GRAFICKÝ EDITOR **Hruška Marcel**

ÚVOD

Cieľom projektu bolo vytvoriť grafický editor s užívateľským rozhraním. Mal by vedieť vytvárať základné geometrické tvary ako elipsy, obdĺžniky, rovné čiary a polygóny. Tie by sa mali následne dať upravovať a to ich šírka, farba, dajú sa rotovať, meniť veľkosť alebo posúvať. Editor má na ľavej strane UI panel, na ktorom si upravujeme jednotlivé vlastnosti objektu, ktorý chceme vykresliť. Na pravo od neho máme biele pozadie, canvas, na ktorom prebieha samotné vykresľovanie. V ľavom hornom rohu máme ešte možnosť si canvas exportovať, uložiť, načítať, premazať alebo zmeniť jeho veľkosť.

SPRACOVANIE

Editor som vytváral v C++ pomocou Qt. Používam teda knižnice a featury, ktoré ma Qt k dispozícii ako sú QWidget, QTransform, QMouseEvent etc .

Program pozostáva z 2 hlavných častí. Celý formulár ako QMainWindow – editor a z QWidgetu - drawPlace. Na drawPlace sa dejú všetky vykreslenia, všetky widgets Qt ako Buttons alebo drawPlace sú obalené QWidgetom centralWidget. Editor ako taký ako class len riadi zmeny medzi UI a drawPlace, teda pomocou CONNECT posiela signály pri zmene hodnôt na paneli do slotov draw. drawPlace dedí z classy draw.

Ďalšou časťou je object.h, kde máme class jednotlivých objektov. Teda abstraktnú class Object, z ktorej potom dedia ďalej class Circle, Polygon, Rectangle a Line.

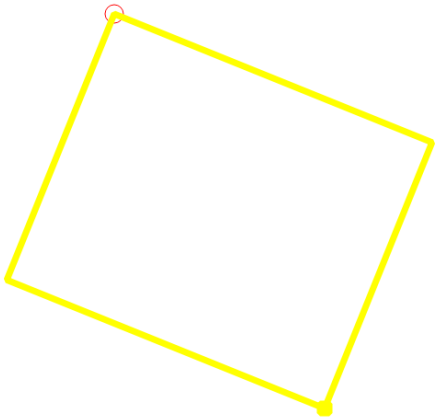
POUŽÍVANIE

Editor funguje obdobne ako Microsoft Paint. Na začiatku nemáme žiaden objekt vybraný, farba je červená, hrúbka čiary je 5 pixelov a objekt nebude vyplnený. Používateľ si môže vybrať jeden zo 4 objektov – elipsu, obdĺžnik, čiaru alebo polygón. Tie vidíme vľavo hore ako QButtons (tlačidlá). Pred kreslením si ešte môže určite farbu objektu pomocou tlačidla „Color“, na to som použil QColorDialog, ktorý je v Qt už predrobený, hrúbku čiary pomocou slideru s nadpisom „Size“ alebo či má byť objekt vyplnený zaškrtnutím QCheckBoxu Fill. Hrúbka sa dá nastaviť od 1 po 10 pixelov.

Pokiaľ máme vybrané, začneme kresliť kliknutím ľavého tlačidla myši na plochu, podržaním a potiahnutím od počiatočného bodu. Objekt sa nám vykresľuje medzi týmito dvomi bodmi. Keď pustíme tlačidlo, objekt nám ostane tak vykreslený.

Polygóny fungujú tak, že po pustení tlačidla a následnom kliknutí a podržaní na kresliacej ploche sa nám pridá nová čiara toho polygónu, ktorý vytvárame. Jeho kreslenie ukončíme tak, že potiahneme poslednú čiaru na začiatok už vykreslenej časti polygónu. Pokiaľ bolo vybrané kreslenie iného objektu pred dokončením polygónu, čiary, ktoré ho doteraz tvorili, tam ostanú.

Po vykreslení si môže používateľ navoliť možnosť „Pick“. Tá nám vyberá objekty na nakliknutie. Po kliknutí na hranicu objektu zožltne a jemne zhrubne. To značí, že objekt bol vybraný. Pokiaľ klikneme mimo akéhokoľvek objektu, aktuálny vybraný sa nám zruší. Pokiaľ vybereme nové kreslenie, takisto sa nám výber zruší. Pokiaľ je objekt vybraný, môžeme ho jeho podržaním za niektorú z čiar (aj keď je vyplnený, reaguje len na čiary) posúvať. Kurzor myši sa nám pri prejdení zmení na otvorenú dlaň, pri držaní na zatvorenú. Keď myš pustíme, uloží sa jeho nová poloha a tam ostane. Zároveň sa dajú vybrané objekty rotovať posúvaním slideru s názvom „Rotate“ od 0 po 360 stupňov. Stred rotácie objektu je v prvej hranici, teda v bode, kde sme ho začali vykreslovať. Ten sa nám zvýrazní, krúžkom vo farbe objektu. Meniť veľkosť môžeme za potiahnutie za akýkoľvek (okrem prvého) z rohov objektu. Tie sú všetky vyznačené žltými štvorčekmi a pokiaľ používateľ ponad ne prejde myšou, kurzor sa zmení na 4 šípky. Všetky tieto zmeny sa po zrušení výberu uložia.

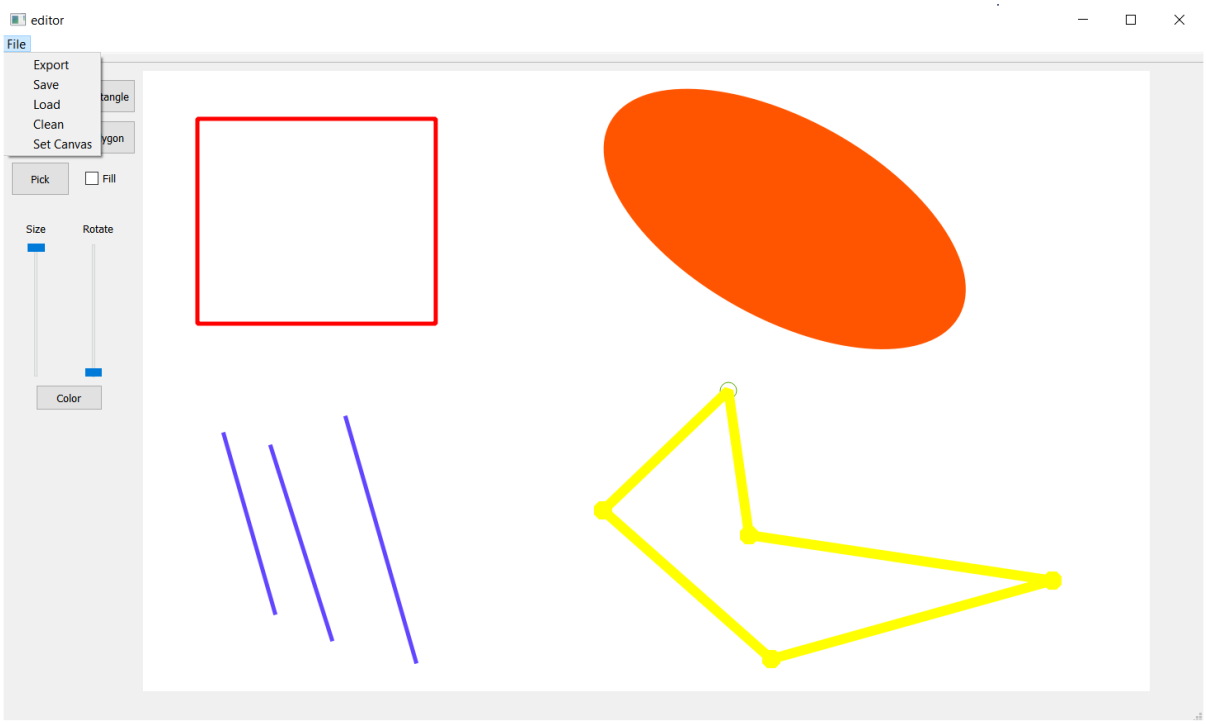


Vľavo hore vidíme prvý bod vykreslenia a teda stred okolo ktorého sa objekt otáča

Vpravo dole hranicu(zvýraznený roh), za ktorú keď potiahneme, meníme veľkosť

Po nakliknutí na horný ľavý roh, kde vidíme „File“ sa nám otvorí menu s výberom „Export“. Následne sa nám otvorí okno na uloženie celého obrázku do bežných obrázkových formátov, „Save“ na uloženie do .edf pre ďalšie používanie, „Load“ na načítanie takéhoto súboru, „Clean“, ktorý premaže všetky vykreslené objekty a „Set Canvas“, ktorý otvorí okno na určenie novej výšky a šírky kanvasu.

Nakoniec je možnosť mazania objektu kliknutím pravého tlačidla myši na akúkoľvek hranicu objektu.



Vykreslené všetky objekty, elipsa je vyplnená a pootočená, polygón vybraný. Takisto môžeme vidieť menu.

ČASTI PROGRAMU

Object.h

Tu máme classy, do ktorých ukladáme objekty.

Rodičovská class je **Object.** Nemá konstruktor, tie majú jednotlivé derivované vlastné. V protected má všetky vlastnosti objektu teda:

- *Int width_*

- *vektor<QPoint> borders_* (sú v ňom body, ktoré definujú ten objekt)
- *QColor color_*
- *Tool tool_* - určuje, čo je to za objekt, tool_ je enum Tool, kde môžeme mať enTr (polygon), enRec (obdĺžnik), enCir (elipsa), enLi (čiara), enNone (nič) alebo enPick (výber)
- *Bool selected_* - či je objekt vybraný
- *QReal rotation_*
- *Bool filled_*

Všetky tieto hodnoty vieme dostať z ich Get funkcií, ktoré sú spravené už v Object a teda spoločné pre všetky derivované.
Ďalej máme Set procedúry, ktoré upravujú hodnoty vybraného objektu:

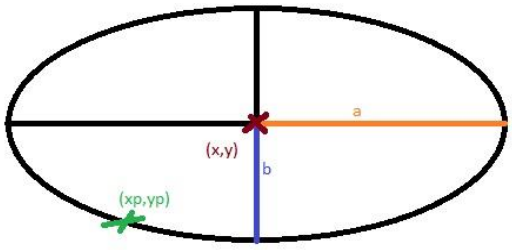
- *void Select(bool select)*
- *void SetRotation(qreal r)*
- *void SetColor(QColor c)*
- *void SetWidth(int w)*
- *void SetFilled(bool f)*
- *void SetBorder(QPoint p, int i)* – priamo v Object, prepíše hodnotu hranice i na bod p a prerobí objekt

Následne máme 2 virtuálne metódy, *bool pointIsOver(QPoint)*, ktorá vráti true, pokiaľ je bod nad objektom a *void SetPosition(int, int)*, ktorá posúva (offsetuje podľa argumentov) body v borders_ a následne prepisuje objekt samotný, ktorý sa má vykresliť.
Potom máme funkciu *int pointIsOnBorder(QPoint p) const*, ktorá nám vráti index hranice/rohu, na ktorý sme klikli z *borders_*.
Ďalej je hotová funkcia *bool isOnLine(const QPoint &first, const QPoint &second, const QPoint &toKnow) const*, ktorá zisťuje, či je bod toKnow na úsečke medzi first a second. Funguje tak, že si zrátame float (kvôli väčšej presnosti) vzialenosť medzi first a second, first a toKnow a second a toKnow. Pokiaľ je first_second = first_toKnow + toKnow_second, aj s nejakou odchylkou, je bod na úsečke. Na zrátanie vzdialeností používam vzorec $\sqrt{(x1-x2)^2 - (y1-y2)^2}$.
Nakoniec sú to procedúry *void store(QDataStream &out)* a *void load(QDataStream &in)*. Store dostane odkaz na stream, vloží doňho informácie o objekte v poradí – 1. *tool_*, aby sme neskôr vedeli, ktorý objekt chceme, 2. veľkosť *borders_*, kvôli nadchádzajúcemu for cyklu. Následne jednotliví body a ostatné vlastnosti objektu. Pri load dostaneme in, z ktorého zase nalievame info do nášeho nového objektu.

Derivované classy
Vo všeobecnosti sú veľmi podobné. Každá z nich má u seba uložený objekt na vykreslenie, class Circle – QRect (elipsa je u mňa vpísaná do obdĺžnika), class Polygon – QPolygon, class Rectangle – QRect a class Line – Qline. Každá má 2 konstruktory, jeden bezparametrický, len na vytvorenie objektu, to používam pri načítavaní zo súboru. Druhý prijíma argumenty do konstruktora *Body* (typedef na vector <QPoint>) *borders, QColor color, int width a bool fill*. Vo vnútri si ich poukladá, nastaví *tool_* podľa toho, o aký objekt sa jedná a zavolá *SetPosition(0,0)* – nechceme offset zatiaľ, teda sa nám len uloží objekt na vykreslenie.

Bool pointIsOver

- Line: triviálny prípad, zavoláme rodičovský *isOnLine* na oba body *borders_*
- Rectangle: voláme postupne na všetky 4 hrany obdĺžnika *isOnLine,* ak aspoň jeden prejde, vráti true
- Polygon: cez cyklus voláme na všetky dvojice bodov *borders_ isOnLine,* poslednú čiaru ako „posledný s prvým“
- Circle: vôbec nepoužívam *isOnLine* ale vzorec na elipsu $\frac{(xp-x)^2}{a^2} - \frac{(yp-y)^2}{b^2} = 1$. xp a yp sú súradnice toKnow bodu, x a y sú stred elipsy, a je vzdialenosť od stredu po prvý kraj a b je vzdialenosť od stredu po horný kraj. Prirátal som tam aj nejakú odchylku.



Void SetPosition

- Prvú časť majú všetky objekty rovnakú, prejdem všetky body *borders_* a prirátam k nim offset
- V druhej časti upravím podľa nových hodnôt objekty na vykreslenie

•

Draw.h/Draw.cpp
Draw dedí vlastnosti z QWidget, aby sme mohli používať metódy ako *mousePressEvent, mouseMoveEvent, MouseReleaseEvent* a *paintEvent.*
Draw si pamätá:

- *QPoint point* – bod, kde si budeme pri pohybe myši ukladať pozíciu
- *QPoint beginPoint* – uložíme bod po kliknutí
- *Tool tool, QColor color, int width, Body borders* – aktuálne nastavené hodnoty
- *Bool keyHold* – držanie tlačidla
- *Bool drawSave* – povie, či je objekt hotový a máme ho uložiť do stálych
- *QPen selectPen, QPen currentPen, QPen objectPen* – pero výberu, aktuálne pero, pero vybraného objektu
- *Vector<ObjectPtr> toDraw* – ObjectPtr je unique_ptr na Object, v toDraw sú uložené všetky objekty na vykreslenie
- *Bool polyDraw* – či je vykreslovaný polygón
- *VectorSizeChanged vsc*
    - pomocný struct
    - pamätá si poslednú veľkosť vektora a určí, či terajšia je väčšia / či sa vektor zmenil pomocou *bool changed*
- *Body tempLine* – pomocný vektor na vykreslenie jednotlivých čiar polygónu
- *size_t currentlySelected* – index vybraného objektu v toDraw
- *size_t clickedObject* – index teraz nakliknutého objektu v toDraw
- *bool fillObject* – či sa má objekt vyplniť
- *int onBorderIndex* – index hranice aktuálne vybraného objektu
- *void setToDefault()* – nastavuje na základné hodnoty pri zmene výberu nástroja
- *std::pair<int, int> isMouseOverObject(QPoint p) const* – vráti pár intov, prvý je index objektu v toDraw na ktorom je QPoint *p* a druhý je index hranice objektu, na ktorom je *p*. Vracia -1 ak nie je.

Ďalej v ňom máme sloty, ktoré nám menia hodnoty hore uvedených premenných v draw pomocou odoslaných signálov z jednotlivých objektov UI. Sú to:

- *void shapeChangedtoLine()* – zmení *tool* na *Tool::enLi*
- *void shapeChangedtoRectangle()* – zmení *tool* na *Tool::enRec*
- *void shapeChangedtoCircle()* – zmení *tool* na *Tool::enCir*
- *void shapeChangedtoPolygon()* – zmení *tool* na *Tool::enTr*
- *void pickModeOn()* – zmení *tool* na *Tool::enPick*
- *void newWidth(int w)* – nastaví *width* na w
- *void newColor(QColor c)* – nastaví *color* na c
- *void rotate(int r)* – nastaví *rotate_* vybraného objektu na r
- *void fill(bool f)* – true ak sa má vyplňovať

std::pair<int, int> isMouseOverObject(QPoint p) const

Najprv si vytvoríme novú transformáciu *QTransform backTrans*, ktorá predstavuje opačnú transformáciu k tej v objekte (rotate). Cez cyklus prechádzame všetky elementy v *toDraw*, vytváram si *backTrans* podľa rotácie v prechádzanom objekte. Teda, posuniem ju na prvý bod vykreslenia objektu, zrotujem a následne posuniem naspäť. Túto transformáciu invertujem pomocou *inverted()* a namapujem na *QPoint newp* pomocou *map()*. Potom zavolám *pointIsOnBorder(newp)* a výsledok uložím do dočasnej *borderIndex*. Ak nám vrati iné ako -1, teda na nejakú hranicu sme klikli, vrátime index objektu v *toDraw* a index hranice. Inak volám metódu *pointIsOver(newp)* daného objektu. Ak vráti true, funkcia vráti index v *toDraw* ale v druhej zložke bude mať -1. Ak nenájde nič, vracia pár (-1,-1).

Draw

Hodnoty na začiatku. *keyHold a drawSave* vypnuté, *tool* nemáme žiadny, farba je červená, hrúbka je 5, *polyDraw takisto vypnutý*, *filled* na false a *currentlySelected* na -1, čo bude značiť „nie je vybraný žiadny objekt“.

void setToDefault()

Po zmene nástroja sa vypne *polyDraw*, vyčistia sa body v *borders* a pokiaľ je vybraný nejaký objekt, nastaví sa jeho *selected_* na *false* a *currentlySelected* na -1.

void mousePressEvent(QMouseEvent *e)

Nakliknutie myši na *drawPlace.* Najprv zistíme, čo je *e*. Pokiaľ *e* je *Qt::LeftButton*, teda bolo stlačené ľavé tlačidlo, potrebujeme rozlišovať, či chceme posúvať existujúci objekt, meniť jeho veľkosť alebo vytvárať nový.

Teda, ak je vybraný *tool Tool::enPick*, chceme posúvať/resizovať objekt. Uložíme si našu dvojicu do *clickedObjectInfo* pomocou *isMouseOverObject(e->pos())* (vracia pozíciu *QMouseEvent*), nájdeme si index *clickedObj* objektu, na ktorý sme klikli a hranicu do *onBorderIndex*. Ak sme neklikli na žiadny objekt, zrušíme aktuálne vybraný. Ak sme na nejaký klikli, uložíme si polohu kliku do *beginPoint* a nastavíme danému objektu *Select(true).* Ak bol vybraný iný objekt, jeho *selected_* zrušíme.

Ak máme v *tool* akýkoľvek iný nástroj, len si uložíme *beginPoint*. Pokiaľ chceme kresliť polygón, musíme nastaviť *polyDraw* na *true*, hodiť prvý bod polygónu do *borders* a nastaviť vo *vsc* veľkosť na 1. Na konci vždy spustíme *keyHold*.

Ak nám *e->button()* vráti *Qt::RightButton*, ide o zmazanie objektu, čiže si rovnakým spôsobom ako som uviedol hore zistíme, na ktorý objekt sme klikli a pomocou *erase()* ho zmažeme z *toDraw*.

void mouseMoveEvent(QMouseEvent *e)

Pohyb myši po *drawPlace*. Pokiaľ ide o „pohyb“ (*e->type() == QEvent::MouseMove*), ukladáme pozíciu myši do *point*. Pokiaľ máme nastavený výber, bude sa ným kurzor myši meniť pri prechode nad objektom. Ak sme nad hranicou, objavia sa nám 4 šípky na resize, ak sme nad čiarou objektu, uvidíme otvorenú dlaň, pokiaľ ťaháme, dlaň sa zatvorí. Inak máme klasickú šípku.

void mouseReleaseEvent(QMouseEvent *e)

Pustenie myši. Pokiaľ kreslíme polygón (*polyDraw = true*), chceme overiť, či bod kde sme pustili tlačidlo nie je v okolí prvého bodu polygónu. Okolie myslené 10 pixelov na každú stranu. Ak áno, polygón je hotový a vypneme *polyDraw*. Inak pridáme nový bod do *borders*. Zakaždým ale na pustenie tlačidla vypneme *keyHold*.

void paintEvent(QPaintEvent *)

Kde sa deje všetko vykreslovanie. Vytvoríme si *QPainter painter(this)* a spustíme mu Antialiasing. Máme 3 fázy vykreslenia. Vykresľovanie počas *keyHold*, teda simulácia vytvárania nového objektu. Vykresľovanie počas *drawSave*, ktorý sa spúšťa bezprostredne po *keyHold*, aby sme zaručili, že hneď na pustenie tlačidla sa nám objekt uloží. Posledná časť je vykresľovanie všetkých objektov z *toDraw*.

keyHold časť

currentPen si nastavíme na aktuálne hodnoty *draw* a *painter* nastavíme na toto pero.

Máme switch, ktorý určí, aký objekt podľa vybraného nástroja chceme vykreslovať. Simulácia tvorby funguje tak, že na začiatku pri kliknutí na plochu sme si uložili počiatočný bod *beginPoint* a aktuálny pri pohybe máme v *point*. Čiže každý objekt budeme vykreslovať medzi nimi.

Pri polygóne (*Tool::enTr*) vytvárame len čiaru od posledného bodu *borders* k aktuálnemu. Všetky predošlé čiary polygónu sú totiž už uložené v *toDraw* (bude vysvetlené ďalej).

Pri výbere (*Tool::enPick*) rozdeľujeme 2 prípady. Či ide o hranicu a chceme meniť veľkosť alebo ide o čiaru a chceme posúvať. V prvom prípade si potrebujeme najprv zase náš bod, kam chcem presunúť novú hranicu namapovať na inverznú rotáciu. V druhom prípade posielame práve vybranému objektu aktuálny offset, teda rozdiel medzi *point* a *beginPoint*. Samozrejme po tomto poslaní sa nám aktuálna pozícia zmení a teda musíme posunúť *beginPoint.*

Spustí sa *drawSave.*

drawSave časť

Znova switch cez všetky objekty. Všetky objekty okrem polygónu fungujú na princípe, vložíme *beginPoint a point* (teraz je *point* na pozícii, kde sme myš pustili) do *borders*, vytvoríme pointer na Object s novou derivovanou classou, ktorá má v konstruktore *borders, color* a *width*. Následne ho pomocou move vložíme do *toDraw* a vyčístíme *borders*.

Polygón funguje trochu inak. Najprv musíme zistiť, či sa jeho kreslenie už dokončilo alebo nie.

Ak nie, zistíme pomocou našeho pomocného structu, či sa *borders* zväčšil. Ak áno, znamená to, že bol pridaný nový bod a teda musí byť uložená nová čiara. Čiže sa do *tempLine* vložia posledné 2 body *borders*, vytvorí sa čiara a vloží do *toDraw. tempLine* sa vyčistí.

Ak sme dokreslili, vybereme všetky čiary, ktoré sme doteraz „dočasne“ povkladali do *toDraw* (podľa počtu bodov v *borders*) a vytvoríme nový polygón.

toDraw časť

Cyklus, ktorý prechádza všetky objekty z *toDraw*. Nastaví object*Pen* podľa objektu, vytvorí transformáciu *QTransform transform*, vybere, či je objekt *selected (*nastaví *selectPen,* zvýrazní hranice a stred rotácie) alebo nie je a nastaví sa *objectPen*. Potom podľa *tool* vybere o aký objekt ide, nastaví transformáciu, vykreslí objekt a zruší ju.

Editor.h/editor.cpp

Spája UI s *draw* pomocou posielania connect signálov. Dedí z *QMainWindow*. Máme uložené nasledovné sloty (jednotlivé časti UI vysvetlené ďalej):

- *void on_colorButton_clicked()*
- *void on_actionSave_triggered()*
- *void on_actionExport_triggered()*
- *void on_actionClean_triggered()*
- *void on_actionSet_Canvas_triggered()*
- *void on_closeButton_clicked()*
- *void on_okButton_clicked()*
- *bool is_digit(std::string s)* – len kontroluje, či je celý string číslo

- *void on_actionLoad_triggered()*

Všetky reprezentujú konkrétnu akciu na objekte UI.

Prerobené signály sú *on_colorButton_clicked,* kde mám vytvorené Qt predrobené *QColorDialog*, teda okno, ktoré umožňuje používateľovi vybrať farbu. Tú si len uložím do *newColor* a pošlem signál *changeColor(newColor).*

Druhý je *on_actionExport_triggered(),* čo predstavuje hornú ľavú položku pod „File“. Na nakliknutie sa nám znova otvorí Qt predrobené okno, tentokrát *QFileDialog*. Používateľ vybere formát a cestu, kam sa má uložiť. Cestu si uložím do *QString imagePath*, následne pomocou *.grab()* a *.save()* uložím vykreslený *drawPlace* na *imagePath.*

Ďalej sú to z horného menu File *void on_actionSave_triggered() a void on_actionLoad_triggered().* Obidve obdobným spôsobom otvoria QFileDialog na špeciálny formát .edf. Save posiela všetky údaje do otvoreného streamu (filu) zavolaním metódy store z objekto *toDraw.* Load zase vyčistí aktuálny kanvas a pomocou prvých *tool* údajov z in streamu (loadovaného súboru) vytvára nové prázdne objekty na ktoré volá ich load.

void on_actionClean_triggered() volá clear na *toDraw.*

Posledné 3 záležitosti sa týkajú zmeny veľkosti kanvasu. Celú dobu je za kanvasom schované okno, kde sú dva Input Boxy výška a šírka a tlačidlá Ok a Close. Jednoducho sa vyberú stringy zadané do input boxov, skontroluje sa či sú to čísla, ak áno, zmeníme veľkosť kanvasu, ak nie, vyhodíme chybu.

Editor.ui

Formulár, kde sú uložené všetky objekty UI. Konkrétne ide o:

- *QPushButton circleButton* – nastaví nástroj na elipsu
- *QPushButton lineButton* – nastaví nástroj na čiaru
- *QPushButton rectangleButton* – nastaví nástroj na obdĺžnik
- *QPushButton polygonButton* – nastaví nástroj na polygón
- *QPushButton pickButton* – nastaví nástroj na výber
- *QPushButton colorButton* – otvorí okno na výber farby
- *QSlider sizeSlider* – nastaví hrúbku čiary od 1 po 10 posúvaním smerom nahor, defaultne 5
    - *QLabel sizeSlider_label* – nadpis „Size“ nad *sizeSlider*
- *QSlider rotateSlider* – nastaví rotáciu aktuálneho objektu od 0 po 360 posúvaním smerom nahor, defaultne 0
    - *QLabel rotateSlider_label* – nadpis „Rotate“ nad *rotateSlider*
- *QWidget drawPlace* – plocha na vykreslovanie, dedí z class *draw*
- *QWidget canvasSize*
    - *QLabel canvasSize_label* – „Set Canvas Size“
    - *QPushButton closeButton* – schová okno
    - *QPushButton okButton* – uloží nové údaje
    - *QLabel hw_label* – popísok „Width Height“
    - *QLineEdit widthEdit* – input pre šírku
    - *QLineEdit heightEdit* – input pre výšku
- *QMenuBar menuBar*
    - *QMenu menuFile* – otvorí rolovacie menu
        - *QAction actionSave* – spustí „Save“
        - *QAction actionExport* – spustí „Export“
        - *QAction actionLoad* – spustí „Load“

- *QAction actionClean* – spustí „Clean“
- *QAction actionSet_Canvas* – spustí „Set Canvas“

POZNÁMKY NA ZÁVER

Rotácia je spravená takýmto netradičným spôsobom kvôli veľkým komplikáciám so zmenou stredu objektu pri jeho resize keď je zrotovaný.