáno v roce 2012, vývojové prostředí Visual Studio .NET 2012 [17]

### 4.1.3 Výkon

V rámci platformy .NET Framework je často kritizován tzv. garbage collector (zkráceně GC), který se automaticky stará o správu dostupné paměti. Vývoj nových aplikací je tak velmi snadný, neboť se programátor nemusí starat o veškeré náležitosti spojené s existencí a zánikem nových objektů a proměnných.

Každé pro má však také své proti a právě usnadnění, které garbage collector poskytuje, je vykoupen zvýšením výpočetního výkonu, který GC potřebuje pro svou činnost.

Základním principem GC je vyhodnocování, které objekty v paměti jsou využívány a které ne. To s sebou pochopitelně přináší nutnost tuto informaci někde uchovávat a vyhodnocovat. Tyto režie mohou v některých případech způsobovat velmi znatelné zpomalení projevující se špatnou plynulostí běhu aplikace.

### 4.1.4 Bezpečnost

Nevýhodou aplikací využívajících CLI bytový kód je poměrně jednoduchá možnost získání původního zdrojového kódu. Pomocí technik reverzního inženýrství lze díky specializovaným nástrojům získat z binárního souboru prakticky původní zdrojový kód, a to dokonce ve vybraném jazyce (i jiném, než v jakém byla aplikace původně napsána).

Tento problém z části řeší nástroje známé jako Obfuscatory, jež se snaží v několika krocích provést takové úpravy, které co nejvíce znemožní sestavení původního zdrojového kódu.

Často jde hlavně o odstranění komentářů a informací týkajících se dokumentace. Vymazání formátování, přejmenování proměnných a identifikátorů, ale také samotných procedur, tříd a funkcí.

Po tomto zásahu však musí být výsledný kód významově totožný se vstupem. Využití této techniky je vhodné zejména pro zamezení možnosti dalších úprav výsledné aplikace neoprávněnými osobami, nebo snaha uchovat použité know-how.

## 4.2 Microsoft Visual Studio

V drtivé většině případů se pro vývoj aplikací postavených na platformě Microsoft .NET Framework využívá vývojového prostředí Microsoft Visual Studio.

Obecně je vývojové prostředí často označováno také jako IDE, zkratka z anglického Integrated Development Environment. Používání vhodného IDE s sebou přináší značné usnadnění. Často jde totiž o kombinaci editoru určeného pro psaní zdrojového kódu, kompilátoru i debuggeru, který nabízí bohaté možnosti při ladění vyvíjené aplikace.

Microsoft Visual Studio je komplexní nástroj podporující vývoj jak konzolových aplikací, grafických aplikací Windows Form, tak i webových služeb. Editor zdrojového kódu podporuje kvalitní IntelliSense, jež přináší rychlejší a pohodlnější vývoj.

Vestavěný debugger umožňuje pracovat na dvou úrovních, tzn. na úrovni zdrojového i strojového kódu.

Mezi další důležité vestavěné nástroje patří editor pro grafický návrh aplikací s uživatelským rozhraním (Windows Form) nebo designér pro návrh databází. Možnosti vývojového prostředí lze samozřejmě dále rozšířit například o podporu pohodlnější spolupráce ve větším vývojovém týmu (Microsoft Team Foundation Server) nebo verzování.

Použití pro konkrétní programovací jazyk není nijak limitováno. Za pomoci jazykových služeb lze docílit podpory libovolného jazyka. Díky jazykovým balíčkům může být definováno zvýrazňování syntaxe, inteligentní doplňování kódu, podpora pro kompilování na pozadí, atp. V základu jsou však podporovány tři hlavní programovací jazyky v těchto variantách produktu:

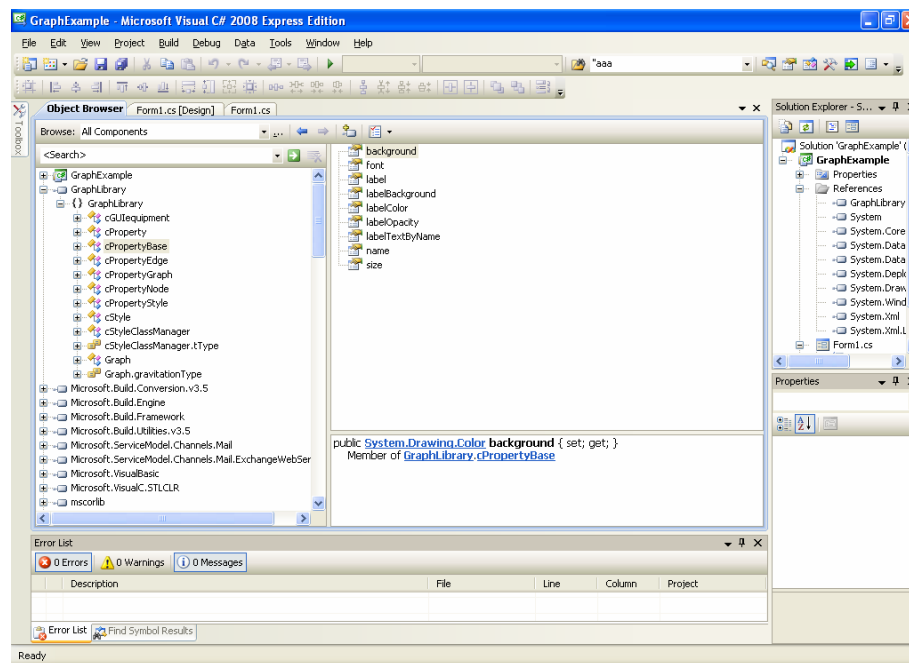
- **Visual C++**, jazyk C/C++
- **Visual Basic .NET**, jazyk VB.NET
- **Visual C#**, jazyk C#

Vývojové prostředí Visual Studio je možné získat v několika různých edicích, jednotlivé varianty jsou rozlišeny podle dostupných možností a funkcí, které nabízejí.

Nejpokročilejším produktem je Visual Studio Team System podporující vývoj a kompletní spolupráci ve velkých týmech. Následuje Visual Studio Professional a dále Visual Studio Standard, rozdíl je vždy v omezení funkčnosti a možnostech nastavení. Oproti nejvyšší verzi chybí například plná podpora pro procesory Itanium.

Uvedené verze nejsou dostupné zdarma a jsou zpoplatněny. Zajímavou alternativou k těmto produktům je řada Visual Studio Express. Ta existuje v samostatných variantách

Visual C++ Express, Visual Basic Express, Visual C# Express a Visual Web Developer Express. Důležitá informace je ta, že všechny tyto produkty řady Express jsou dostupné zcela zdarma a je možné je využít i k vývoji komerčních produktů. Oproti ostatním variantám je zde celá řada omezení (například nelze používat rozšíření, omezené možnosti nastavení projektu, nemožnost kompilace 64-bitových aplikací, atp.), nicméně pro běžný vývoj a vyzkoušení základních principů práce s tímto prostředím je to dostačující. [16]



Obr. 6. Náhled vývojového prostředí Visual Studio C# Express

## 5 FYZIKÁLNÍ ZÁKLADY

Abychom mohli v rámci praktické části realizovat algoritmus pro automatické rozvržení jednotlivých vrcholů grafu, je nutné zvolit vhodnou techniku. Jako nejideálnější se jeví nechat se inspirovat v klasické mechanice, tedy v oboru fyziky.

### 5.1 Pružnost

Pružnost, která se někdy nazývá také elasticita, je součástí mechaniky a zabývá se vztahy mezi deformacemi těles a navzájem působícími vnějšími silami.

Ve spojení s pružností je známo jméno britského fyzika Roberta Hooke, který se jako jeden z prvních zabýval zkoumáním vztahů působících sil na těleso a tím způsobenou deformací těchto těles. Byl velmi všestranným člověkem, který se zabýval mnoha odvětvími, v nichž přišel na významné objevy a vynálezy, jež se používají dodnes.

### 5.2 Hookův zákon

Hooke již v roce 1676 zformuloval zákon říkající, že při pružné deformaci je normálové napětí přímo úměrné relativnímu prodloužení. [18]

Zjednodušeně lze Hookeův zákon vyjádřit jako

$$F = kx \quad (2)$$

přičemž  $F$  je působící síla,  $k$  konstanta pružnosti materiálu a  $x$  prodloužení materiálu.

### 5.3 Elastická hystereze

Elastickou hysterezi lze také označit jako dopružování. Je možné si ji představit jako deformaci, která přetrvává i po odstranění všech vnějších sil, deformace mizí postupně. Deformací bude v našem konkrétním případě myšleno dočasné odchýlení vrcholů grafu od svých ustálených pozic. Při statické reprezentaci výstupu grafu nebude patrná, nicméně v rámci probíhajících iterací výpočtu dílčích pozic k ní bude docházet.

## **II. PRAKTICKÁ ČÁST**

## 6 ZÁKLADNÍ NÁVRH

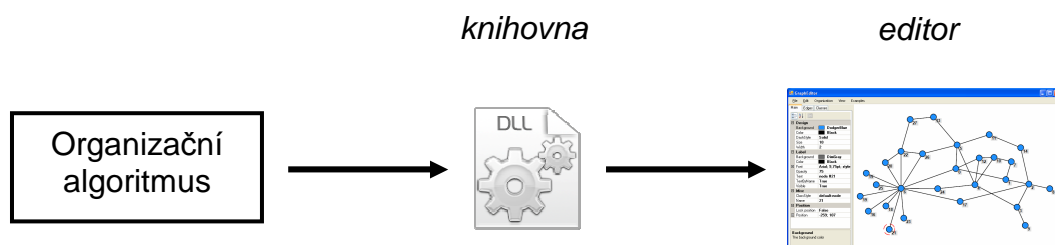
V rámci práce se budeme zabývat několika samostatnými částmi, které však budou ve výsledku propojeny do společného celku.

Hlavním cílem je navrhnout postup pro vizualizaci grafů zadaných maticově, pomocí XML, či jiným, pro člověka ne příliš názorným, způsobem. Zásadním prvkem tedy bude algoritmus zajišťující dynamické výpočty pozic jednotlivých vrcholů grafu.

Tento algoritmus bude součástí knihovny, jež bude zastřešovat všechny další potřebné funkce pro práci s grafem jako celkem i jeho jednotlivými objekty.

Výsledná knihovna bude následně použita jako jádro editačního nástroje, který zpřístupní všechny možnosti a funkce vestavěné knihovny prostřednictvím intuitivního uživatelského rozhraní.

Veškeré vytváření, prohlížení a editace grafů by mělo jít provádět právě pomocí tohoto grafického editoru. I běžný uživatel by tak měl mít možnost bez znalosti programování vytvářet vlastní grafy a schémata v jednoduchém graficky orientovaném prostředí.



*Obr. 7. Základní návrh a rozložení*

Knihovna bude dále využita v konkrétním příkladě pro vizualizaci algoritmu analytického programování. Bude sloužit jako grafický výstup z algoritmu, který jinak generuje pouze číselnou matici, či jiný na první pohled nesrozumitelný výstup.

## 7 KNIHOVNA PRO VIZUALIZACI GRAFŮ

Byla navržena a vytvořena knihovna, kterou lze použít v libovolném projektu postaveném na platformě Microsoft .NET Framework, od verze 3.5 a vyšší. Jak vyplývá z použité technologie, knihovnu je možné pochopitelně využívat bez ohledu na omezení konkrétního programovacího jazyka.

Její elementární funkcí je grafické znázornění vstupu zadaného jednou z dostupných možností. Výstupem může být statický obrázek nebo, v případě automatického rozmístění vrcholů grafu, průběžně vykreslovaná sekvence za sebou jdoucích organizačních kroků.

### 7.1 Vnitřní reprezentace vrcholů

Každý vrchol (uzel) je reprezentován jedním objektem vytvořeným dle třídy *cNode*. Všechny tyto vrcholy jsou uchovávány v privátním poli *nodes*, které je pro jednoduchost indexováno číselně, od 0.

Pro snazší práci a přístup k jednotlivým vrcholům byl zaveden pomocný index formou slovníkové struktury *nodesIndex*. Klíčem v tomto slovníku je název vrcholu, tzv. jeho unikátní identifikátor, hodnota ukazuje přímo do pole *nodes*.

Každý vrchol v sobě nese několik základních informací:

- **id** – textový unikátní identifikátor vrcholu, povoleny jsou znaky *[0-9A-Za-z]*
- **pos** – pozice vrcholu, zadaná jako datový typ *Point*, souřadnice X a Y
- **velocity** – pomocná proměnná používaná při automatické organizaci, jde o datový typ *PontF* umožňující uchovávat pozici X a Y pomocí desetinných čísel
- **netForce** – pomocná proměnná používaná při automatické organizaci, jde o datový typ *PontF* umožňující uchovávat pozici X a Y pomocí desetinných čísel
- **isLocked** – proměnná typu Boolean značící, zda je pozice vrcholu uzamčená
- **title** – textový popis vrcholu, který bude viditelný, může obsahovat libovolné znaky
- **style** – jde o velmi důležitou proměnnou ukazující na objekt třídy *cStyle*, který zajišťuje definici vizuální podoby vrcholu

## 7.2 Vnitřní reprezentace hran

Hrany spojující vrcholy grafu jsou ukládány do slovníkové struktury. Hodnotou prvku, uloženou pod daným klíčem, je vždy objekt dle třídy *cEdge*.

Objekt hrany uchovává pouze dvě informace:

- **type** – typ orientace uložený v datovém typu *Char* (viz kapitola 7.2.1)
- **style** – stejně jako v případě vrcholu jde o proměnnou ukazující na objekt vytvořený dle třídy *cStyle*, který zajišťuje definici vizuální podoby hrany

Klíč je tvořen ze tří přesně určených částí:

- Počáteční vrchol
- Orientace hrany
- Konečný vrchol

Počáteční a konečný vrchol jsou uváděny jako indexy směřující do pole *edges*, jde tedy vždy o číselné hodnoty.

Orientace hrany je jednoznaková hodnota, která určuje typ spojení obou uvedených vrcholů.

Konkrétní klíč hrany tak může vypadat například takto:

„10>16“

a definuje, že vrchol s indexem 10 je připojen k vrcholu s indexem 16, přičemž jde o orientovanou hranu, kdy grafická šipka bude směřovat z počátečního do koncového vrcholu.

### 7.2.1 Typy orientací hran

Celkem lze hraně určit jednu ze čtyř různých orientací:

- „-“ – hrana není orientovaná
- „>“ – orientace z počátečního do koncového vrcholu
- „<“ – orientace z koncového do počátečního vrcholu
- „=“ – grafická šipka bude uvedena u obou vrcholů

### 7.3 Uchovávání pozic

Zápis a uchovávání pozic jednotlivých objektů, zejména vrcholů, je realizován pomocí proměnných datového typu *Point*, případně *PointF*, je-li nutné uchovávat pozici s přesností desetinných čísel. *PointF* se využívá hlavně při pomocných výpočtech u automatické organizaci rozmístění vrcholů, kde je vyžadována větší přesnost a zjednodušení pouze na celá čísla by zbytečně zhoršovalo kvalitu algoritmu.

Jde-li o určení pozice vrcholu na ploše (typ *Point*), je jedna číselná jednotka rovna jednomu pixelu. Počátek souřadného systému, bod [0; 0], je umístěn vždy v levém horním rohu. Dále v kladném směru jsou osy orientovány směrem doleva (osa X) a dole (osa Y).

### 7.4 Možnosti vstupu

Knihovna podporuje dvě různé metody vstupu. Každá z těchto metod je vhodná pro jiný účel použití. V obou případech je nejdůležitější zachycení vztahů mezi jednotlivými uzly. Celkem lze využít čtyři různé typy hran. Formou vestavěného překladače je dostupná i třetí možnost specifického vstupu, viz kapitola 11.

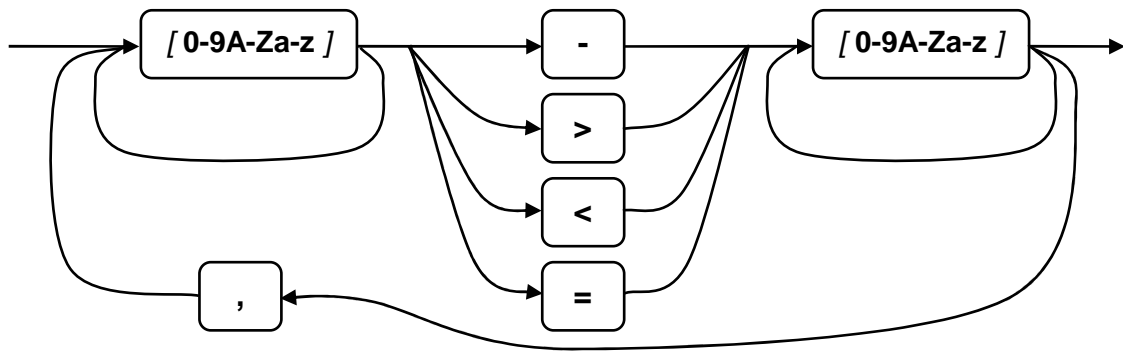
#### 7.4.1 Textový řetězec

Definice grafu pomocí textového řetězce sestaveného dle přesně daných pravidel je nejjednodušší variantou zadání. Jedná se poměrně o snadnou a názornou variantu vstupu.

Možnosti sestavení textového vstupu nejlépe demonstruje obrázek (Obr. 8), na němž je dle diagramu patrné, že zápis jedné vazby se vždy skládá z alfanumerického identifikátoru počátečního vrcholu, následuje typ vazby a dále opět alfanumerický identifikátor, tentokrát cílového vrcholu. Tím je definován jeden propoj. Definice jedné vazby má tedy vždy minimálně 3 znaky (v případě, že počáteční a cílový vrchol mají jednoznakové identifikátory), její délka zápisu se odvíjí od délky identifikátorů vrcholů a lze ji vyjádřit jako:

$$\text{délka definice jedné vazby} = \text{délka}(\text{počáteční vrchol}) + 1 + \text{délka}(\text{cílový vrchol})$$

Jednotlivé definice vazeb jsou odděleny čárkou, zapisovány za sebou.



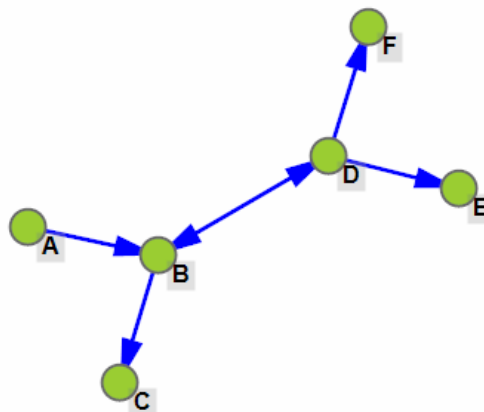
Obr. 8. Diagram tvorby textového vstupu

Mějme vrcholy grafu A, B, C, D, E a F. Uvažujme, že spojnice z vrcholu A vede do vrcholu B, z vrcholu B vede spojnice do C. Vrchol B je navíc obousměrně propojeno s D. Z vrcholu D vedou spojnice do vrcholů E a F.

Textový zápis zachycující uvedené vazby by vypadal následovně:

$$A>B,B>C,B=D,D>E,D>F$$

Jak je vidět, definice je velmi jednoduchá a intuitivní. Knihovna na základě tohoto zadání poskytne následující grafický výstup:



Obr. 9. Grafický výstup dle textového zadání

Můžeme si všimnout, že při vstupu nijak nedefinujeme vizuální podobu vrcholů ani hran. Rovněž na grafickém výstupu z knihovny je patrné, že všechny prvky jsou vizuálně totožné. Jde o hlavní omezení textového zadávání. Pro jednoduchost byla tato metoda vstupu navržena tak, aby splňovala svůj základní účel a zbytečně nebyla přehlcena složitostí syntaxe.

Přesto, že je zadávání takto jednoduché a nenabízí přímou možnost ovlivnit vizuální podobu grafu, lze tato nastavení nad jednotlivými prvky grafu, nebo grafem jako celkem, provádět dodatečně, a to pomocí vestavěných funkcí, viz kapitola 8: Manuál pro práci s knihovnou.

#### 7.4.2 XML vstup

XML vstup je varianta definice grafu, která nabízí mnohem více možností. Je využíváno základní verze XML s použitím znakové sady UTF-8. [19]

```
<?xml version="1.0" encoding="utf-8" ?>
- <graph>
+ <settings>
+ <classes>
+ <nodes>
+ <edges>
+ <editor>
</graph>
```

Jak je vidět v uvedené ukázce XML, tak základ tvoří kořenový element <graph>, ten obsahuje pět základních elementů:

- <settings> - nese informace o nastavení diagramu (grafu)
- <classes> - definice grafických tříd
- <nodes> - uchovávání zejména pozic vrcholů a jejich popisků
- <edges> - určuje vztahy mezi vrcholy, jejich hrany
- <editor> - doplňující nastavení týkající se individualizace preferencí editoru

Ne všechny tyto elementy jsou povinné. Největší význam a důležitost mají sekce <nodes> a <edges>, zbylé elementy obsahují doprovodné informace, buď o vzhledu vykreslovaných prvků, nebo o celkovém nastavení.

Minimalistická varianta zápisu XML využívající pouze kořenový element <graph> a element <edges> může vypadat i následovně:

```
<?xml version="1.0" encoding="utf-8" ?>
- <graph>
  <edges>A>B,B>C,B=D,D>E,D>F</edges>
</graph>
```

Grafický výstup z knihovny by byl v tomto případě shodný s tím, který byl uveden v předchozí kapitole při textovém zadání definice grafu (Obr. 9). Oba tyto zápisy jsou rovnocenné.

### Sekce settings

Tato sekce obsahuje dílčí elementy <setting>, které mají vždy povinný atribut „name“. Dále se používá také atribut „value“, který však není povinný, místo něj může být uvedeno například určení pozice pomocí dvojice atributů „x“ a „y“.

Obecně se tato sekce využívá k nastavení týkající se polohy počátku, umístění a chování celého grafu.

### Sekce classes

Elementy <class> nacházející se v této části definují vizuální podobu vrcholů a hran. Tyto grafické třídy, jak je budeme i v následujícím textu nazývat, mají vždy unikátní identifikátor zapsaný v atributu „id“. Důležitým atributem je „style“, kde je uvedena kompletní definice vzhledu. Implementována je také možnost hierarchického dědění vlastností používaných tříd. O možnostech stylování a dalších podrobnostech pojednává také kapitola 7.5.

### Sekce nodes

Vlastnosti vrcholů jsou zapsány jako elementy <node>. Unikátní identifikátor vrcholu je uložen v atributu „id“. Je-li potřeba specifikovat přesnou pozici vrcholu, lze použít dvojici atributů „x“ a „y“. Hodnota elementu je použita jako popis vrcholu.

Pro nastavení grafické podoby vrcholu můžeme využít dva různé způsoby:

- **atribut „style“** – stejně jako v případě grafických tříd, tak i přímo konkrétnímu elementu reprezentujícímu vrchol, můžeme pomocí tohoto atributu nastavit individuální vizuální vlastnosti dle dostupných možností
- **atribut „class“** – pro efektivnější použití a jednotnost v rámci skupiny více podobných vrcholů lze pomocí tohoto atributu přiřadit vrcholu vybranou grafickou třídu, tím je možné docílit toho, že při požadavku na změnu konkrétního parametru není nutné upravovat tuto hodnotu u všech vrcholů, ale pouze v jedné použité třídě

## Sekce edges

Pro zaznamenání samotných vazeb mezi vrcholy grafu je možné využít dvojího zápisu. Lze zadat přímo textovou sekvenci sestavenou dle dříve uvedeného diagramu (Obr. 8) jako hodnotu elementu <edges> nebo použít pro každou vazbu samostatný element <edge> s atributem „value“:

```
<edge value="A>B" />
```

Druhá varianta je pochopitelně více přehledná a je také implementována jako výchozí při procesu ukládání XML do souboru.

Určení vizuální podoby hrany se provádí stejně jako v případě elementu <node> pro vrcholy. Tzn. je zde možnost individuálního nastavení stylu pomocí atributu „style“ nebo využití grafických stylů.

## Sekce editor

Z pohledu knihovny nemá tato sekce žádný význam, není tedy prováděno ani její další zpracovávání. Je určena výhradně pro potřeby editoru, který z ní načítá informace o svých preferencích (stav zapnutí automatické organizace vrcholů, rychlost, podkladový obrázek pro snazší vytváření grafu, atp.).

## 7.5 Stylování

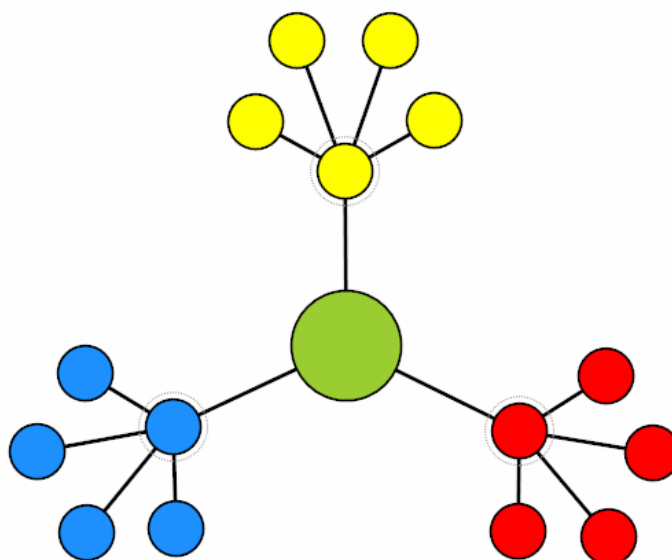
Princip stylování je navržen shodně pro vrcholy i hrany. Konkrétní styl je vždy uchovávan v objektu, který je tvořen dle třídy *cStyle*. Třída *cStyle* poskytuje veškeré metody potřebné k sestavení objektu do odpovídající podoby a funkčnosti. Jelikož je nastavení stylů v XML uchováváno jako textový řetězec, kde jsou jednotlivé názvy hodnot s jejich parametry odděleny středníky, je vždy nutné v konstruktoru, kterému se tento řetězec předává, provést patřičné parsování a přetypování na odpovídající datové typy. Takto vytvořený objekt je pak schopen, na základě uvedení názvu klíče žádané hodnoty, dynamicky vracet požadované nastavení v odpovídajícím a očekávaném datovém typu.

Vnitřně jsou dynamické vlastnosti objektu (rozumějme jako definice stylu) uchovávány ve slovníkové struktuře. Klíč je vždy textový název vlastnosti a odpovídající hodnota je přímo hodnotou vlastnosti (obecně objekt).

Jak již bylo naznačeno v kapitole o možnostech XML vstupu, konkrétně u sekce <nodes>, tak lze rozlišovat dva typy stylování:

- **Absolutní nastavení** – nastavení je individuální pro konkrétní objekt (vrchol nebo hranu) a je uloženo přímo v objektu třídy *cStyle*, na který vede ukazatel uložený v proměnné *style* právě objektu reprezentujícím vrchol nebo hranu; každý vrchol nebo hrana tak může mít odlišný vzhled
- **Využití předdefinované třídy** – každý objekt třídy *cStyle* může mít určenu grafickou třídu, ze které dědí nastavení; reálně je vždy nejprve žádaná vlastnost hledána uvnitř objektu, pokud není definována tam, je vlastnost načtena z nadřazené grafické třídy

Agregátorem všech vyskytujících se grafických tříd je objekt tvořený dle třídy *cStyleClassManager*. Díky jednoduchému návrhu bylo možné logiku tříd implementovat jako skupinu objektů třídy *cStyle*, kde jednotlivé objekty jsou přístupné pomocí unikátního identifikátoru (název třídy).



Obr. 10. Ukázka použití grafických tříd

Může vyvstat otázka proč se neomezit pouze na individuální stylování konkrétních vrcholů či hran? Podívejme se na obrázek (Obr. 10), kde můžeme vidět několik skupin vrcholů (v ukázce konkrétně tři), které mají větší počet prvků (pro názornost je uvedeno vždy pouze pět vrcholů v každé skupině). Pokud vytvoříme tři různé grafické třídy a tyto třídy přiřadíme jednotlivým skupinám, můžeme následně velmi snadno měnit grafickou podobu

celé skupiny vrcholů pouze úpravou vlastností vybrané třídy. Bez použití grafických tříd bychom museli nastavovat vlastnosti každého vrcholu samostatně.

### 7.5.1 Dynamické dědění grafických tříd

Integrace podpory grafických tříd umožnila i definování 3 výchozích tříd, od kterých jsou odvozeny všechny další:

- **default** – obsahuje definici všech dostupných vlastností a tvoří základ pro ostatní třídy
- **default:node** – dědí od grafické třídy default a upravuje definici vlastností specifických pro vizuální podobu vrcholů (např. barvu pozadí)
- **default:edge** – dědí od grafické třídy default a upravuje definici vlastností specifických pro vizuální podobu hran (např. styl linie)

Pokud použijeme načtení grafu z textového řetězce nebo nijak nedefinujeme vlastní vzhled vrcholů a hran v XML, tak se automaticky použijí grafické třídy „default:node“ pro vrcholy a „default:edge“ pro hrany.

Výhodou je, že lze tyto třídy libovolně předefinovat a nastavit si tak vlastní výchozí vzhled.

Chceme-li přidat nové grafické třídy, vždy bychom měli nastavit třídu, z níž má ta nová vycházet. Takto vytvořená třída pak převezme všechny vlastnosti od svého předka a v nové třídě budou uchovány pouze požadované odlišnosti.

Tab. 2. Ukázka dědění vlastností grafických tříd

default:node	mojeTrida	potomekMojiTridy
color: <i>black</i>	color: <i>gray</i>	color: <i>gray</i>
background: <i>green</i>	background: <i>green</i>	background: <i>green</i>
size: <i>1</i>	size: <i>1</i>	size: <i>3</i>

Tabulka (Tab. 2) znázorňuje tři různé vlastnosti (color, background a size), které jsou postupně redefinovány v jednotlivých třídách. Pokud byla vlastnost v konkrétní třídě (sloupci) přenastavena, je její název uveden tučně. Jako výchozí je použita třída „default:node“, kde vidíme, že došlo k nastavení všech tří hodnot. Od této výchozí třídy je odvozena třída „mojeTrida“, v níž bylo provedeno nastavení parametru „color“, ostatní hodnoty vycházejí z nadřazené třídy. Jako poslední je „potomekMojiTridy“, kde lze

pozorovat úpravu opět pouze jedné hodnoty „size“ a zbylá konfigurace vychází z nadřazených tříd.

Pokud bychom nyní provedli změnu vlastnosti „background“ v třídě „default:node“, úprava by se projevila také v třídě „potomekMojiTridy“. Naopak upravili-li bychom vlastnost „size“ třídy „default:node“, změna by byla viditelná pouze u vrcholů využívajících třídu „mojeTrida“, ale již ne u těch s nastavením vzhledu dle „potomekMojiTridy“, neb ta má pro tuto vlastnost svoji individuální konfiguraci.

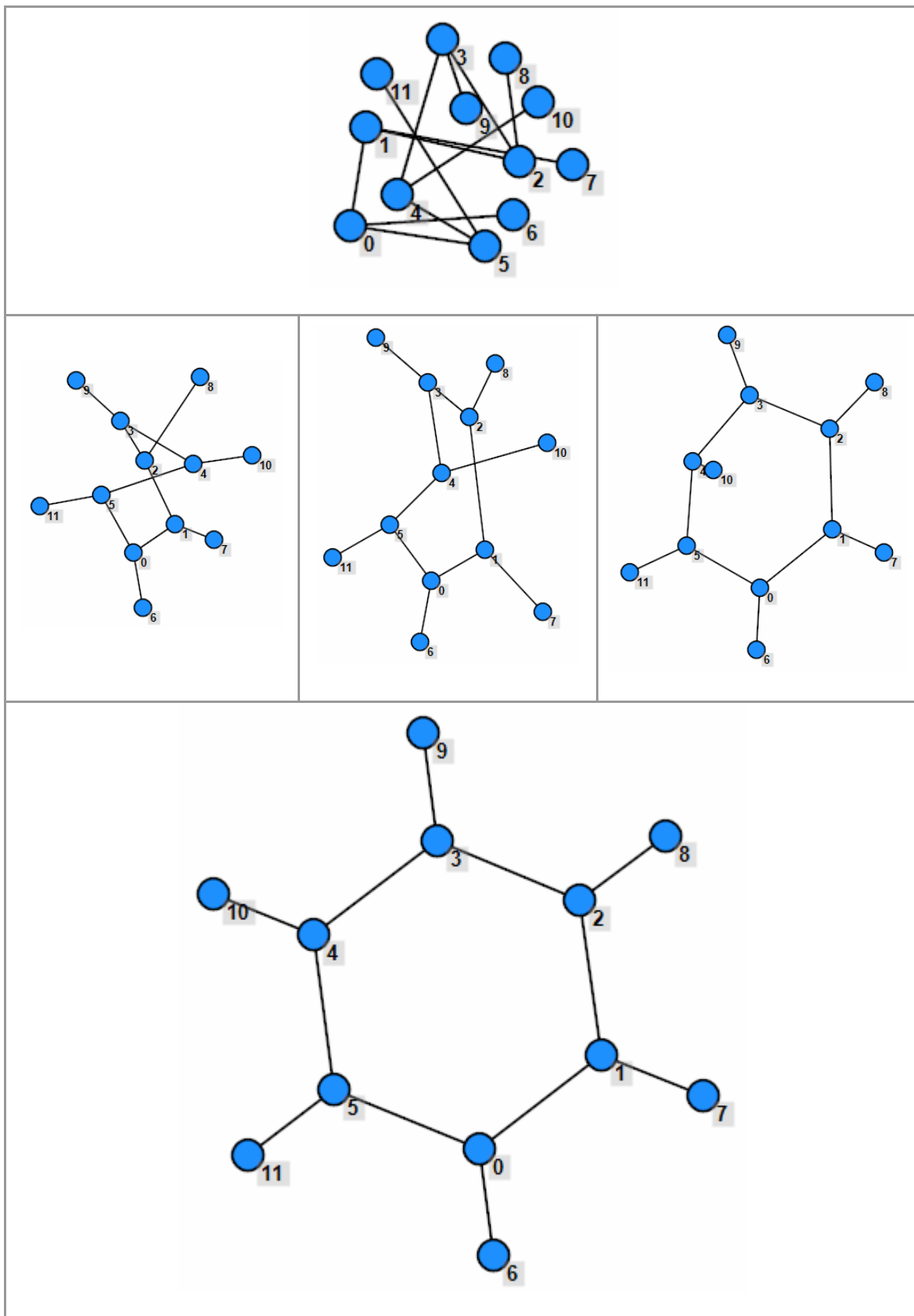
### 7.5.2 Podporované vlastnosti

Podporované vlastnosti pro stylování vrcholů a hran implementované ve třídě *cStyle* jsou společné pro oba typy objektů. V následujícím seznamu můžeme nalézt vždy název vlastnosti, informaci o využívání pro vrcholy nebo hrany a datový typ, který slouží pro vnitřní uchování hodnot dané vlastnosti:

- **color** – vlastnost využívaná u hran i vrcholů, datový typ *Drawing.Color*
- **background** – vlastnost využívaná u vrcholů, datový typ *Drawing.Color*
- **labelcolor** – vlastnost využívaná u vrcholů, datový typ *Drawing.Color*
- **labelbackground** – vlastnost využívaná u vrcholů, datový typ *Drawing.Color*
- **width** – vlastnost využívaná u hran i vrcholů, datový typ *Integer*
- **size** – vlastnost využívaná u vrcholů, datový typ *Integer*
- **labelopacity** – vlastnost využívaná u vrcholů, datový typ *Integer*
- **dashstyle** – vlastnost využívaná u hran i vrcholů, datový typ *Drawing2D.DashStyle*
- **font** – vlastnost využívaná u vrcholů, datový typ *Font*
- **label** – vlastnost využívaná u vrcholů, datový typ *Boolean*
- **byname** – vlastnost využívaná u vrcholů, datový typ *Boolean*

### 7.6 Organizační algoritmus

Tab. 3. Ukázka postupné organizace vrcholů



Jednou z hlavních částí knihovny je algoritmus sloužící k automatické organizaci pozic vrcholů vyskytujících se v grafu.

Jeho konkrétní příklad použití demonstruje tabulka (Tab. 3), v níž je uvedeno pět vizualizací. Mějme graf tvořený 12ti vrcholy označenými celými čísly 0 až 11, které jsou propojeny celkem 12ti hranami. Definice vazeb mezi hranami je dána následujícím předpisem:

„0-1,1-2,2-3,3-4,4-5,5-0,0-6,1-7,2-8,3-9,4-10,5-11“

První vizualizace v tabulce znázorňuje počáteční stav, kdy byla každému vrcholu přiřazena náhodná pozice v souřadném systému. Jak je na první pohled patrné, nelze z tohoto znázornění jednoduše získat představu o struktuře zachyceného grafu.

Následující tři vizualizace znázorňují postupný vývoj organizace vrcholů. Všimněme si rozdílu mezi počáteční a druhou vizualizací. Je zde dobře patrná snaha o excentrické vzdálení okrajových vrcholů od středu grafu. U dalších vizualizací můžeme sledovat postupný vývoj až do stavu úplného ustálení, které je viditelné na posledním, pátém, znázornění.

Rozdíl v názornosti mezi textovým zadáním a grafickým uspořádáním je významný. Vhodná grafická reprezentace přináší značné zjednodušení v pochopení existujících vazeb grafu.

### 7.6.1 Princip organizace

Při určování nových pozic vrcholů využijeme proměnných *netForce* a *velocity* implementovaných v třídě *cNode*, o které bylo napsáno v kapitole 7.1: Vnitřní reprezentace vrcholů.

Základním předpokladem pro správné provedení organizace je náhodné rozmístění vrcholů tak, aby žádný z vyskytujících se vrcholů neměl shodnou pozici  $x$  a  $y$  s jiným.

V hlavním cyklu algoritmu (značme jako A) nejprve procházíme všechny vrcholy. Aktuálně vybranému vrcholu nastavíme parametr *netForce* pro obě souřadnice  $x$  i  $y$  na 0.

V dalším cyklu (značme jako B), který je vnořen do toho hlavního, opět procházíme všemi vrcholy, ale vynecháme ten, který byl vybrán hlavním cyklem.

Součástí hlavního cyklu je ještě jeden cyklus (značme jako C), který opět prochází všechny vrcholy, ale vynechává ty, se kterými neexistuje vazba na vrchol vybraný hlavním cyklem (tzn. není mezi nimi definována hrana).

Tělo cyklu B tvoří následující výpočet, kde  $nodeA$  je vrchol vybraný cyklem A,  $nodeB$  je vrchol vybraný cyklem B a  $distance$  je konstanta ovlivňující délku hran:

$$nodeA.netForce.X += \frac{distance \cdot (nodeA.pos.X - nodeB.pos.X)}{(nodeA.pos.X - nodeB.pos.X)^2 + (nodeA.pos.Y - nodeB.pos.Y)^2}$$

$$nodeA.netForce.Y += \frac{distance \cdot (nodeA.pos.Y - nodeB.pos.Y)}{(nodeA.pos.X - nodeB.pos.X)^2 + (nodeA.pos.Y - nodeB.pos.Y)^2}$$

Výpočet je proveden pro každou ze souřadnic samostatně a můžeme si všimnout, že vždy dochází k přičítání výsledné hodnoty k dané souřadnici proměnné  $netForce$ .

Jak již bylo uvedeno, důležité je v rámci cyklu B vyloučit iteraci, která by proběhla s vrcholem, který je aktuálně načtený cyklem A. Pokud bychom tak neučinili, došlo by ve jmenovateli k odečtení obou souřadnic, což by vedlo na pokus o dělení nulou.

Další cyklus C provede jednoduchou operaci u všech vrcholů, mezi kterými je definována hrana s vrcholem vybraným hlavním cyklem A:

$$nodeA.netForce.X += kons \tan ta \cdot (nodeC.pos.X - nodeA.pos.X)$$

$$nodeA.netForce.Y += kons \tan ta \cdot (nodeC.pos.Y - nodeA.pos.Y)$$

V závěru cyklu A se vypočtené hodnoty z proměnné  $netForce$  vybraného vrcholu promítnou do proměnné  $velocity$ , u které, jak jsme si zatím mohli všimnout, nedochází k žádnému nulování a její stav je tak průběžně ovlivňován při každém spuštění organizačního algoritmu:

$$nodeA.velocity.X = (nodeA.velocity.X + nodeA.netForce.X) \cdot tlumeni$$

$$nodeA.velocity.Y = (nodeA.velocity.Y + nodeA.netForce.Y) \cdot tlumeni$$

Mimo tělo hlavního cyklu je pro každý z vrcholů vypočítána pozice, a to právě přidáním hodnoty přírůstku  $velocity$  vždy pro danou souřadnici:

$$nodeA.pos.X += nodeA.velocity.X$$

$$nodeA.pos.Y += nodeA.velocity.Y$$

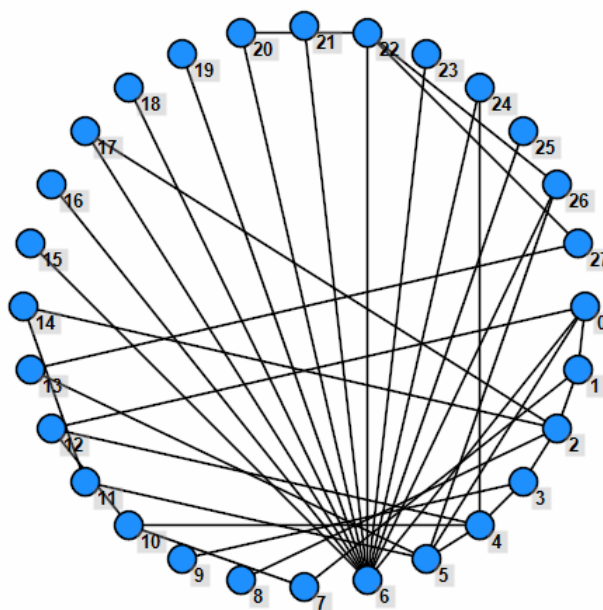
Každým spuštěním tohoto algoritmu dojde k posunu všech bodů o určitý vypočtený přírůstek uchovaný v proměnné *velocity*, která, jelikož je vektorem, určí také směr pohybu.

Chceme-li tedy docílit ideálního rozmístění všech vrcholů grafu, měli bychom vykonat více iterací organizačního algoritmu za sebou. Tento počet vykonání může být konstantní, nebo můžeme pomocí veřejně dostupné vlastnosti *velocity* u objektu vytvořeného dle třídy *Graph* zjistit sumu hodnot parametrů *velocity* nad všemi vrcholy grafu. Čím je tato hodnota menší, tím je graf stabilnější. Po dosažení hodnoty, co nejvíce blíží se nule, lze pak rozhodnout o ukončení organizace.

## 7.7 Kruhové uspořádání

Další variantou, jak vizuálně znázornit vztahy mezi vrcholy grafu, je kruhové uspořádání. Jedná se o vizualizaci, při níž jsou jednotlivé vrcholy rovnoměrně umístěny na pomyslnou kružnici. Hrany jsou vedeny z počátečního do koncového vrcholu přímo uvnitř kruhu.

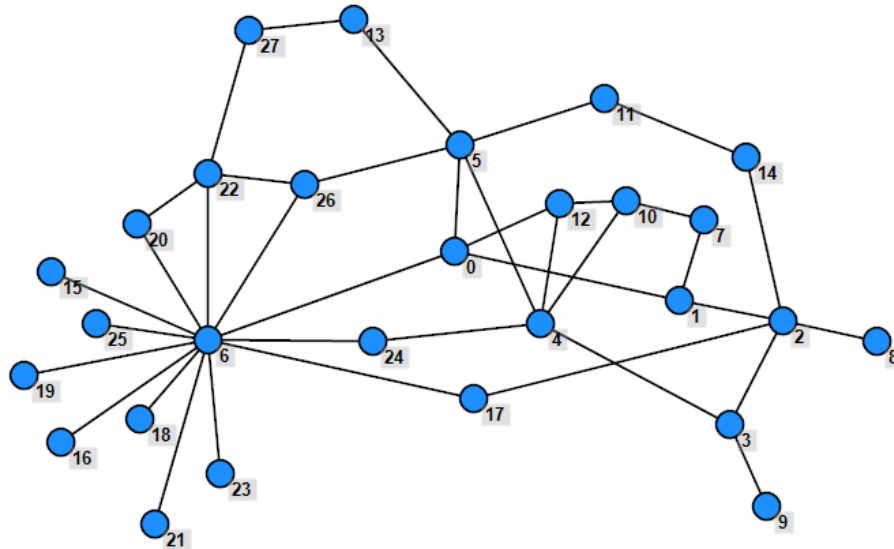
Obrázek (Obr. 11) znázorňuje konkrétní kruhovou vizualizaci tvořenou 28 vrcholy.



Obr. 11. Ukázka kruhového uspořádání

Pokud bychom nad stejným grafem provedli 100 iterací organizačního algoritmu, popsaného v předchozí kapitole, dostali bychom výsledek, který znázorňuje obrázek (Obr. 12).

Pro různé konkrétní případy může být vhodný jeden či druhý algoritmus. Nutno však zmínit, že uspořádání do kruhu je z pohledu výpočetní složitosti mnohem rychlejší. Výpočet pozic vrcholů v tomto případě probíhá v lineární složitosti, což je zásadní výhoda oproti organizačnímu algoritmu. Tento fakt může být v mnoha případech rozhodující.



Obr. 12. Graf s 28 vrcholy

Výpočet pozic při uspořádání do kruhu spočívá v několika málo příkazech. Nejprve určíme krok ve stupních, po kterém budou vrcholy rozmíst'ovány. Následně pak v každé iteraci cyklu, jež proběhne pro všechny vrcholy, provedeme následující výpočty:

$$node.pos.X = radius \cdot \cos\left(angle \cdot \frac{\Pi}{180}\right) + center.X$$

$$node.pos.Y = radius \cdot \sin\left(angle \cdot \frac{\Pi}{180}\right) + center.Y$$

Proměnná *angle* určující úhel na kružnici (s poloměrem nastaveným v proměnné *radius*) musí být postupně inkrementována vždy o předem určený krok. Tím se zajistí rovnoměrné rozprostření všech vrcholů.

## 8 MANUÁL PRO PRÁCI S KNIHOVNOU

Tato kapitola bude pojednávat o možnostech, které vytvořená knihovna nabízí a jak je správně využít. Postupně budou vysvětleny všechny důležité součásti. Vždy bude uváděn zdrojový kód ukázky v jazyce C#. Velmi obdobný by byl i kód pro Visual Basic.

### 8.1 Přidání knihovny do projektu

Aby bylo možné s knihovnou pracovat a využívat její možnosti, je na ni nejprve nutné přidat referenci do projektu.

Nemáme-li vytvořen projekt, vytvoříme jej: File > New project > Windows Form Application. Nový projekt můžeme pojmenovat například jako „GraphExample“.

Přidání knihovny je velmi jednoduché: Project > Add Reference... > v záložce Browse vybereme z umístění na disku soubor „GraphLibrary.dll“ a potvrdíme.

### 8.2 Deklarace proměnné

Pro jednoduchost si vytvoříme privátní proměnnou *graph* přímo v třídě hlavního okna.

```
namespace GraphExample
{
    public partial class Form1 : Form
    {
        private GraphLibrary.Graph graph;
    }
}
```

...

Privátní proměnná *graph* reprezentující objekt vytvořený dle třídy *GraphLibrary.Graph* bude dostupná v rámci celé třídy *Form1*.

### 8.3 Inicializace

Před jakýmkoliv prvním použitím proměnné *graph* je nejprve nutné vytvořit odpovídající objekt. Vytvoření nového objektu provedeme dle třídy *GraphLibrary.Graph* v privátní metodě *Form1\_Load*.

```
private void Form1_Load(object sender, EventArgs e)
{
    graph = new GraphLibrary.Graph();
}
```

...

Metoda *Form1\_Load* se provádí před prvním načtením (zobrazením) hlavního okna. Proměnnou *graph* lze samozřejmě inicializovat i mimo tuto metodu, vždy však před tím, než bude poprvé použita.

## 8.4 Plocha pro vykreslování

Pro grafické znázornění musíme definovat plochu, do které bude knihovna provádět renderování výstupu. Pro tento účel lze zvolit buď samotnou plochu okna formuláře nebo můžeme využít vhodnějšího způsobu použitím komponenty *PictureBox*.

Přetáhněme tuto komponentu na formulář a pojmenujme ji jako „canvas“. Aby byla plocha dostatečně velká, můžeme aktivovat rozprostření po celé ploše formuláře – nastavíme parametr *canvas.Dock* na hodnotu *fill*.

Opět využijeme metody *Form1\_Load*, v níž u objektu „canvas“ provedeme přidání handleru *Paint*:

```
canvas.Paint += new System.Windows.Forms.PaintEventHandler(graph.paint);
```

...

Tím docílíme toho, že pokaždé, když bude požadováno překreslení zobrazovací plochy, bude provedena metoda *graph.paint*, jež zajistí vyrenderování aktuální podoby grafu právě do určené komponenty.

## 8.5 Nastavení grafu

Nastavení týkající se grafu je možné provádět nad proměnnou *graph* prostřednictvím objektu *settings* (navenek jde prakticky o vlastnost), který je vytvářen dle třídy *cPropertyGraph*. Tato třída importuje namespace *System.ComponentModel*, její metody tak mohou být doplněny o přídatné informace. Takto připravený objekt může být přímo předán komponentě *PropertyGrid*, která umožní jeho pohodlnou editaci.

### 8.5.1 Určení středu

Pro určení počátku grafu v souřadném systému slouží vlastnost *center*. Chceme-li nastavit počátek na souřadnice [0, 0] (levý horní roh), použijeme následující kód:

```
graph.settings.center = new Point(0, 0);
```

...

### 8.5.2 Délka hran

Všem hranám grafu je možné globálně nastavit preferovanou délku. Při použití organizačního algoritmu je pak snaha tuto délku co nejvíce dodržovat. Samozřejmě z principu optimální organizace nelze zajistit, aby měly všechny hrany konstantní délku, tento parametr tak slouží hlavně jako určení měřítka této vzdálenosti. Konkrétní příklad použití:

```
graph.settings.distance = 120;
```

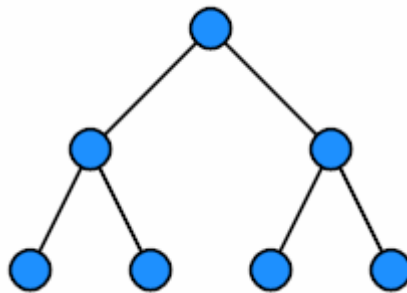
### 8.5.3 Gravitace

U některých typů grafů je potřeba zachytit dominanci kořenového vrcholu, který bývá často umístěn v horní části diagramu a podřazené vrcholy jsou situovány směrem dolů. K tomuto účelu je určena vlastnost *gravitation*.

```
graph.settings.gravitation = GraphLibrary.Graph.gravitationType.vertical;
```

...

Výsledný graf tak může vypadat například takto:



Obr. 13. Stromový graf se zapnutou gravitací

Vlastnost *gravitation* může nabývat tří různých hodnot výčtového typu *GraphLibrary.Graph.gravitationType*:

- **none** – gravitace je vypnutá
- **vertical** – gravitace směrem shora dolů
- **horizontal** – gravitace zleva doprava

### 8.5.4 Změna měřítka

Mnohdy je potřeba znázornit rozsáhlejší strukturu vrcholů, která se při běžném zobrazení jen stěží vejde na viditelnou plochu obrazovky. Díky parametru *scale* lze docílit zmenšení celého výstupu na žádanou hodnotu.

```
graph.settings.scale = 0.8F;
```

...

Jako hodnotu parametru *scale* je možné zadat desetinné číslo v rozmezí 0,1 až 1,0. Hodnota 1,0 odpovídá měřítku 1:1, zadáním čísla 0,5 zajistíme zmenšení na 50% originální velikosti.

## 8.6 Režim vykreslování

Chceme-li zajistit hladší a kvalitnější renderování grafického výstupu, můžeme za pomoci parametru *smoothing* aktivovat tzv. vyhlazovací režim. Tím docílíme pohlednějšího vzhledu u textových popisků a obecně u všech linií. Je-li režim vypnutý, mohou být výsledné objekty zobrazeny kostrbatě.

```
graph.smoothing = true;
```

## 8.7 Načtení grafu

Pro načtení grafu můžeme využít dvě různé metody. Každá z metod nabízí své výhody a jejich výběr vždy záleží na konkrétním případě a potřebách užití. První z nich vyniká svou jednoduchostí, druhá naopak nabízí široké možnosti individualizace.

### 8.7.1 Načtení z textové definice

Potřebujeme-li provést jednoduché, a z pohledu rozsahu definice úsporné, načtení struktury grafu, lze použít metodu *loadByString*.

```
graph.loadByString("A>B,B>C,B=D,D>E,D>F");
```

...

Tento způsob se hodí hlavně v případech, kdy potřebujeme často měnit strukturu grafu bez potřeby provádět individuální nastavení grafické podoby vyskytujících se vrcholů a hran. V takovém případě můžeme nastavit vizuální podobu prvků pouze obecně a dále již se

zabýváme pouze strukturou. O možnostech definice grafického znázornění pomocí k tomu určených metod pojednává sekce 8.10: *Další dostupné možnosti*, metoda *node*.

### 8.7.2 Načtení dle XML

Chceme-li načíst graf včetně definic vizuální podoby vyskytujících se objektů, můžeme použít XML zápis. Metoda podporující vstup proměnné datového typu *System.Xml.Linq.XElement* je nazvaná *load*.

XML máme možnost sestavit přímo do proměnné typu *XElement* nebo jej lze načíst z textového řetězce:

```
String xmlString = "<graph> ... </graph>";
System.Xml.Linq.XElement xml;
xml = System.Xml.Linq.XElement.Parse(xmlString);
graph.load(xml, true);
```

...

Druhým vstupním parametrem metody *load* je možné specifikovat, zda bude pozice vrcholů určena náhodně nebo se uplatní souřadnice z XML.

Definicí podoby XML vstupu se více zabývá kapitola 7.4.2: XML vstup.

## 8.8 Přidání vrcholu

Chceme-li do grafu přidat nový vrchol, využijeme metody *addNode*. Ta má celkem tři parametry. První z nich specifikuje vrchol, ke kterému má být nový vrchol připojen, další definuje jeho unikátní textový identifikátor a poslední je znaková hodnota určující typ hrany.

Metoda *addNode* má dvě varianty. Jedna varianta podporuje zadání prvního parametru jako číselnou hodnotu specifikující index cílového vrcholu (viz kapitola 7.1):

```
graph.addNode(0, "vrcholB", '-');
```

Naopak druhá varianta poskytuje možnost zadat přímo textový identifikátor vrcholu:

```
graph.addNode("vrcholA", "vrcholNovy", '-');
```

V obou případech je druhý a třetí parametr nepovinný. Stačí tedy určit pouze cílový vrchol, ke kterému bude ten nově vzniklý připojen:

```
graph.addNode("vrcholA");
```

## 8.9 Přidání hrany

Nepotřebujeme-li přidávat nový vrchol, ale chceme-li pouze propojit dva již existující, použijeme metodu *addEdge*.

Význam vstupních parametrů je obdobný jako v případě metody *addNode*, pouze v pořadí druhý parametr by měl obsahovat index/identifikátor existujícího vrcholu.

Pro propojení vrcholů ve směru z A do F použijeme následující kód:

```
graph.addEdge("vrcholA", "vrcholF", '>');
```

Třetí parametr specifikující typ hrany je opět nepovinný, při jeho nevyplnění se použije defaultní hrana bez orientace „-“.

## 8.10 Další dostupné možnosti

### **axes**

Potřebujeme-li do grafického výstupu grafu vykreslit osy X a Y, například pro snazší orientaci nebo doladění pozic, nastavíme tomuto přepínači hodnotu True. Obě osy budou následně znázorněny tečkovaně, jejich průsečík bude procházet souřadnicí [0; 0].

### **axesCenter**

Chceme-li zobrazit pouze středový kříž ve středu souřadného systému, v místě, kde se kříží osy X a Y, použijeme přepínač *axesCenter*.

### **circleNode**

Po zavolání této metody bude provedeno pravidelné rozmístění vrcholů na kružnici. Jedná se o alternativní způsob organizace grafu.

### **edge**

Vlastnost *edge* (*get\_edge*) nám zpřístupní objekt konkrétní hrany, jejíž identifikátor předáme jako parametr vlastnosti. Objekt je přizpůsoben pro editaci pomocí komponenty *PropertyGrid*. Přímý přístup k dílčím vlastnostem hrany je samozřejmě také podporován. Tím se nám dostává možnosti si individuálně přizpůsobit podobu každé hrany samostatně. Co se týče grafické podoby, tak schopnost primárně převzít nastavení dle přiřazení grafické třídy je pochopitelně zachována. Vlastnosti, které nejsou nastaveny individuálně, se přejímají.

## **node**

Prostřednictvím vlastnosti *node* (*get\_node*) se nám zpřístupní objekt konkrétního vrcholu, který specifikujeme parametrem vlastnosti. Objekt je podobně jako v případě hrany přizpůsoben pro editaci pomocí komponenty *PropertyGrid*. Přímý přístup k dílčím vlastnostem vrcholu zůstává také podporován, takže stejně jako u hran, tak i u vrcholů můžeme provádět individuální nastavení vizuální podoby nad rámec přiřazené třídy.

## **styleClass**

Do skupiny vlastností, orientovaných na možnost přímé spolupráce s komponentou *PropertyGrid*, patří také *styleClass* (*get\_styleClass*), která zpřístupňuje konfiguraci objektu vybrané grafické třídy.

## **classesNames**

Chceme-li získat seznam všech definovaných grafických tříd, použijeme vlastnost *classesNames*. Datovým typem je vždy seznam textových hodnot, které reprezentují identifikátory existujících grafických tříd.

## **getNodeEdges**

Pro získání identifikátorů všech hran, které jsou propojeny s konkrétním vrcholem, je určena metoda *getNodeEdges*. Nespecifikujeme-li vrchol pomocí volitelného parametru, získáme seznam všech existujících hran v grafu.

## **getXML**

Abychom mohli graf nějakým způsobem uchovat a zachytit jeho aktuální podobu, je zde metoda *getXML*, která provede převedení celé struktury a všech nastavení grafu do předepsané XML formy. Více o návrhu XML je napsáno v kapitole 7.4.2: XML vstup.

## **nodeRename**

Jsou dvě možnosti, jak změnit identifikátor vybraného vrcholu. Lze použít jednak k tomu přímo určenou metodu *nodeRename*, nebo prostřednictvím vlastnosti *id*, kterou nám poskytuje jeho objekt.

## **organize**

Metoda *organize* zpřístupňuje funkcionalitu důležitého algoritmu pro automatické rozmístění vrcholů grafu na co možná nejvhodnější pozice. Jednou aktivací této metody

proběhne jedna iterace organizace, kdy se každý vrchol grafu posune ve směru nejmenšího odporu. Činnost algoritmu je detailně popsána v kapitole 7.6: Organizační algoritmus. Metoda má jeden volitelný parametr, pokud jej nespecifikujeme, je defaultně nastaven na hodnotu 1. Tím je definováno, že má proběhnout právě jedna iterace organizace. Pokud nastavíme například hodnotu 200, provede se přesně tento počet iterací a u většiny základních grafů by měl být výsledek s tímto nastavením již ustálený a dostatečně přehledný.

### **randomizeNodes**

Přesný opak organizace provádí metoda *randomizeNodes*, ta každému vrcholu grafu nastaví náhodnou pozici v souřadném systému. Zohledňuje přitom pravidlo, aby žádné dva vrcholy grafu nebyly umístěny na stejné pozici. Náhodné rozmístění vrcholů je velmi důležitý předpoklad pro provedení automatické organizace.

### **removeNode**

Provede odebrání specifikovaného vrcholu a všech hran, které s ním byly propojeny.

### **selNodeByPosition**

Zejména pro potřeby editoru se hodí metoda *selNodeByPosition*, která dokáže detekovat, zda na předané pozici leží některý z vrcholů grafu. Pokud je nalezen konkrétní vrchol, je návratová hodnota metody (funkce) rovna hodnotě *True*. Dotazovaná pozice se předává jako druhý parametr metody datovým typem *Point*, prvním parametrem je vstupně výstupní proměnná datového typu *Integer*, která v případě úspěšné detekce obsahuje index vrcholu.

### **styleClassCreate**

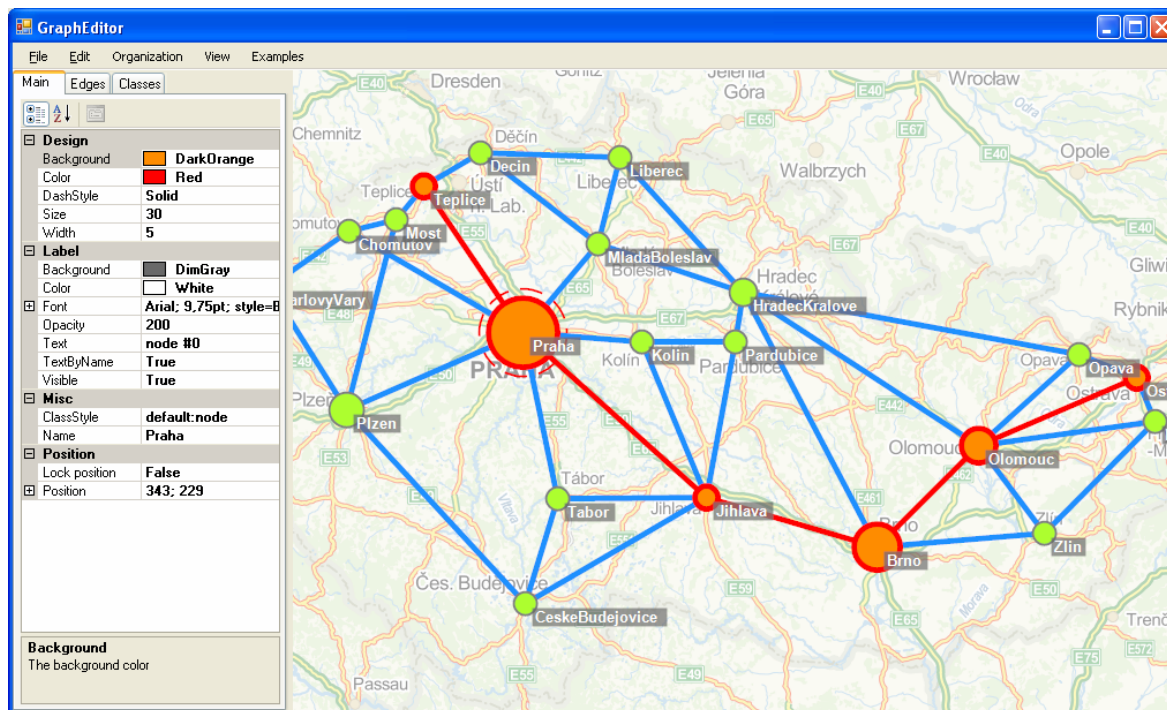
Metoda *styleClassCreate* poskytuje možnost vytvoření nové grafické třídy zadaného názvu, jejíž preference mohou být dále dodatečně nastaveny.

### **velocity**

Chceme-li dynamicky rozhodnout o počtu provedených iterací organizačního algoritmu, můžeme za pomoci této vlastnosti zjistit kvalitu organizace grafu. Čím je návratová číselná hodnota menší, tím je výsledek zpravidla kvalitnější (přehlednější) a ustálenější. V ideálním případě by se mělo toto číslo co nejvíce blížit nule.

## 9 EDITAČNÍ NÁSTROJ

Konkrétním příkladem použití vytvořené knihovny je implementace funkčního editoru grafů s grafickým uživatelským rozhraním. Tento editor poskytuje možnosti a funkcionality knihovny v podobě použitelné i pro běžné uživatele.



Obr. 14. Náhled editoru, mapa ČR [21]

### 9.1 Implementační návrh

Základem aplikace je připravená knihovna „GraphLibrary.dll“, jejímž hlavním principem je vytváření grafického výstupu dle zadaného vstupu. Aby bylo možno nabídnout uživateli patřičný komfort v ovládání, bylo nutné nad tuto knihovnu implementovat další vrstvu, která obstarává odpovídající interaktivní funkcionalitu.

Grafický výstup je knihovnou vykreslován standardně do určené komponenty jako statický obraz. Editor však implementuje rozšíření, kterým dokáže detekovat a následně vizuálně znázorňovat například aktuálně vybraný vrchol nebo efekt při najetí nad objekty grafu.

Vykreslování těchto vizuálních rozšíření probíhá v samostatné vrstvě, která je ve výsledku zkombinována s výstupem z knihovny. Tím je docíleno snadného ovládání, které nabízí pohodlnou manipulaci s vrcholy grafu, přidávání nových vrcholů a vytváření vazeb mezi nimi.

## 9.2 Pracovní plocha

Okno grafického editoru je rozděleno na dvě hlavní části. První důležitou sekcí je editační okno v pravé části obrazovky. Na levé straně je umístěn užší sloupec se třemi záložkami nabízejícími úpravu parametrů u vyskytujících se objektů.

### 9.2.1 Editační okno

Editační okno je centrálním prvkem aplikace. Je to plocha, do kterého probíhá renderování grafického výstupu z knihovny a navíc je rozšířena o podporu interakce.

Mezi její základní možnosti patří:

#### **Vybrání vrcholu**

Pro vybrání konkrétního vrcholu na něj jednoduše klikneme levým tlačítkem myši. Kolem vybraného vrcholu se vždy zobrazuje červená přerušovaná kružnice.

#### **Uzamčení pozice vrcholu**

Potřebujeme-li uzamknout, zafixovat, pozici vrcholu, klikneme na něj pravým tlačítkem myši. Kolem takto uzamčeného vrcholu se zobrazuje šedá tečkovaná kružnice. Uzamčení vrcholu se hodí zejména při používání organizačního algoritmu, kdy jsou takto zafixované vrcholy vyloučeny z procesu reorganizace a zůstávají na svých původních pozicích.

#### **Přesouvání vrcholů**

Změnu pozice existujících vrcholů provedeme pouhým uchopením vrcholu a přetažením na novou pozici.

#### **Přidávání vrcholů**

Chceme-li do grafu přidat nový vrchol, nejprve označíme vrchol, ke kterému má být připojen, stiskneme klávesu Ctrl a kurzorem myši vybereme pozici, kde má být nový vrchol vložen. Je-li stisknuta klávesa Ctrl, tak se pro názornost červeně zobrazuje nová hrana vedoucí z počátečního vrcholu ke kurzoru myši (k místu, kde po kliknutí vznikne nový vrchol).

#### **Vytváření hran**

Podobně jako při vytváření nového vrcholu, tak i v případě definice nové hrany, vybereme nejprve počáteční vrchol, stiskneme klávesu Ctrl a klikneme na vrchol, ke kterému má být

počáteční vrchol připojen. Opět je viditelné označení červené spojnice při tiknutí klávese Ctrl.

### Posun s plátnem

Posun po ploše plátna lze provést jednoduše kliknutím do plochy a tažením myši v žádaném směru.

### Změna oddálení

Pro změnu měřítka grafu použijeme kolečko myši. Pohybem směrem dolů dochází ke zmenšení, pohybem nahoru ke zvětšení. Maximální podporované měřítko je 1:1, minimální oddálení je proveditelné do velikosti desetiny původního rozměru.

### Pomocná mřížka

Často se nám může stát, že potřebujeme vrcholy grafu pravidelně rozmístit na pomyslné přímký, které jsou vzájemně kolmé. Právě k tomuto účelu je určena funkce mřížky, při jejíž aktivaci je dovoleno měnit pozice vrcholů pouze do průsečíků těchto pomyslných přímek. Mřížka je znázorněna formou teček.

## 9.2.2 Editor vlastností

V levé části se nachází sloupec se třemi záložkami.

První záložka „**Main**“ nabízí editaci dvou různých typů prvků. Klikneme-li do prostoru grafu, zobrazí se nabídka týkající se nastavení celého grafu. Je tak možné nastavit barvu pozadí plochy, pozici středu grafu, předdefinovanou délku hran (při použití organizačního algoritmu), zapnutí nebo změnu směru gravitace, či měřítko.

Naopak klikneme-li na některý z vrcholů, zobrazí se nabídka týkající se vybraného vrcholu. Dostupné hodnoty jsou dle příbuznosti rozděleny do čtyř skupin:

- **Design** – barva pozadí vrcholu; barva ohraničení; typ čáry ohraničení; velikost kruhu; tloušťka čáry ohraničení
- **Label** – barva pozadí popisku; barva textu popisku; nastavení fontu; průhlednost pozadí popisku; zobrazovaný text; určení, zda se má jako text použít identifikátor vrcholu; zapnutí nebo vypnutí zobrazování popisku
- **Misc** – výběr rodičovské grafické třídy; změna unikátního identifikátoru vrcholu

- **Position** – uzamčení nebo odemčení pozice vrcholu (fixace); určení pozice x a y

Druhá záložka „**Edges**“ poskytuje souhrnný přehled všech vyskytujících se hran v grafu. Respektive klikneme-li do prostoru grafu, zobrazí se seznam všech hran, klikneme-li ale na konkrétní vrchol, zobrazí se pouze hrany, které s ním souvisí. Vybráním kterékoli hrany ze seznamu získáme nabídku jejího nastavení. Provádět lze úpravu barvy, stylu čáry, tloušťky, orientace hrany nebo výběr rodičovské grafické třídy.

Třetí záložka „**Classes**“ znázorňuje definované grafické třídy. Jejich vzájemná provázanost je do menu zachycena formou stromové struktury. Opět podobně jako při editaci hran, tak i zde se po kliknutí na konkrétní název třídy zobrazí seznam preferencí, které lze měnit.

Jelikož byly grafické třídy implementovány jako obecné objekty schopné uchovávat definovaná nastavení, a to bez ohledu na to, zda jsou pak použity pro stylování vrcholů nebo hran, tak je výčet dostupných vlastností sjednocením množin vlastností vyskytujících se u vrcholů i hran.

Důležitou editovatelnou položkou je volba rodičovské třídy. Změna tohoto nastavení se projeví také přemístěním třídy v rámci stromové struktury menu. Úprava hodnoty jakékoli vlastnosti se samozřejmě okamžitě zohlední u všech odpovídajících objektů na výsledné vizualizaci v editačním okně.

## 10 PŘÍKLADY POUŽITÍ

Předmětem této kapitoly je prezentovat rozličné možnosti využití připravené knihovny pro vizualizaci grafů. Lze vymyslet nespočet případů vhodného použití, ukážeme se však pouze několik připravených ukázek.

### 10.1 Vytvoření grafu z XML

Základním způsobem použití knihovny je vykreslení vizuální podoby grafu z XML souboru, který udává předpis jednak pro definici vazeb vyskytujících se vrcholů a jednak také specifikuje vizuální podobu figurujících objektů.

Vstupní XML soubor sestavený v souladu s vytvořeným návrhem:

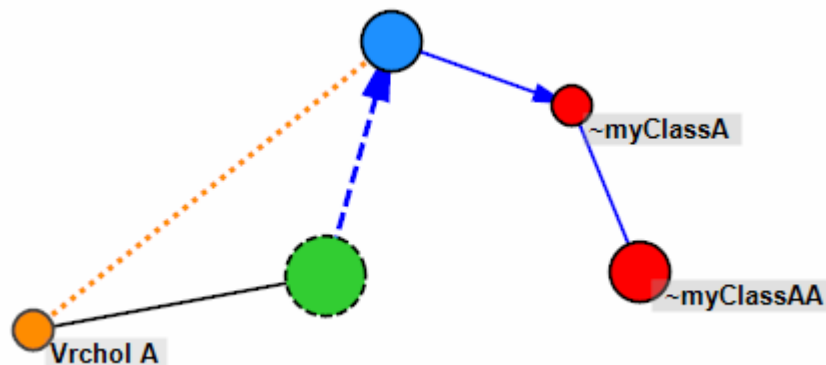
```
<?xml version="1.0" encoding="utf-8" ?>
- <graph>
- <settings>
  <setting name="center" x="248" y="287" />
  <setting name="background" value="White" />
  <setting name="distance" value="200" />
  <setting name="gravitation" value="none" />
  <setting name="scale" value="100" />
</settings>
- <classes>
  <class id="default" style="width:2; color:White; background:Red; dashstyle:Solid; font:Arial/
    9,75pt/ style=Bold; size:10; label:True; byname:False; labelcolor:OrangeRed;
    labelbackground:DimGray; labelopacity:75" />
  <class id="default:node" class="default" style="color:Black; background:DodgerBlue; label:False" />
  <class id="default:edge" class="default" style="color:Blue; dashstyle:Solid" />
  <class id="myClassA" class="default:node" style="background:Red; labelcolor:Black; label:True" />
  <class id="myClassAA" class="myClassA" style="size:15" />
</classes>
- <nodes>
  <node id="A" x="-31" y="51" style="size:10; background:DarkOrange; labelcolor:Black;
    labelbackground:DimGray; labelopacity:60; color:#404040; label:True">Vrchol A</node>
  <node id="B" x="148" y="-93" style="label:False; size:15" />
  <node id="C" x="115" y="24" style="background:LimeGreen; dashstyle:Dash; size:20; width:2" />
  <node id="D" x="238" y="-61" class="myClassA">~myClassA</node>
  <node id="E" x="272" y="22" class="myClassAA">~myClassAA</node>
</nodes>
- <edges>
  <edge value="A-B" style="dashstyle:Dot; color:DarkOrange; width:3" />
  <edge value="C-A" style="width:2; color:Black" />
  <edge value="C>B" style="width:3; dashstyle:Dash" />
  <edge value="D<B" />
  <edge value="E-D" />
</edges>
- <editor>
  <setting name="auto" value="false" />
  <setting name="speed" value="normal" />
  <setting name="image" value="" />
</editor>
</graph>
```

Jak můžeme vyčíst, tak celkem je definováno pět grafických tříd. Tři z nich jsou redefinice výchozích tříd (*default*, *default:node*, *default:edge*), dvě jsou uživatelské (*myClassA*, *myClassAA*).

Graf obsahuje pět různých vrcholů nazvaných identifikátory A, B, C, D a E. První tři využívají pro svou vizuální prezentaci standardní grafickou třídu (*default:node*; není třeba specifikovat, uplatní se automaticky) a jejich individuální podoba je upřesněna pomocí atributů *style*. Vrcholy D a E mají naopak přiřazeny navzájem příbuzné třídy *myClassA* a *myClassAA*. Jednoduše lze vyčíst, že třída *myClassA* vychází z defaultní třídy *default:node* a je rozšířena o definici barvy pozadí a nastavení popisku. Třída *myClassA* je pak dále předkem další třídy *myClassAA*, ta upravuje pouze jediný parametr týkající se velikosti uzlu.

Existující uzly jsou propojeny pěti vazbami definovanými v sekci *edges*. U prvních tří vazeb můžeme pozorovat individuální stylování pomocí atributu *style*, vizuální podoba všech vazeb však vychází z předdefinované grafické třídy *default:edge*.

Finální výstup odpovídající popsanému vstupnímu XML, které bylo zpracováno a následně vyrenderováno knihovnou, vypadá následovně:



Obr. 15. Výstup dle ukázkového XML

Jak je vidět, finální podoba grafu může být poměrně snadno a přehledně individuálně upravena. Vrcholy i hrany mohou být stylovány samostatně nebo lze společné vlastnosti zastřešit pod uživatelsky definované grafické třídy, které nabízejí podporu dynamického dědění používaných parametrů.

## 10.2 Příklad použití knihovny v jazyce C#

Následující zdrojový kód, napsaný v jazyce C#, demonstruje použití knihovny pro vizualizaci grafu zadaného textovou definicí. Je demonstrováno jak předefinování globálního nastavení vzhledu vrcholů a hran, tak také individuální přizpůsobení pro konkrétní použité objekty (vrchol A, hrana mezi vrcholy A a B).

```
// privátní proměnná
private GraphLibrary.Graph graph;
...

// vytvoření objektu
graph = new GraphLibrary.Graph();

// zapne vyhlazování
graph.smoothing = true;

// načte definici grafu
graph.loadByString("A>B,B>C,D-B");

// provede globální nastavení pro všechny vrcholy
GraphLibrary.cPropertyStyle nodes = graph.get_styleClass("default:node");
nodes.labelTextByName = true;
nodes.labelColor = Color.Black;
nodes.width = 2;
nodes.color = Color.Black;

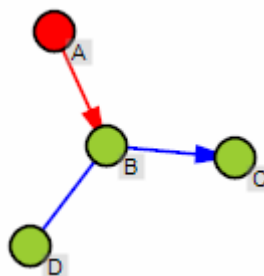
// provede globální nastavení pro všechny hrany
GraphLibrary.cPropertyStyle edges = graph.get_styleClass("default:edge");
edges.width = 2;

// individualizace vzhledu konkrétního vrcholu a hrany
graph.get_node("A").background = Color.Red;
graph.get_edge("A>B").color = Color.Red;

// provede 50 iterací automatické organizace
graph.organize(50);

// přidání události pro renderování v zobrazovací oblasti
canvas.Paint += new System.Windows.Forms.PaintEventHandler(graph.paint);
```

Uvedenému zdrojovému kódu odpovídá následující grafický výstup:



Obr. 16. Výstup ukázkového kódu v jazyce C#

### 10.3 Příklad použití knihovny v jazyce Visual Basic

Následující zdrojový kód, napsaný v jazyce Visual Basic, je adekvátní příkladu popsanému v předchozí kapitole. Posloupnost příkazů je totožná, můžeme sledovat pouze drobné zjednodušení v syntaktickém zápisu.

```
'privátní proměnná
Private graph As GraphLibrary.Graph
...

'vytvoření objektu
graph = New GraphLibrary.Graph()

'zapne vyhlazování
graph.smoothing = True

'načte definici grafu
graph.loadByString("A>B,B>C,D-B")

'provede globální nastavení pro všechny vrcholy
Dim nodes = graph.styleClass("default:node")
nodes.labelTextByName = True
nodes.labelColor = Color.Black
nodes.width = 2
nodes.color = Color.Black

'provede globální nastavení pro všechny hrany
Dim edges = graph.styleClass("default:edge")
edges.width = 2

'individualizace vzhledu konkrétního vrcholu a hrany
graph.node("A").background = Color.Red
graph.edge("A>B").color = Color.Red

'provede 50 iterací automatické organizace
graph.organize(50)

'přidání události pro renderování v zobrazovací oblasti
AddHandler canvas.Paint, AddressOf graph.paint
```

Rozdíl v syntaxi je patrný zejména ve způsobu deklarace proměnných, absenci středníků na konci řádků a v celkovém zjednodušení zápisu. Například přístup k objektu vybraného vrcholu se v jazyce Visual Basic realizuje přímo přes vlastnost *node*, přičemž v jazyce C# je objekt dostupný přes metodu *get\_node*. Odlišné je také provázání události *paint* s vykreslovací komponentou.

Oba zápisy však ve výsledku samozřejmě poskytují totožnou funkcionalitu a shodný grafický výstup prezentace žádaného grafu.

Jednotlivé vrcholy a vazby mohou být pochopitelně vytvářeny také dynamicky za běhu programu. Níže uvedený zdrojový kód (byly vynechány některé části společně s předchozí ukázkou) demonstruje přidání deseti nových vrcholů, které jsou připojeny k existujícímu vrcholu A.

...

```
'načte základní definici grafu
graph.loadByString("A-0")

'k vrcholu A připojí dalších 10 nových vrcholů
For i As Integer = 1 To 10
    graph.addNode("A", i.ToString)
Next

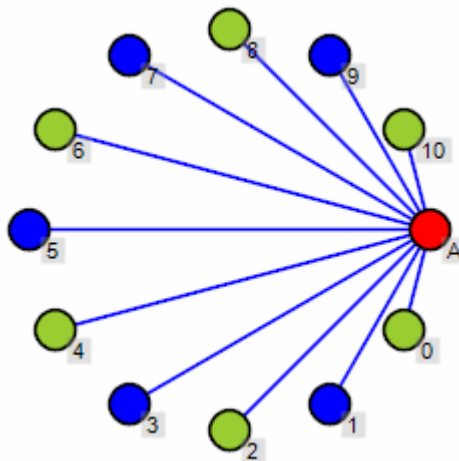
'individualizace vzhledu konkrétního vrcholu
graph.node("A").background = Color.Red

'každému druhému vrcholu nastaví modré pozadí
For i As Integer = 1 To 10 Step 2
    graph.node(i.ToString).background = Color.Blue
Next

'provedeme uspořádání vrcholů do kruhu
graph.circleNodes()

'přidání události pro renderování v zobrazovací oblasti
AddHandler canvas.Paint, AddressOf graph.paint
```

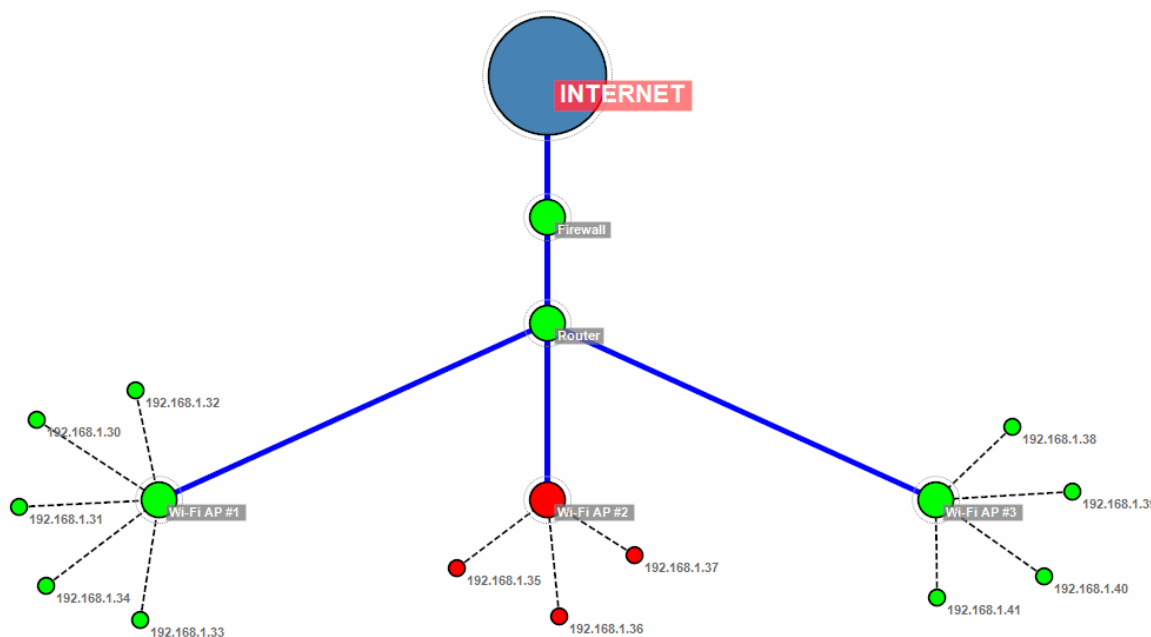
Zde k nahlédnutí odpovídající grafický výstup:



Obr. 17. Výstup ukázkového kódu v jazyce  
Visual Basic

## 10.4 Interaktivní znázornění topologie sítě

Knihovna může být jednoduše použita k vytvoření jednoduché interaktivní vizualizace znázorňující aktuální stav interní sítě. Kromě znázornění stavu jednotlivých aktivních prvků a propojení (firewall, router, přístupové body Wi-Fi, atp.), lze v reálném čase zachytit také samotná koncová zařízení, která jsou průběžně k přístupovým bodům připojována. Reálně je připojení nového klienta vizualizováno jako vznik nového uzlu, který se připojí k některému z přístupových bodů. Díky aktivní automatické organizaci vrcholů a tomu, že jsou vrcholy znázorňující aktivní síťové prvky na svých pozicích zafixovány, jsou vždy klienti připojení k danému přístupovému bodu rozmísťováni rovnoměrně do logických skupin.



Obr. 18. Interaktivní topologie sítě

Jak je vidět, tak obrázek (Obr. 18) znázorňuje jednoduchou síť, která díky třem přístupovým bodům Wi-Fi poskytuje bezdrátové připojení dílčím klientům. Aktuálně můžeme sledovat výpadek přístupového bodu nazvaného „Wi-Fi AP #2“ a tím také nedostupnost tří k němu připojených klientů. Tento stav by v zápětí způsobil, na vizualizaci graficky viditelný a animovaný, přesun těchto klientů k některým z dostupných přístupových bodů.

Takové interaktivní schéma je možné jednoduše realizovat pro libovolné síťové prvky. Ideálně pokud jsou komponenty postaveny například na systému UNIX a nabízejí vzdálený

přístup přes některý ze standardních protokolů (ssh, telnet, atp.). Problémem není propojení ani s cenově levnějšími zařízeními, která jsou primárně určena pro domácí použití a nabízejí například pouze webovou administraci. Důležité je vždy zajistit načtení aktuálního stavu zařízení, případně seznamu připojených klientů. Pro nejjednodušší a nejobecnější použití stačí knihovnu propojit se systémovým příkazem *ping*, který je vykonáván cyklicky v definovaných intervalech.

## 11 VYUŽITÍ KNIHOVNY PRO VIZUALIZACI ALGORITMU ANALYTICKÉHO PROGRAMOVÁNÍ

Jedním z hlavních účelů použití knihovny bylo její propojení s algoritmem analytického programování. I přesto, že práce není zaměřena na oblast umělé inteligence a implementace metod analytického programování, je vhodné zmínit alespoň několik základních informací týkající se této problematiky.

Analytické programování není vyhraněno na konkrétní algoritmus, lze využít libovolný evoluční algoritmus. Například diferenciální evoluci nebo samoorganizující se migrační algoritmus, známý pod označením SOMA (*anglicky Self-Organizing Migrating Algorithm*). [20]

SOMA se vyznačuje zejména tím, že při běhu algoritmu se nevytvářejí noví jedinci, probíhá pouze jejich přemísťování v určeném prostoru. K tzv. samo organizaci dochází při vzájemném ovlivňování se jedinců.

Celkově je analytické programování z části inspirováno generickým programováním a numerickými metodami. V generickém programování se inspiruje evolučním vytvářením symbolických řešení. Idea funkcionálního prostoru a konstrukce výsledné funkce pak vychází z numerických metod, konkrétně z Hilbertových prostorů.

Analytické programování se pokouší syntetizovat přijatelná řešení. Využívá k tomu následující matematické objekty:

- funkce
- operátory (+, -, \*, /, ...)
- terminály (konstanty, nezávislé proměnné, ...)

Výsledkem konkrétní interpretace analytického programování je například následující výstupní řetězec, programově reprezentováno jako hodnota uložená v proměnné datového typu String.

Textová reprezentace:

$$AN[ AN[x] + AN[x] + AN[x] + AN[ x + AN[ x + AN[ AN[x] + x + AN[x] ] ] ] + x + AN[x] + AN[x] + AN[ x + AN[x] ]$$

Vytvořená knihovna nabízí přímo metodu určenou ke zpracování takového vstupu. Po načtení grafu prostřednictvím metody *loadByStringAP* se interně provede překlad vstupní hodnoty a proběhne také počáteční nastýlování hlavních objektů X a Y. Vrcholům AN je přednastaven jednotný styl. Následně je vhodné provést automatickou organizaci vrcholů, níže v ukázce se vykoná konkrétně 200 iterací. Pro názornější zobrazení slouží metoda *autoScale*, která dle zadaných hodnot (šířka, výška a okraj) zajistí ideální rozmístění výstupu do viditelné oblasti zobrazovacího prvku.

...

```
// proměnná obsahující výstup z algoritmu AP
String data = "AN[ AN[x] + AN[x] + AN[x] + AN[ x + AN[ x + AN[ AN[x] ...

// vytvoření objektu
graph = new GraphLibrary.Graph();

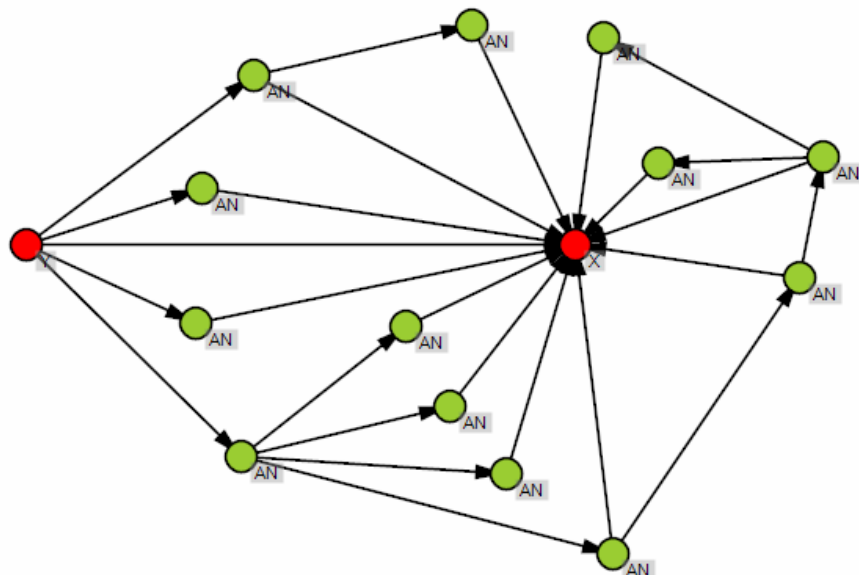
// načtení dat do grafu
graph.loadByStringAP(data);

// provedení organizace
graph.organize(200);

// upravení velikost vizualizace
graph.autoScale(canvas.Width, canvas.Height, 30);
```

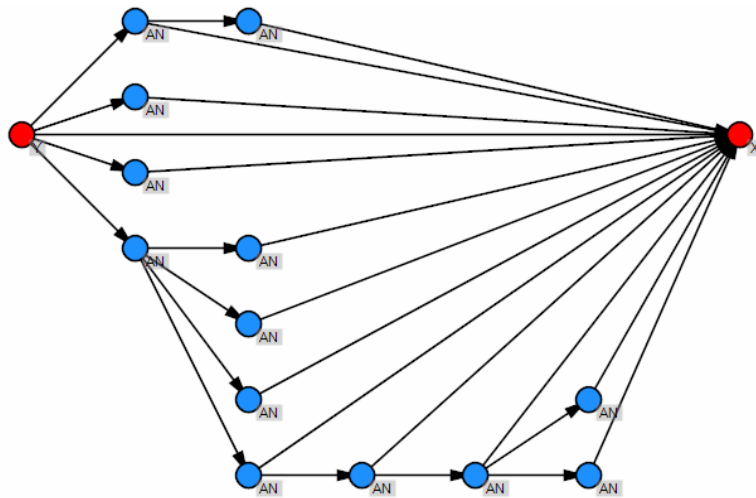
...

Uvedenému vstupu odpovídá následující grafická vizualizace:



Obr. 19. Vizualizace výstupu AP

Pomocí editoru lze vizualizaci přizpůsobit, například do následující podoby:



Obr. 20. Upravená vizualizace výstupu AP

Metoda `loadByStringAP` nabízí také možnost zpracování pole více výrazů najednou. Díky tomu lze využívat i výrazy zadané ve tvaru „ANNX = výraz“, kde X je identifikátor dotvářející klíč, podle kterého jsou jednotlivé výrazy odlišeny. Takový klíč může být dále využit v následujících definicích, konkrétně například takto:

$$ANN0 = x + AN[x]$$

$$ANN1 = ANN0 + AN[ANN0] + ANN0$$

$$ANN2 = ANN0 + ANN1 + AN[ANN1]$$

Adekvátní zápis ve zdrojovém kódu:

```
...
// vytvoření objektu
graph = new GraphLibrary.Graph();

// připravení seznamu
List<String> data = new List<String> { };

// postupné přidání výrazů
data.Add("ANN0 = x + AN[x]");
data.Add("ANN1 = ANN0 + AN[ANN0] + ANN0");
data.Add("ANN2 = ANN0 + ANN1 + AN[ANN1]");

// načtení dat do grafu
graph.loadByStringAP(data.ToArray());

// provedení organizace
graph.organize(200);

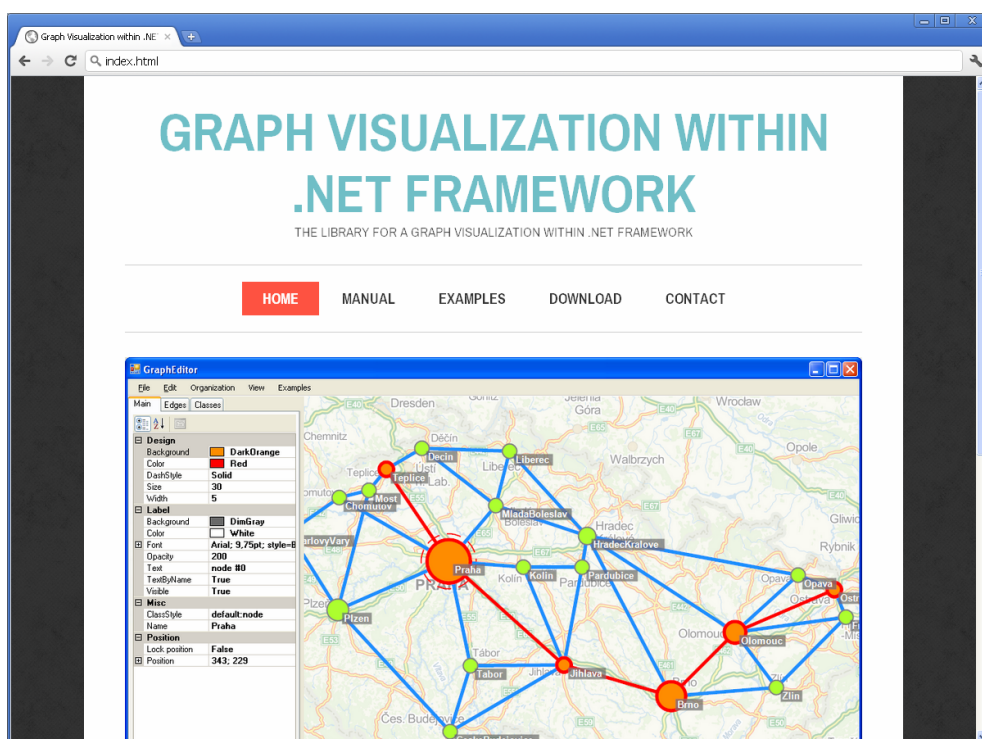
// upravení velikost vizualizace
graph.autoScale(canvas.Width, canvas.Height, 30);
...
```

## 12 PREZENTACE V PROSTŘEDÍ INTERNETU

Za účelem prezentace výsledné knihovny a jejího zpřístupnění široké veřejnosti byla vytvořena jednoduchá webová stránka.

Webová prezentace přehlednou formou poskytuje základní informace týkající se knihovny a možností, které jsou nabízeny.

Část týkající se stručného manuálu, ale i sekce zaměřená na publikaci ukázkových příkladů, mohou být dále průběžně doplňovány a rozvíjeny.



Obr. 21. Náhled webové prezentace

## ZÁVĚR

Diplomová práce se v úvodu zabývá zhodnocením existujících řešení sloužících k vizualizaci grafů. Dále se v obecné rovině seznamujeme s fyzikálními základy klasické mechaniky, které jsou prakticky využity v algoritmu sloužícímu k automatické organizaci vrcholů grafu. Samotný algoritmus je pak součástí implementace vytvořené dynamické knihovny, která poskytuje všechny potřebné funkcionality tak, aby mohla být reálně využita. Demonstrací možností knihovny a logickým vyústěním je samostatná aplikace - editor s grafickým uživatelským rozhraním nabízející schopnost navrhovat, vytvářet a upravovat nejrůznější grafy. Díky jednoduchému prostředí lze tak během chvíle do grafu zachytit například síť evropských měst dle předlohy umístěné na pozadí pracovní plochy.

Jak vyplývá z dalších konkrétních prezentovaných ukázek užití vytvořené knihovny, tak její rozhraní a rozsah nabízených možností je dostatečný pro nejrůznější aplikace bez omezení na konkrétní obor.

Pro definici struktury grafu byly do knihovny implementovány tři různé metody vstupu. Nejkomplexnější je varianta schopná zpracovat XML strukturu odpovídající navrženému předpisu. Tímto způsobem lze grafu specifikovat kompletní nastavení pro existující vrcholy a vztahy mezi nimi, nebo provést konfiguraci grafu jako celku, a to se všemi jeho dostupnými parametry. Naopak nejjednodušší varianta vstupu nabízí sestavení struktury grafu z pouhého textového řetězce tvořeného z názvů dvojic propojených vrcholů, které jsou odděleny čárkami. Třetí možností je zpracování výstupu z algoritmu analytického programování ve formě funkcionálního výrazu.

Kromě variabilních možností vstupu byla pozornost věnována také návrhu logiky stylování vykreslovaných vrcholů a hran. Dostupné jsou dva základní typy stylování, prvním z nich je forma přímé definice stylu u vybraného objektu, druhým typem je varianta přiřazení obecné grafické třídy. Možnost vytvářet vlastní grafické třídy, nebo předefinovávat ty výchozí, nabízí, i díky implementované podpoře pro dynamické dědění vlastností těchto tříd, značné zjednodušení nejen při hromadné úpravě vizuálních parametrů.

Dostupné ukázky demonstrují jednoduchost a intuitivnost použití knihovny v dvou různých programovacích jazycích. Nezávislost knihovny na konkrétním programovacím jazyce přináší potenciál jejího nasazení v různých projektech postavených na platformě Microsoft .NET Framework. V příkladech můžeme sledovat porovnání jazyků C#

a Visual Basic .NET. Principiálně jde o totožnou logiku použití, odlišnosti jsou patrné pouze v úspornosti a skladbě syntaxe.

Případný budoucí vývoj knihovny by se mohl týkat implementace dalších typů automatických organizačních algoritmů, případně zdokonalování a optimalizace těch již připravených. Prostor pro dodatečné rozšíření je také v možnostech stylování a individualizace vzhledu vrcholů a hran. Zde lze na navrženém základu a nastoleném principu implementace provést rozšíření prakticky o neomezený počet parametrů, které mohou dle potřeb sloužit k ještě rozsáhlejším možnostem vzájemné diverzifikace vyskytujících se objektů.

## ZÁVĚR V ANGLIČTINĚ

In introduction, the thesis reviews existing programs regarding graph visualization. Following part generally describes physical laws of classical mechanics, which are utilized in the algorithm for automatic vertex organization. The algorithm itself is then included in the implementation of the library that provides all possible functionality so it can be used in real environment. Standalone application with graphical interface and support for designing, creating and editing arbitrary graphs is created to demonstrate all possible use-cases of this library.

As follows from other presented use-cases of the library, its interface and range of implemented functionalities is sufficient for creating various applications without restrictions in field.

Three different inputs for graph structure definition were implemented into the library. The most complex variant is capable of processing XML that equals to the corresponding graph. In this manner, it is possible to specify complex configuration for existing vertices and relations between them, or design whole graph as a unit. On contrary, the easiest input option offers graph definition as simple as plaintext that is compiled from commas separated pairs of linked vertices. Third and final option provides the ability to process an output from algorithm of analytics programming that is represented as a functional expression.

Apart from offering wide spectrum of input options, attention to styling of graph's vertices and edges was also paid. The results are two basic methods – direct definition of graphical attributes in the selected graph object or assignment of a graphics class. This method of creating graphical classes or redefining the default ones offers considerable simplification in bulk editing of the visual parameters, mainly thanks to the support of inheritance in class definition.

Simplicity and intuitiveness of this library is demonstrated in the available source code examples, which are written in two different programming languages. Independence from the programming language brings potential to use the library in various projects built on Microsoft .NET Framework. Specifically, comparison of C# and Visual Basic .NET can be seen in the provided examples. As shown, fundamental logic is independent on chosen language. The differences can be spotted only in the syntax structure and efficiency.

Possible future development could regard other algorithms of automatic graph organization, or improve and optimize those already implemented. There is also space for additional improvements in styling and rendering of vertices and edges. Thanks to designed then embedded library core, extensions of infinite number of various parameters can be implemented any time. Those parameters can further diversify occurring objects.

**SEZNAM POUŽITÉ LITERATURY**

- [1] XIONG, Bin a Zhongyi ZHENG. Graph theory. Shanghai: East China Normal University Press, c2010, ix, 146 s. Mathematical olympiad series (World Scientific Publishing). ISBN 978-981-4271-12-7.
- [2] CLARK, John. A first look at graph theory. Singapore: World Scientific, 1996, xiv, 330 s. ISBN 98-102-0490-6.
- [3] NASH-WILLIAMS, W.T. Tutte; foreword by Crispin St. J.A. Graph theory. Transferred to digital print. Cambridge: Cambridge University Press, 2001. ISBN 05-217-9489-7.
- [4] Hrana (graf). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-03-17]. Dostupné z: [http://cs.wikipedia.org/wiki/Hrana\\_\(graf\)](http://cs.wikipedia.org/wiki/Hrana_(graf))
- [5] MATOUŠEK, Jiří a Jaroslav NEŠETŘIL. Kapitoly z diskretní matematiky. 2. upr. vyd. Praha: Karolinum, 2000, 377 s. ISBN 80-246-0084-6.
- [6] BONDY, J a U MURTY. Graph theory with applications. 2. upr. vyd. New York: North Holland, 1976/1979, x, 264 p. ISBN 04-441-9451-7.
- [7] PIRNOT, Thomas L. Mathematics all around. Boston: Addison Wesley, 2001. ISBN 978-020-1308-150.
- [8] ČADA, Roman, Tomáš KAISER a Zdeněk RYJÁČEK. Diskretní matematika. 1. vyd. Plzeň: Západočeská univerzita v Plzni, 2004, 170 s. ISBN 80-708-2939-7.
- [9] CORMEN, Thomas H. Introduction to algorithms: Representations of graphs. 2nd ed. Cambridge: MIT Press, c2001, xxi, 1180 s. ISBN 02-620-3293-7.
- [10] YFiles.NET: Graph Layout and Visualization Library. [online]. [cit. 2013-04-09]. Dostupné z: [http://www.yworks.com/en/products\\_yfilesdotnet\\_commercialinfo\\_priceseur.html](http://www.yworks.com/en/products_yfilesdotnet_commercialinfo_priceseur.html)
- [11] Gephi: Features. [online]. [cit. 2013-03-21]. Dostupné z: <https://gephi.org/features/>
- [12] LIBERTY, Jesse. Programming .NET 3.5. Sebastopol: O'Reilly, 2008, xvii, 455 s. ISBN 978-0-596-52756-3.

- [13] C# Graphics Programming. Hoboken: John Wiley, 2010. ISBN 978-111-8035-504.
- [14] RICHTER, Jeffrey. .NET Framework: programování aplikací. 1. vyd. Překlad Jiří Hynek. Praha: Grada, 2002, 552 s. ISBN 80-247-0450-1.
- [15] MOELLER, Gunther Lenz and Thomas. .NET: a complete development cycle. 1. vyd. Překlad Jiří Hynek. Boston: Pearson Education, 2003, 552 s. ISBN 03-211-6882-8.
- [16] Microsoft .NET Framework: .NET Downloads, Developer Resources & Case Studies. [online]. 2013 [cit. 2013-04-10]. Dostupné z: <http://www.microsoft.com/net/>
- [17] .NET. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013 [cit. 2013-04-08]. Dostupné z: <http://cs.wikipedia.org/wiki/.NET>
- [18] BARTUŠKA, Karel a Emanuel SVOBODA. Fyzika pro gymnázia: molekulová fyzika a termika. 4. přeprac. vyd. Praha: Prometheus, 2000, 244 s. ISBN 80-719-6200-7.
- [19] ESPOSITO, Dino. XML: efektivní programování pro .NET. 1. vyd. Překlad Jaroslav Černý. Praha: Grada, 2004, 596 s. ISBN 80-247-0775-6.
- [20] ZELINKA I., OPLATKOVÁ Z., NOLLE L. Analytic Programming - Symbolic Regression by Means of Arbitrary Evolutionary Algorithms In: Special Issue on Intelligent Systems, International Journal of Simulation, Systems, Science and Technology, Volume 6, Issue 9, August 2005, ISSN 1473-8031.
- [21] Mapy.cz. [online]. [cit. 2013-04-02]. Dostupné z: <http://www.mapy.cz>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
CTS	The Common Type System
GC	Garbage Collector
IDE	Integrated Development Environment
VES	Virtual Execution System

**SEZNAM OBRÁZKŮ**

<i>Obr. 1. Vrcholy a hrany</i> .....	12
<i>Obr. 2. Vrcholy a hrany tvořící cestu</i> .....	14
<i>Obr. 3. Příklady úplných grafů</i> .....	15
<i>Obr. 4 - Náhled grafu vytvořeného v aplikaci yFiles</i> .....	19
<i>Obr. 5. Náhled aplikace Gephi [11]</i> .....	21
<i>Obr. 6. Náhled vývojového prostředí Visual Studio C# Express</i> .....	28
<i>Obr. 7. Základní návrh a rozložení</i> .....	31
<i>Obr. 8. Diagram tvorby textového vstupu</i> .....	35
<i>Obr. 9. Grafický výstup dle textového zadání</i> .....	35
<i>Obr. 10. Ukázka použití grafických tříd</i> .....	39
<i>Obr. 11. Ukázka kruhového uspořádání</i> .....	45
<i>Obr. 12. Graf s 28 vrcholy</i> .....	46
<i>Obr. 13. Stromový graf se zapnutou gravitací</i> .....	49
<i>Obr. 14. Náhled editoru, mapa ČR</i> .....	55
<i>Obr. 15. Výstup dle ukázkového XML</i> .....	60
<i>Obr. 16. Výstup ukázkového kódu v jazyce C#</i> .....	61
<i>Obr. 17. Výstup ukázkového kódu v jazyce Visual Basic</i> .....	63
<i>Obr. 18. Interaktivní topologie sítě</i> .....	64
<i>Obr. 19. Vizualizace výstupu AP</i> .....	67
<i>Obr. 20. Upravená vizualizace výstupu AP</i> .....	68
<i>Obr. 21. Náhled webové prezentace</i> .....	69

**SEZNAM TABULEK**

<i>Tab. 1. Cenový přehled yFiles .NET [10]</i> .....	20
<i>Tab. 2. Ukázka dědění vlastností grafických tříd</i> .....	40
<i>Tab. 3. Ukázka postupné organizace vrcholů</i> .....	42

## SEZNAM PŘÍLOH

PI CD-ROM