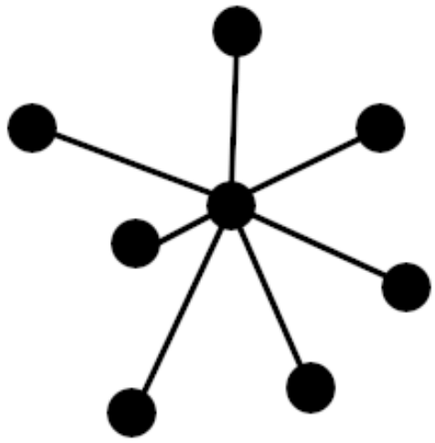
Dokumentace – Grafový editor



Ondřej Kříž
18. 9. 2021

Obsah

Anotace

Cílem softwaru je poskytnout uživateli jednoduchý editor podoby Windows aplikace, určený pro tvorbu orientovaných a neorientovaných grafů. Software podporuje několik rozličných editačních nástrojů, které ulehčují práci při manipulaci a vytváření grafu. Nabízí také základní paletu grafických nastavení určených pro úpravu vizuální podoby grafu. Dále aplikace podporuje export dat grafu, a to v čistě textové nebo vizuální podobě.

1 Specifikace

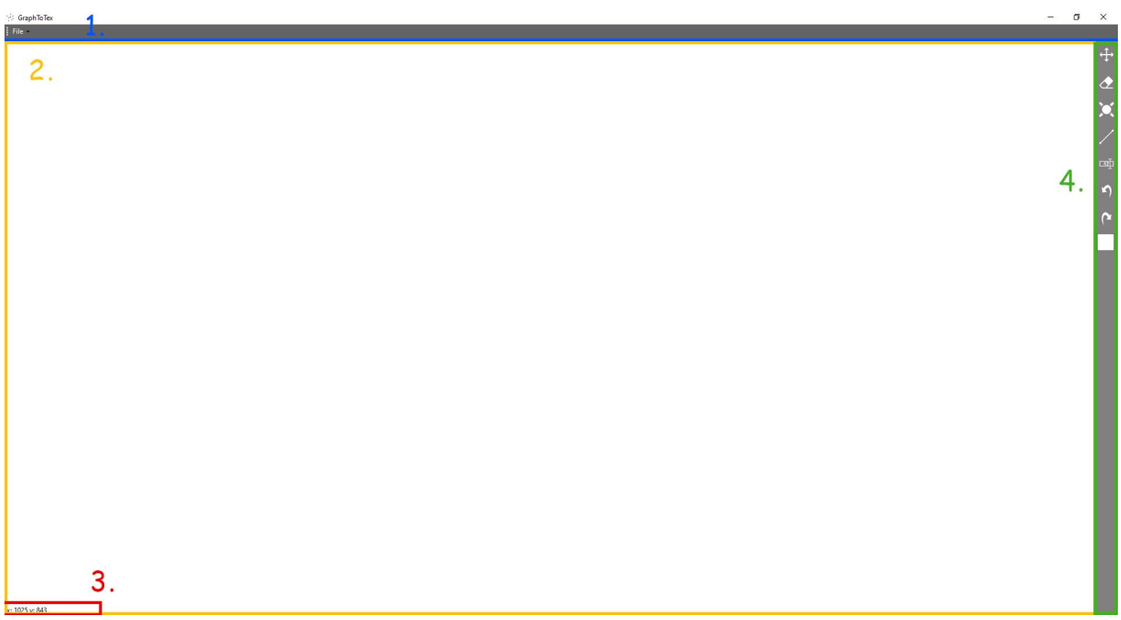
Aplikace je navržena v prostředí Windows Forms a je tedy primárně cílena na OS Windows. Uživateli poskytuje řadu jednoduchých nástrojů, jak jednotlivé objekty grafu(uzly a hrany) upravovat. Pro tvorbu grafu uživateli nabízí následující editační nástroje:

- přesun uzlu,
- smazání uzlu/hrany,
- vložení hodnoty,
- undo/redo operace,
- změnu barvy pozadí grafu.

Dále aplikace umožňuje export vytvořeného grafu v třech rozličných podobách, a sice jako:

- seznamu hran,
- seznam následovníků,
- snímek plátna.

Obrázek 1 níže zobrazuje výchozí rozložení uživatelského rozhraní aplikace.



Obrázek 1: Výchozí rozložení aplikace

1. Horní lišta (toolstrip).
2. Kreslící plátno aplikace.
3. Informační text pro uživatele.
4. Panel s editačními nástroji grafu.

1.1 Vytváření a editace grafu

Veškeré ovládací prvky, sloužící k práci s grafem jsou k nalezení v pravém panelu s editačními nástroji grafu. Každá operace má nicméně i svojí ekvivalentní klávesovou zkratku. Pro zrušení aktuální operace slouží klávesa – *Esc*.

Obrázek 2: Příslušná ikona v panelu nástrojů

1.1.1 Vkládání uzlů

Pro přepnutí aplikace do módu vkládání uzlů stačí kliknout na výše vyzobrazenou ikonu (obr. 2) v editačním panelu nástrojů nebo zmáčknout klávesu – *I*.
Uživateli se následně v pravé části okna otevře nový panel s vlastnostmi aktuálně vkládaného uzlu. Názorná ukázka je na obrázku níže.

Obrázek 3: Panel vlastností vkládaného uzlu

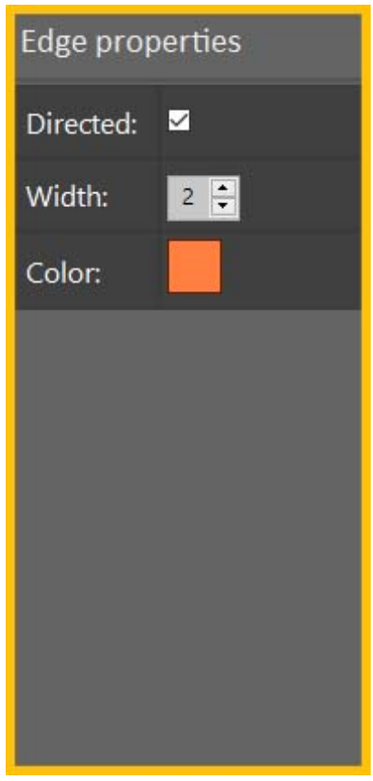
- Shape – nastavení tvaru uzlu(kruh, čtverec).
- Size – Velikost uzlu (10-40 px).
- Background color – Barva uzlu.
- Border color – Barva okraje uzlu.
- Border width – Tloušťka okraje (0-10 px).

Při vkládání se nový uzel vykresluje na aktuální pozici myši. Vložení uzlu do grafu na daných souřadnicích, se potvrzuje levým tlačítkem myši.

1.1.2 Vkládání hran

Obrázek 4: Příslušná ikona v panelu nástrojů

Po kliknutí na ikonu obr. 4 v editačním panelu nástrojů, nebo zmáčknutím klávesy – *E*, se aplikace přepne do módu vkládání hran. Uživateli se otevře následující panel s vlastnostmi aktuálně vytvářené hrany:



Obrázek 5: Panel vlastností vkládané hrany

- Directed – vložení orientované hrany(defaultně zapnuto).
- Width – tloušťka hrany(1-5 px).
- Color – barva hrany.

Pro vložení nové hrany uživatel nejprve klikne na libovolný uzel v grafu. Následně se objeví atributy definovaná hrana začínající ve zvoleném uzlu a končící u uživatelovi pozice myši na plátně. Pro vložení stačí kliknout na libovolný uzel v grafu, který ještě není s prvotním uzlem spojený. Aplikace nepodporuje vícenásobné hrany a identitu.

1.1.3 Vkládání hodnot

Obrázek 6: Příslušná ikona v panelu nástrojů

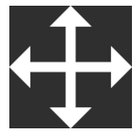
Kliknutím na výše uvedenou ikonu nebo stisknutím klávesové zkratky – *V*, se aplikace přepne do módu vkládání hodnot. Pro vložení hodnoty stačí kliknout na příslušnou hranu či uzel. Následně se otevře textové pole na pozici vkládané hodnoty. Pokud již prvek měl hodnotu definovanou pole bude obsahovat tuto hodnotu. Pro ukončení/potvrzení vložení hodnoty u daného prvku slouží klávesa – *Enter*.

1.1.4 Mazaní prvků grafu

Obrázek 7: Příslušná ikona v panelu nástrojů

Pro zapnutí módu slouží mimo tlačítka v editačním panelu nástrojů klávesová zkratka – *D*. Pokud uživatel v tomto módu klikne na uzel nebo hranu grafu, příslušný prvek se z grafu okamžitě odstraní.

1.1.5 Přesouvání již vložených uzlů



Obrázek 8: Příslušná ikona v panelu nástrojů

Tento mód lze zapnout stisknutím tlačítka – *M* nebo kliknutím na ikonu výše. Po kliknutí levým tlačítkem myši na libovolný uzel na plátně se uzel bude pohybovat společně s myší uživatele až do potvrzení nové pozice opětovným stisknutím levého tlačítka myši.

1.2 Export grafu

Pro export dat grafu slouží horní *tool strip*, nabídka *file->export*. Při exportu dat se uživateli otevře Windows ukládací průzkumník, ve kterém si zvolí umístění a název daného exportu.

1.2.1 Export dat

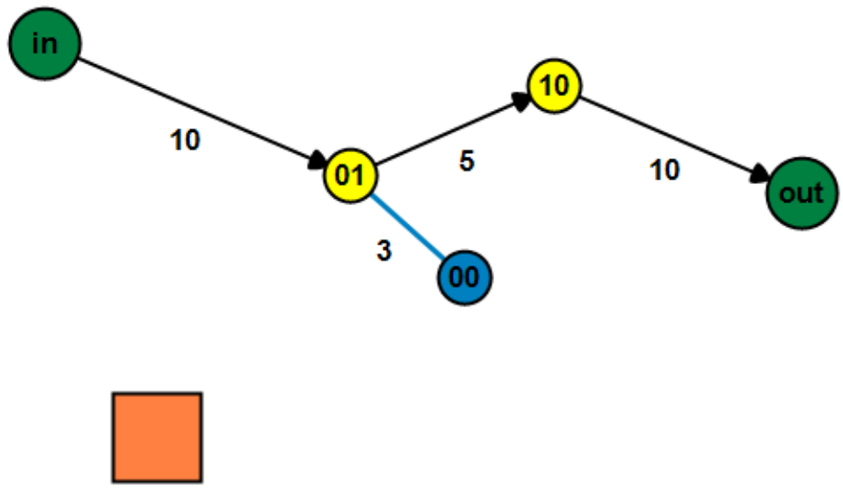
Jak již bylo zmíněno, data grafu lze exportovat jako

- seznamu hran nebo
- seznam následníků.

Každý vytvořený uzel má v grafu své jednoznačné identifikační označení v podobě:

node_XXX,

kde XXX značí trojciferné číslo doplněné o *leading zeroes*, začínající 000. Číslo poté reprezentuje kolikátý byl uzel do grafu přidán[1].
Při exportu je poté tento identifikátor obohacen o závorku – (value), reprezentující hodnotu uzlu.
Reprezentace hodnoty hrany je udána v hranaté závorce.
Pokud objekty nemají přiřazenou hodnotu, závorky nejsou do výstupu přidány. Výsledný exportovaný souboru je poté ve formátu *.txt*.
Pro graf na obr. 9, jsou následující příklady exportů.



Obrázek 9: Příklad grafu vytvořeného v aplikaci

Příklad exportu seznamu hran:
node011(in) - node013(01)[10]
node013(01) - node016(10)[5]
node013(01) - node017(00)[3]
node017(00) - node013(01)[3]
node016(10) - node010(out)[10]
node012

Příklad exportu seznam následníků:
node011(in) - node013(01)[10]
node010(out)
node017(00) - node013(01)[3]
node013(01) - node016(10)[5], node017(00)[3]
node012 node016(10) - node010(out)[10]

1.2.2 Export plátna

Další možnost exportu je jako *screenshot*(snímek) aktuálního plátna ve formátu *.jpg*. Velikost exportovaného obrázku poté odpovídá současné velikosti plátna. Příklad exportu reprezentuje obr. 9.

[1]Hodnota – XXX – je počítána interně v rámci operace přidání nového uzlu v GraphRepresentation-Model

2 Algoritmy

Většina algoritmů je především určena k výpočtu geometrie pro správné vykreslování dat a registraci interakce s grafem. V této kapitole je shrnut výběr nejpodstatnější algoritmů, které představují jádro těchto operací. Dále je nutné předem zmínit, že popisy algoritmů včetně pseudokódů se mohou lehce od implementace lišit, a to především v pojmenování proměnných[2]. Nicméně obecné principy výpočtů, průběhy algoritmů, použité datové struktury a matematické formule s nimiž aplikace pracuje jsou stejné.

2.1 Registrování interakcí s objekty

2.1.1 Uzel

Detekce probíhá na základě toho, zdali vstupní souřadnice (souřadnice myši při stisknutí levého tlačítka) leží v kruhu definovaném středem uzlu a jeho průměrem – velikostí uzlu. Implementace se nachází v metodě *GetNodeOnCoords* GraphRepresentationModel.
Algoritmus iteruje přes všechny uzly grafu. U každého uzlu určí rozdíl souřadnic pozice uzlu a pozice vstupních souřadnic – dx, dy. Následně podle rovnice kruhu v rovině určí jeho poloměr $r = \sqrt{(dx)^2 + (dy)^2}$
V algoritmu mimojiné probíhá kontrola, zdali má právě iterovaný uzel vlastnost *Enabled*[3] nastavenou na *true*.

```
1  Function GetNodeOnCoords(coords) is
2      for all nodes – n do
3          dx, dy ← 0
4          distance ← 0
5          dx ← Abs(n.X − coords.x)
6          dy ← Abs(n.Y − coords.y)
7          distance = Sqrt((dx)^2 + (dy)^2)
8          if distance < n.size & n.IsEnabled then
9              return n /*Nalezli jsme požadovaný uzel.
10         return null /*Jinak
```

Algorithm 1: Pseudokód algoritmu na detekci interakce s uzlem[a]

[a] V kódu se některé části mohou implementačně lišit.

2.1.2 Hrana

Pro určení jestli vstupní souřadnice leží na úsečce hrany bylo především využito základních poznatků o projekci do podprostoru přímky.
Algoritmus postupně prochází všechny hrany a v každém kroku ověřuje následující dvě podmínky, jenž musejí platit současně:

1. vstupní souřadnice leží na přímce z hlediska koeficientu tloušťky čáry
2. a zdali souřadnice leží mezi koncovými body definující hranu.

Ověření první podmínky je založeno na určení vzdálenosti mezi vstupními souřadnicemi a přímkou dané směrovým vektorem – s[4] hrany grafu. Algoritmus tedy nejprve určí tento vektor. Na jeho základě a poskytnutých vstupních souřadnicích poté pomocná funkce *GetProjectionOnLine* spočítá ortogonální projekci na přímku danou s. Následně norma(velikost) vektoru mezi projekcí a vstupními souřadnicemi je vzdálenost vstupních souřadnic od přímky definované s. Značeno *perpDistFromLine*.
Protože se však hrana vyskytuje v afinním podprostoru je nutné pro určení skutečné vzdálenosti mezi hranou a vstupními souřadnicemi odečíst vzniklý posun – *distRef*. Ten

[2] Dále jsou například některé triviální operace(především u přiřazování hodnot) zapsány jen ve formě funkce(která v implementaci nemusí být), s cílem udělat pseudokód čitelnější.
[3] Což je spíše implementační detail v rámci této aplikace.
[4] Pojmenování v kódu *directionVect*

lze spočítat stejným způsobem jako vzdálenost vstupních dat od přímky výše, s tím rozdílem, že nyní jako vstupní souřadnice algoritmus použije počáteční(nebo koncový) bod hrany. Výsledná vzdálenost tedy poté je:

$$realDist = |perpDistFromLine - distRef| \tag{1}$$

Podmínka je splněna, pokud je vzdálenost menší než *maxDistCoef*, definovaný jako:

$$maxDistCoef := (edgeWidth/2) + Settings.EdgeSelectionTolerancCoef. \tag{2}$$

Pro ověření druhé podmínky jsou spočítány nejprve obě projekce koncových bodů hrany do podprostoru přímky. Následně se určí vzdálenost projekce vstupních souřadnic od obou projekcí koncových bodů hrany, značené $d1, d2$. Pokud bod leží na úsečce(hraně) definované těmito krajními body, musí platit:

$$(d1 + d2) - |e| = 0, \tag{3}$$

kde |e| značí velikost hrany.
Samozřejmě tento přístup funguje pouze pokud uvažujeme hranu, která je rovná úsečka. Pokud bychom měli křivky libovolného tvaru, byl by výpočet složitější. Dále protože v aplikaci je pozice zaokrouhlována na celá čísla, nemusí být vždy výsledný rozdíl(za předpokladu, že vstupní pozice se skutečně nachází na hraně) roven 0. Proto je nutné zavést jistou toleranci $\epsilon = 1$.

Celý algoritmus je k nalezení v **Namespace**: SimpleGraphEditor.Models.GraphModel, jako samostatná třída – **CoordsOnEdge**. Níže uvádím velmi zjednodušenou(implementačně) verzi v pseudokódu.

```
1  Function CheckIfCoordsOnEdge(coords) is
2      maxDistCoef
3      ε ← 1 result ← true
4      for all edge – e do
5          s ← CalculateDirVector(e) /*Směrový vektor
6          coords_P ← CalculateProjection(coords, s) /* Projekce coords
7          start_nd_P ← CalculateProjection(e.End_nd, s) /*Projekce start node
            end_nd_P ← CalculateProjection(e.Start_nd, s) /* Projekce end node

8          /* První podmínka:
9          /* Určení afinního posunu
10         distRef ← GetVectorDist(start_nd.coords, start_nd_P)
11         perpDistFromLine ← GetVectorDist(coords, coords_P)
12         realDistance ← Abs(perpDistFromLine − distRef)
13
14         if realDistance >= distRef then
15             /* Pokud je vzdálenost souřadnic od hrany větší jak maximální
                povolená return false
16             return false
17
18
19         /* Druhá podmínka:
20         d1 ← GetVectorsDistance(start_nd_P, coords_P)
21         d2 ← GetVectorsDistance(end_nd_P, coords_P)
22         edgeLength ← |e|
23         realDist ← Abs((d1 + d2) − edgeLength)
24         if realDist > ε then
25             return false
26         return true; /* Jinak jsou obě podmínky splněny
27     return false;
```

Algorithm 2: Pseudokód algoritmu na detekci interakce s hranou

2.2 Výpočet pozice textu hodnoty objektu

Výpočet pozice textu hodnoty hrany je podobný předešlému algoritmu. Text je standardně vykreslován na prostředku hrany. Pozice je poté určena délkou kolmého vektoru, kde pata této normály o dané délce je právě ve středu hrany. Pro určení normály je nejprve vypočítán směrový vektor mezi začátkem hrany a jejím středem. Neboť se pohybujeme v rovině, lze kolmý vektor zkonstruovat pouze prohozením souřadnic a změnou znaménka u jedné ze složek (v implementaci se konkrétně jedná o 2. složku vektoru). Na konec je pouze nutné velikost vektoru přeškálovat. V aplikaci je délka předdefinována v obecném Settings. Implementace je obsažena v veřejném rozhraní třídy **EdgeData**.

```
1 Function CalculateEdgeLablePosition(e) is
2     /* e – vstupní hrana
3     center ← GetCenterPosition(e) /* Řešeno aritmetickým průměrem přes
        jednotlivé složky souřadnic.
4     s ← CalculateDirVector(e) /* Určení směrového vektoru hrany
5     /* Určení kolmého vektoru na směrový vektor
6     perp_vec ← GetPerpedicularVector(s)
7     result ← GetVectorOfGivenLength(perp_vec)
8     return result;
```

Algorithm 3: Pseudokód algoritmu pro výpočet pozice textu hodnoty hrany

2.3 Historie editačních změn

2.4 Kopie grafu

Historie undo a redo operací je v aplikaci navržena dle vzoru *memento pattern*, kde memento objekt obsahuje kopii dat reprezentující určitý stav v historii aplikace (tedy konkrétně data grafu). K vytvoření mementa bylo zapotřebí vytvořit *deep copy* dat grafu. Kopírování je řešeno prohledáváním grafu do hloubky rekurzivní metodou – *SetGraphCopy*. Tato metoda je volána na všechny doposud nenavštívené vrcholy z metody – *CreateCopy*. Tím je docíleno, že se zkopírují všechny komponenty grafu. Samotná implementace se nachází v třídě **CopyGraphData**, obsahující zmíněné metody. Pseudokód níže schématicky zobrazuje implementaci obou hlavních metod algoritmu:

```
1 originalData /*Původní data grafu (seznam uzlů a incidentních hran)
2 visitedNodes ← ∅ /* Slovník kde klíčem je původní odkaz na uzel a hodnota –
    reference na kopii
3 newGraphData ← ∅ /* Nová kopie dat grafu
4 Function CreateCopy()is
5     if #originalData > 0 then
6         for all node in originalData do
7             if visitedNodes.Member(node) = false then
8                 SetGraphCopy(node)
9
```

Algorithm 4: Pseudokód algoritmu kopírování dat grafu

```
1  Function SetGraphCopy(nd1)is
2      /* nd1 – počáteční uzel hrany
3      newNd1 ← Copy(nd1)
4      /* Vytvoří se nový objekt uzlu. Odkazy na samotná data uzlu(obsahující např.
         template) zůstanou nezměněna
5      visitedNodes.Add(nd1, newNd1)
6      newEdgeList ← ∅ /*Nová kopie seznamu hran.
7      newGraphData.Add(newNd1, newEdgeList)
8
9      for all e in originalData[nd1] do
10     Kde originalData[nd1] je seznam hran incidentních hran s nd1 v původním
       grafu  /*Koncový uzel nové hrany
11     newNd2 ← ∅
12     /* Pokud navštívené vrcholy již obsahují koncový uzel hrany
13     if visitedNodes.Member(e.nd2) then
14         newNd2 ← visitedNodes[e.nd2]
15     else
16         /*Jinak koncový uzel hrany ještě nebyl zkopírován a tedy rekurzíme
17         newNd2 ← SetGraphCopy(e.nd2)
18     /*Přidáme novou kopii hrany do dat nového grafu:
19     newEdge ← CreateNewEdge(newNd1, newNd2, e.Data)
20     newGraphData[newNd1].Add(newEdge)
```

Algorithm 5: Pseudokód algoritmu kopírování dat grafu

3 Program

Návrh celé aplikace využívá principu *Model View Presenter* (zkráceně MVP), konkrétně tzv. *passive view* přístup. Tento návrh zajišťuje kompletní rozdělení aplikace na samotnou reprezentaci dat, realizované zejména pomocí modelů, a frontend, který je založen na rozhraní Windows Forms. View jsou v aplikaci definovány pomocí interface, kterému odpovídá příslušný presenter.

Celkem se v aplikaci nacházejí 3 samostatné windows formuláře reprezentjící hlavní samostatná *view* aplikace. Jedná se o:

- EditorForm
- NodePropertiesForm
- EdgePropertiesForm

3.1 EditorForm

Editor form je nejrozsáhlejším view celé aplikace. Skládá se z několika ”subviews”, které jsou definovány jako *partial class* třídy *EditorForm*. Každá takováto *partial class*(reprezentující view) má definovanou svou veřejnou interface a tedy je s nimi v aplikaci zacházeno jako se samostatnými views. Soubory obsahující tyto *partial class* jsou:

- EditorForm.cs
- ToolsPanel.cs
- ToolStrip.cs
- InfoTextBox.cs

Zde pojmenování souborů slouží pouze pro orientaci mezi soubory, neboť samotné názvy *partial* tříd nesou(již z definice *partial class*) společný název – EditorForm. Některé tyto views tak spolu mohou sdílet jednotlivé presentery. (Konkretně se takto v aplikaci používá , který přísluší **IGraphView**)

Dále je poté část třídy EditorForm obsažena v souboru *ClientKeyInputs.cs*, implementující *listener*: **MainForm_KeyDown** , zpracovávající vstup klávesnice.

3.1.1 Rozhraní IGaphView

Namespace: SimpleGraphEditor.Views

Rozhraní obsahuje především metody a atributy, sloužící pro vykreslování objektů na hlavním plátně editoru. Rozhraní přísluší souboru pojmenovaném *EditorForm.cs*.
Pro atributy vykreslovaných objektů(vrcholy, hrany, text hodnot) je v kódu zavedena (neformální)konvence určující pojmenování, které začíná předponou *New*.

Obsahuje následující položky:

- **GraphPresenter** MainPresenter **set;** – Reference presentru starající se především o plátno aplikace.
- **void** AddNodeShape((**int** x, **int** y) coordinates) – vykreslení(za pomoci Windows forms knihoven) nového uzlu na plátno.
- **void** AddEdgeShape((**int** x, **int** y) startNodeCoordinates, (**int** x, **int** y) endNodeCoordinates); – vykreslení nově přidané hrany definované souřadnicemi počátečního a koncového bodu.
- **void** AddElementValueText((**int** x, **int** y) textPosition, **string** nodeData);

Metody pro správu plátna aplikace:

- **void** UpdateCanvas(); – Překreslení celého okna.
- **void** ClearCanvas(); - Vyčištění plátna.
- **Color** CanvasBackColor **get;** – Atribut obsahující momentální barvu plátna.

Aktualizují nástroje pomocí nichž se vykreslují nové objekty grafiky:

- **void** UpdateNodeBrush();
- **void** UpdateNodePen();
- **void** UpdateEdgePen();

Následující dvě deklerace ovládají zobrazení *text-boxu*, kterým uživatel v aplikaci vkládá data. Parametr *Coords* jsou souřadnice, na kterých se *text-box* má vykreselit (Tedy například pro vrchol grafu je to přímo na jeho souřadnicích, apod.):

- **void** ShowValueInsertionBox((**int** x, **int** y) coords, **string** defaultValue = "");
- **void** HideValueInsertionBox();
- **void** OpenNodeProperties();
- **void** OpenEdgeProperties();
- **void** ClosePropertiesPanel();

Atributy slouží k nastavení momentálně vykreslovaného uzlu:

- **int** NewNodeBorderWidth **get; set;**
- **Settings.NodeShape** NewNodeShape **get; set;**
- **Color** NewNodeBorderColor **get; set;**

- **bool** NewNodeDrawBorder **get; set;**
- **Color** NewNodeColor **get; set;**
- **int** NewNodeSize **get; set;**

Nastavení parametrů aktuálně vykreslované hrany:

- **bool** NewEdgeIsDirected **get; set;**
- **Color** NewEdgeColor **get; set;**
- **int** NewEdgeWidth **get; set;**

Nastavení momentálně vykreslovaného textu hodnoty uzlu:

- **public int** NewLabelFontSize **get; set;**
- **public** Color NewLabelFontColor **get; set;**
- **string** NewLableTextValue **get; set;**
- **event EventHandler**<**EventArgs**> ClientConfirmedOperation; – Událost vyvolaná při stisku potvrzující klávesy. (V aplikaci nastaveno na Enter.)
- (**int** X, **int** Y) MouseCoords **get; set;**

3.1.2 Rozhranní IToolsPanelView

Přísluší k souboru *ToolsPanel.cs*. Obsahuje pouze metodu pro inicializaci.(Například se zde nachází nastavení *tooltips*(popisků), k jednotlivým položkám(tlačítkům) panelu nástrojů.)

- **void** InitializeToolsPanel();

3.1.3 Rozhranní IInfoTextBoxView

Přísluší k souboru *InfoTextBox.cs*. Slouží pro vykreslení textových informací o stavu aplikace. Obsahuje:

- **InfoTextBoxPresenter** TextBoxPresenter **get; set;** – Příslušný prezenter.
- **string** DataText **get; set;** – Momentální text vykreslující se na obrazovce.
- **bool** IsMouseCoordsVisible **get; set;**
- **void** InitializeInfoTextBox();

3.1.4 Rozhranní IToolStripView

- **ToolStripPresenter** StripPresenter **get; set;**
- **void** InitializeToolStrip();

3.2 NodePropertiesForm

Panel pro úpravu vzhledu uzlů. Jedná se o samostatný formulář Window forms obsahující svůj vlastní designer. Jeho veřejná interface je poté **INodePropertiesView**.

Obsahuje atributy pro nastavení momentálně vkládaného uzlu(vždy s předponou *New*):

- **Color** NewBackColor **get; set;**
- **int** NewNodeSize **get; set;**
- **Color** NewBorderColor **get; set;**
- **int** NewBorderWidth **get; set;**
- **Settings.NodeShape** NewNodeShape **get; set;**
- **NodePropertiesPresenter** PropPresenter **get; set;**
- **void** UpdatePropertiesControls(); – Update ovládacích prvků na základě nastavených atribut.

3.3 EdgePropertiesForm

Data panelu pro nastavení vlastností vykreslované hrany:

- **EdgePropertiesPresenter** propPresenter **get; set;**
- **Color** NewEdgeColor **get; set;**
- **bool** NewEdgeIsDirected **get; set;**
- **int** NewEdgeWidth **get; set;**

4 Logika aplikace a presenters

4.1 Zpracování editačních operací

Hlavní logika pro zpracování jednotlivých módů operací je řešena v presenteru **GraphPresenter** příslušící **IGaphView**. Ten obsahuje dependency:

public GraphEditorMachine EditorMachine.

4.1.1 GraphEditorMachine

Namespace: SimpleGraphEditor.Presenters

Reprezentuje konečný stavový automat, jehož veřejné rozhraní obsahuje atribut:

public EditorState CurrentState **get; set;** ,

jenž reprezentuje aktuální stav automatu.
EditorState je *abstraktní* třída, definující rozhraní každého stavu automatu. Obsahuje základní implementaci přechodů do každého stavu. Každá *concrete implementace* stavu poté dědí z této *abstraktní* třídy. Přechody jsou:

- **public virtual void** TurnOnDeletationMode() – zapnutí delete módu
- **public virtual void** TurnOnDragMode() – zapnutí přesunu uzlů po plátně
- **public virtual void** TurnOnEdgeInsertionMode() – vkládání hran
- **public virtual void** TurnOnIdleMode() – přepnutí do defaultního módu (např. při spuštění aplikace)
- **public virtual void** TurnOnNodeInsertionMode() – vkládání uzlů
- **public virtual void** TurnOnValueEditState() – vkládání hodnoty

Základní implementace obsahuje vždy přechod do každého stavu a nastavení správného rendereru CanvasRenderMachine.
Implementace tříd reprezentující příslušné stavy jsou poté následující:

- **IdleModeState** IdleModeState **get; set;**
- **DeletionModeState** DeletionModeState **get; set;**
- **NodeInsertionModeState** NodeInsertionModeState **get; set;**
- **EdgeInsertionModeState** EdgeInsertionModeState **get; set;**
- **ValueEditModeState** ValueEditModeState **get; set;**
- **DragNodeState** DragNodeModeState **get; set;**

Mimo jiné abstraktní třída **EditorState** také deklaruje abstraktní metodu:
public abstract void OnClientInteract((**int** x, **int** y) coords),
jenž se volá pokud klient provede interakci s grafem.

4.2 Vykreslování a data binding grafu

4.2.1 CanvasRenderMachine

Namespace: SimpleGraphEditor.Presenters.CanvasRendererMachine

Jedná se o další, nicméně z hlediska implementace velmi zjednodušený, konečný stavový automat. Jeho hlavním úkolem je zaručit uspořádání v jakém se jednotlivé objekty na plátně budou vykreslovat. (Například aby hrany byly vždy vykresleny dříve než vrcholy grafu, ...). Jednotlivé stavy jsou zde reprezentovány jako "balíček"vykreslovacích funkcí, které jsou implementovany v **GraphPresenter** a které mají být v určeném pořadí po sobě zavolány. Vykreslovací funkce jsou:

- **void** ClearCanvas() – vyčištění plátna,
- **void** UpdateEdges() – vykreslení hran v závislosti na modelu grafu GraphRepresentationModel,
- **void** UpdataNodes() – vykreslení hran v závislosti na modelu grafu,
- **void** UpdateMouseDummyNode() – vykreslení "dummy"uzlu na pozici myši při vkládání nového uzlu,
- **void** UpdateMouseDummyEdges() – vykreslení "dummy"edge při vkládání hrany nebo hran připojených přemisťovaného uzlu, při operaci přemístění,

K zajištění pořadí je zde použita *Queue*, která obsahuje standardní typ delegáta **public delegate void** Action(), který tak předepisuje formát vykreslovací funkce. Jednotlivé stavy mají poté přidružený **enum RenderState**.
Položky tohoto **enum** definují jednotlivé stavy, které jsou (za pomlčkou pořadí vykreslovacích funkcí daného stavu):

- **REND0** – ClearCanvas
- **REND1** – ClearCanvas, UpdataNodes, UpdateEdges
- **REND2** – ClearCanvas, UpdataNodes, UpdateEdges, UpdateMouseDummyNode
- **REND3** – ClearCanvas, UpdateMouseDummyEdges, UpdataNodes, UpdateEdges

4.3 Průběh updatu view grafu

Aktualizace probíhá jak již bylo výše zmíněno pomocí vykreslovacích funkcí definovaných v **GraphPresenter**.Update zde vypadá tak, že metody projdou veškeré existující data (uzly, hrany) v modelu grafu a každý objekt překreslí pomocí metod *AddNodeShape*, *voidAddEdgeShape*, *voidAddElementValueText* obsažených v rozhraní IGraphView.
Pro vykreslení tyto metody používají data z atribut (s předponou *New*) IGraphView, přičemž atributy jsou aktualizované pomocí privátních metod **GraphPresenter**:

- **private void** BindNewNodeShapeTemplate(INodeTemplate template),
- **private void** BindNewEdgeShapeTemplate(**IEdgeTemplate** template),
- **private void** BindNewLabelTextTemplate(**IValueLabelTemplate** template).

Tyto metody updatují atributy view v závislosti na rozhraní templatů, které definují vzhled jednotlivých objektů(uzlů, hran, textu hodnot).

5 Modely a reprezentace dat

Jednotilvé modely reprezentující data jsou se nacházejí v adresáři *./models*

5.1 GraphRepresentationModel

Namespace: SimpleGraphEditor.Models
Představuje hlavní reprezentaci tvořeného grafu. Veřejná interface **IGraphRepresentation<NodeData, EdgeData>** obsahuje především metody určené pro práci s jeho daty:

- **IReadOnlyDictionary<INode<T>, List<IEdge<S, T > > >** GraphData **get;** – seznam vrcholů a jejich hran (read only).
- **void** AddNodeToGraph(**INode<T>** newNode) – Přidání nového vrcholu.
- **void** RemoveNodeFromGraph(**INode<T>** newNode) – Odebere vrchol i všechny jeho incidentní hrany (neorientovaně).

Práce s hranami grafu:

- **void** AddEdgeToGraph(**IEdge<S, T>** newEdge, INode<**NodeData**> node)
- **void** RemoveEdgeFromGraph(IEdge<**EdgeData**, **NodeData**> edgeToDelete)

Metody určené pro vyhledání v grafu:

- **IEdge<EdgeData, NodeData>** GetEdgeOnCoords((**int** x, **int** y) coordinates)
- **INode<NodeData>** GetNodeOnCoords((**int** x, **int** y) coord)
- **HashSet<(INode<T>, IEdge<S, T>)>** GetConnectionsUndirected(**INode<T>** baseNode) – vrací
- **bool** HasThisNeighbour(**INode<NodeData>** baseNode, **INode<NodeData>** searchedNeighbour)
- **bool** AreNodesConectedByEdge(**INode<NodeData>** node1, **INode<NodeData>** node2) – Pokusí se získat hranu,

5.2 Reprezentace uzlu

5.2.1 Rozhraní uzlu

Namespace: SimpleGraphEditor.Models.Interface

Reprezentaci uzlu zajišťuje rozhraní **INode<T>**. Tato interface deklaruje:
Pozice uzlu na plátně:

- **int** Y **get; set;**
- **int** X **get; set;**
- **T** Data **get; set;** – Generický typ reprezentující data uzlu. V implemetaci v aplikaci k tomuto účelu slouží třída **NodeData**.

5.2.2 Třída NodeData

Namespace: SimpleGraphEditor.Models
Jedná se o konkrétní implementaci, reprezentující data uzlu. Veřejné rozhraní nabízí:

- **public string** Name **get; set;** – udržuje identifikátor zmíněný v sekci Export dat.
- **public bool** IsEnabled **get; set;** – Zdali je možné s uzlem interagovat.
- **public bool** CanBeRendered **get; set;** – Zdali se má uzel vykreslovat na plátno.
- **public INodeTemplate** Template **get; set;** – Data udržující vzhled uzlu. Více v sekci Rozhranní **InodeTemplate**.
- **public IValueLabelTemplate** LableTemplate **get; set;** – Informace o vzhledu hodnoty textu, Rozhranní **IValueLabelTemplate**.
- **public string** Value **get; set;** – Samotná hodnota uzlu.

5.3 Konkrétní implementace uzlu

Namespace: SimpleGraphEditor.Models

Implementuje rozhraní **INode<T>**. Dále objekt obsahuje přetížení následujících operátorů:

- **==**, **!=** – Porovnává se na základě pozice, která je jednoznačná. (Nikdy se nestane, že by dva uzly v grafu měli na plátně stejnou pozici.)
- **Equals**

5.4 Templates

Jednotlivé vykreslované objekty mají přidruženou svojí vlastní template, která určuje vzhled, jenž daný objekt(Uzel, hrana, textová hodnota) bude na plátně mít. Všechny následující rozhraní šablon se nacházejí v **Namespace** rozhraní modelů, tedy:
SimpleGraphEditor.Models.Interface

5.4.1 Rozhranní InodeTemplate

Reprezentuje vzhled uzlu grafu. Obsahuje atributy:

- **public Color** BackColor **get; set;**
- **public Color** BorderColor **get; set;**
- **public Settings.NodeShape** Shape **get; set;**

- **public int** BorderWidth **get; set;**
- **public bool** DrawBorder **get; set;**
- **public int** Size **get; set;**

Konkrétní implementace: **NodeTemplate**

5.4.2 Rozhranní IEdgeTemplate

Definuje grafické atributy hrany. Obsahuje:

- **Color** Color **get; set;**
- **bool** IsDirected **get; set;**
- **int** Width **get; set;**

Konkrétní implementace: **EdgeTemplate**

5.4.3 Rozhranní IValueLabelTemplate

Představuje grafickou reprezentaci textu hodnoty:

- (**int** x, **int** y) Coords **get; set;**
- **Color** fontColor **get; set;**
- **int** fontSize **get; set;**

Konkrétní implementace: **LableTemplate**

5.5 Konkrétní implementace templates

Veškeré výše zmíněné konkrétní implementace:

- NodeTemplate,
- EdgeTemplate,
- LableTemplate,

implementují pouze rozhraní šablon a rozhraní **ICloneable**, ze standardních knihoven *System*, které slouží pro vytvoření *shallow* kopie objektu.

5.6 Rozhranní IEditorModel

Reprezentuje momentální stav editoru. Obsahuje:

- **INode<NodeData>** SelectedNode **get; set;** – Právě vybraný uzel

Následující atributy reprezentují aktuálně nastavené šablony, na základě nichž se vykreslují nové objekty na plátně.

- **INodeTemplate** CurrentNewNodeTemplate **set;**
- **IEdgeTemplate** CurrentNewEdgeTemplate **set;**
- **IValueLabelTemplate** CurrentNewLableTemplate **set;**
- (**int** X, **int** Y) CanvasMousePosition **get;** – Aktuální pozice myši.
- **event EventHandler<EventArgs>** MouseMove; – Oznamuje o změně pozice myši.

- **void** SetActiveObjects(**HashSet<(INode<NodeData>, IEdge<EdgeData, NodeData>)>** data);

Metody vracející *shallow copy* šablon objektů:

- **INodeTemplate** GetCopyOfCurrentEdgeTemplate();
- **IEdgeTemplate** GetCopyOfCurrentNodeTemplate();
- **IValueLabelTemplate** GetCopyOfCurrentLableTemplate();

5.7 Rozhranní IExportGraphData

Namespace: SimpleGraphEditor.Models.Interface
Rozhranní definuje objekty sloužící k exportu dat grafu. Deklaruje pouze metodu:

- **public void** ExportData()

Následující jsou konkrétní implementace rozhraní.

5.7.1 ExportAdjacencyList

Namespace: SimpleGraphEditor.Models.Export
Slouží pro export dat ve formě seznamu následníků. Veřejné rozhraní:

- **public char** DefaultDelimiter **get; set;** = '-' – Oddělovač hlavního uzlu a seznamu sousedů.
- **public char** DefaultNeighbourDelimiter **get; set;** = ',' – Oddělovač mezi jednotlivými sousedními uzly.

5.7.2 ExportEdgeList

Slouží pro export dat jako seznam hran.

- **public char** DefaultDelimiter **get; set;** = '-' – Oddělovač názvů koncových uzlů hrany na řádce.

6 Další rozhraní a třídy

6.1 Třída settings

Namespace: SimpleGraphEditor.GeneralSettings
Jedná se o statickou třídu určená pro definici konstant a **readonly** statických polí určujících obecné a počáteční nastavení aplikace. Jednotlivá pole s jejich nastavením jsou uvedeny níže.
Nastavení Uzlu:

- **public enum** NodeShape Circle, Square ;
- **public static readonly** Color DefaultNodeColor = **Color**.Red;
- **public static readonly** Color DefaultNodeBorderColor = **Color**.Black;
- **public static readonly int** DefaultNodeRadius = 40;
- **public static readonly int** DefaultNodeBorderWidth = 2;

Iniciální nastavení týkající se hran:

- **public static readonly Color** DefaultEdgeColor = **Color**.Black;
- **public static readonly int** EdgeSelectionTolerancCoef = 2; – Koeficient určující
- **public static readonly int** DefaultEdgeWidth = 2;
- **public static readonly bool** IsEdgeDirectedDefault = **true**;
- **public static readonly int** EdgeLineTipSize = 5; //TODO!

Nastavení týkající se textu hodnoty:

- **public static readonly string** DefaultLableFont = ”Arial”;
- **public static readonly** Color DefaultLableColor = Color.Black;
- **public static readonly int** DefaultLableFontSize = 13;
- **public static readonly int** DefaultLableDistFromEdge = 20;

Iniciální paleta barev editoru:

public static readonly Color EditorColorDarkTransparent1 = Color.FromArgb(50, 45, 45, 45);

6.2 MathHelpers

Namespace: SimpleGraphEditor.Utils
Statická utility třída obsahující pomocné statické matematické funkce[5]. Rozhraní:

- (**int** x, **int** y) GetProjectionOnLine((**int** x, **int** y) projectedVect, (**int** x, **int** y) dirVector)
- **int** GetVectorsDistance((**int** x, **int** y) vec1, (**int** x, **int** y) vec2) – Vrátí vzdálenost dvou vektorů.
- (**int** x, **int** y) GetVectorOfGivenLength((**int** x, **int** y) vec, **int** len) – Vrátí skalární násobek vektoru *vec* o délce *len*
- **int** GetVectorNorm((**int** x, **int** y) vec) – Vrátí velikost vektoru.
- **int** GetDotProduct((**int** x, **int** y) vec1, (**int** x, **int** y) vec2) – Vrátí skalární součin *vec1* a *vec2*

[5] Především výpočty projekcí a práci s vektory, které nejsou ve standardní knihovně Math.

7 Závěr

Co se čistě obsahu aplikace týče, tak si myslím, že aplikace uživateli poskytuje jednoduše ovladatelný software, pro tvorbu grafů s základní, nicméně plně dostačující volbou grafických úprav. Co se objektového návrhu celé aplikace týče, tak si myslím že v mnoha směrech je celkem zdařilí – zejména z hlediska rozšiřovatelnosti a přidávání nových funkcionalit. Nejvíce jsem nejspíše spokojený s návrhem stavového automatu, který definuje jednotlivé módy aplikace.
Nicméně, v aplikaci je v momentální chvíli stále několik míst, které bych rozhodně udělal jinak. Jeden z největších problémů vidím ve splynutí *view* plátna grafu a editoru, kterým přísluší stejný *presenter* – ***GraphPresenter***. Ideální by bylo, kdyby plátno grafu mělo samostatný Windows formulář, který by byl kódem přiřazen do panelu hlavního editor formuláře, stejně tak, jak je to v momentální chvíli nastaveno pro okna *properties* jednotlivých objektů. Zmenšilo by se tak i rozhraní *IGraphView* a byl by zde lépe splněn *Single responsibility princip*. Také nejsem moc pyšný na současnou reprezentaci grafu. Myslím, že abstrakce by šla provést ještě o něco lépe, především aby šly jednotlivé reprezentace měnit v závislosti na současně tvořeném grafu pro případnou optimalizaci využitého času a místa. Konkrétně jsem v této implementaci zvolil slovník uzlů a jejich *incidentních* hran, nyní si myslím, že lepší řešení by kupříkladu bylo, použít jako klíče ve slovníku pouze pozice jednotlivých uzlů.
Dále si myslím, že je v aplikaci velký prostor pro zlepšení efektivity výpočtu některých operací. Zejména například neefektivní překreslování celého plátna, které by šlo například zlepšit rozdělením kreslící plochy na menší vykreslovací *chunky*, nebo pomocí jiných technik upravit právě překreslovanou oblast. V aplikaci je takových to příležitostí pro zlepšení několik.
Aplikace by tedy dle mého názoru potřebovala ještě v mnoha místech projít *refaktoringem*. Nicméně si i tak myslím, že současný stav aplikace je především celkem dobrým startovním bodem, pro další rozvoj, přidávání nových funkcionalit a ovládacích prvků.