Grafický editor Pablo

Y36PJC – zpráva k semestrální práci

Ondřej Macoszek

Verze — 1.0

ZADÁNÍ

Zadání dle Progtestu

Realizujte 2D vektorový grafický editor. Program bude ovládán interaktivně. Bude umožnovat vkládání 2D grafických primitiv (úsečka, obdélník, oblouk, ...) a nastavovat jejich vlastnosti (barva, tloušťka čáry, výplň, ...). Grafická primitiva půjde kdykoliv během editace opravovat (měnit jejich tvar, velikost, vlastnosti, ...).

Volba prostředí není omezena (WinAPI, X lib, Qt, GTK, ...).

Upřesnění

Interaktivita ovládání bude fungovat pomocí ovládacích panelů (pro správu vrstev a vlastností objektů). Vkládání objektů bude realizováno pomocí tlačítek, které vloží hotový objekt na plátno, ovšem implementace bude počítat s možným budoucím zdokonalením způsobu ovládání (např. budování cesty křivky postupným klikání, vytvoření obdélníku/kruhu tahem).

VIZE ŘEŠENÍ

Architektura

Každý z kreslených objektů lze zobecnit na množinu kontrolních bodů které tvar objektu určují. Kreslené objekty mají také vždy určitou sadu vlastností jako ohraničení a pozadí. Návrh musí umožnit vhodně udržovat tyto informace stranou od jejich reprezentace na plátně, aby nevznikaly nadbytečné udržující stejné/podobné informace - toto také zajistí snadnější implementaci případné budoucí funkcionality ukládání nakreslených objektů. Pro identifikaci typizovaných tvarů (obdélník, kruh, křivka) a implementaci vlastností neposkytovaných obecným kresleným objektem budou tyto tvary upřesněny zvláštními třídami rozšiřujícími obecný kreslený objekt.

Manipulaci s těmito objekty budou mít v maximální možné míře na starosti správci, v implementaci označovaní jako controllery. Nejdůležitějším bude správce vrstev (layer controller), který bude udržovat všechny kreslené objekty v hierarchii vrstev – v současné verzi budou hierarchie fungovat pouze do jedné úrovně (vrstvy se nebudou zanořovat).

Pro zprovoznění interaktivity bude nezbytný správce režimu nástrojů (tool controller), který bude udržovat informaci o aktuálně aktivním nástroji, jeho pomocných informací a bude obstarávat zpracování událostí přijatých z prezentační vrstvy.

Hlášení změn v kreslených objektech

Díky mechanismu slotů-signálů frameworku Qt je možné nastavit aby se každý kreslený objekt ohlásil svému zaregistrovanému posluchači jako aktualizovaný, a posluchač pak na to patřičně zareagoval (např překreslením plátna, posláním informace o aktualizaci dál,...).

V uživatelském rozhraní při výběru vrstvy v panelu vrstev, se vlastnosti kresleného objektu uloženého v této vrstvě zobrazí v panelu vlastností aby bylo možné upravovat aktuální stav.

Při změně vlastností kresleného objektu nebo jeho tvaru, či pozice v hierarchii vrstev se překreslí plátno.

Optimalizace vykreslování během úprav

Každá úprava kreslených objektu se skládá ze tří fází

- zahájení: stisknutí tlačítka myši
- úpravy kontrolních bodů: tah myši a v závislosti na režmu ovládání případná úprava příslušných kontrolních bodů
- ukončení: uvolnění tlačítka myši

Pro změnu některého bodu je potřeba znát jeho novou pozici, ta se zjistí z aktuální pozice myši na plátně. Editor bude umožňovat úpravu jen jednoho kontrolních bodu během operace kliknutí a tahu myší. Toto znamená že během prvotního stisknutí tlačítka myši bude vybrán určitý kontrolní bod (nejbližší aktuální pozici myši a spadající do operačního radiusu myši) a pouze s tímto bodem se bude manipulovat během tahu myši až do uvolnění myši. Operační rádius myši bude znamenat minimální vzdálenost kterou musí mít kurzor myši ke kontrolnímu bodu aby jej mohla aktivovat a manipulovat s ním.

Výběr kontrolního bodu po stisknutí myši bude realizován průchodem všech vrstev s kreslenými objekty a otestováním vzdálenosti jejich kontrolní bodů. Jakmile je jeden takový bod nalezen, bude uložen stranou a průchod zastaven. Tím že se průchod provádí při stisknutí myši a ne při tahu dojde k výraznému zrychlení překreslování plátna.

IMPLEMENTACE

Návrh aplikace

Všechen kód bude rozdělen do několika oblastí vyjádřených strukturou a názvy složek.

- business – obchodní logika aplikace
  - controller – správci jsou používání k manipulaci s entitami
    - LayerController, PropertiesController, ToolController

    - entity – nositelé důležitých informací pro obchodní logiku programu
        - Background, Stroke, Layer, VectorEntity, Circle, Rectangle, Path
- resources – úložiště různorodých souborů
    - icons – ikony používané v prezentační vrstvě
    - resources.qrc – definice aliasů pro soubory které jsou používány v programu
- ui – prezentační vrstva aplikace
    - component – definice uživatelského rozhraní
        - Canvas, LayersPanel, PropertiesPanel, MainWindow
        - soubory s příponou ui – xml definice uživatelského rozhraní z nástroje QT Designer
- main.cpp – vstupní bod do aplikace

Vývojové prostředí

Používal jsem prostředí NetBeans 6.8 na platformě Linux (ubutnu), které má vestavěnou podporu pro vývoj Qt aplikací. Dále pro vizuální návrh určitých obecných částí uživatelského rozhraní jsem použil aplikaci Qt Designer produkující xml definice s popisem rozhraní, které pak lze pomocí nástroje UIC přeložit do zdrojového kódu.

Používané knihovny

Pro usnadnění vývoje jsem si vybral Qt framework. V základní distribuci poskytuje knihovny pro budování GUI a sadu nástrojů jako qmake a uic obstaravající správnou kompilaci aplikace v Qt.

UIC slouží k vytvoření zdrojových kódu navrženého uživatelského rozhraní, tak že je pak možné tyto generované třídy rozšiřovat a přizpůsobovat potřebám aplikace.

Qmake slouží jednak k vytvoření makefile souboru aplikace, ale také k přípravě pro Qt specifických součástí pro následnou kompilaci celého projektu. Zajišťuje také např vyhledání resources a převod souborů uživatelského rozhraní do zdrojového kódu.

Qmake se nad složkou se zdrojovými kódy používá následovně:

```
qmake -project  // vytvoří pro Qt specifický projekt
qmake pablo.pro  // na základě Qt projektu vygeneruje makefile specifický pro platformu

make // klasický přikaz součásti GCC spustí sestavení aplikace
```

Vygenerovaný makefile je možné přizpůsobit. Pro potřeby požadavků semestrální práce je nutné obohatit následující řádek s proměnnou CXXFLAGS o požadované příkazy:

```
CXXFLAGS      = -pipe -O2 -Wall -Wno-long-long -pedantic -D_REENTRANT $(DEFINES)
```

Myslím že by bylo dobré zmínit implementační záležitosti kolem třídy QObject, která je základní třídou pro většinu tříd ve frameworku Qt.

- Třída zprovozňuje funkcionalitu Slots-Signals která je velmi podobná návrhovému vzoru Observer-Observable a umožňuje objektům mezi sebou se propojit a komunikovat
- Neumožňuje kopírovat odvozené třídy (copyconstructor a operátor = jsou zakázány)
- Udržuje informace o svých dětech (objektech které jsou takto explicitně nastaveny) a v případě že se rodičovský objekt začne rušit, nejprve on sám zruší všechny své potomky

ZÁVĚR

Fungování

Požadavky v zadání projektu jsou splněny všechny a funguji správně.

Nedostatky

Pro efektivní kreslení by bylo vhodné vylepšit správce nástrojů, hlavně budování křivek (postupným klikáním cesty).

Vykreslování objektů je specifické pro každou třídu odvozenou z VectorEntity, proto abych se vyhnul dynamic_cast příkazu, použil jsem enumeraci definovanou uvnitř VectorEntity na základně které se idenfikuje odvozená třída objektu, objekt se následně přetypuje a pošle ke zpracování do odpovídající pro tento typ přetižené vykreslovací metody. Byla zde možnost implementovat vykreslování jako virtuální metodu pro každou odvozenou třídu třídy VectorEntity, avšak tomu jsem se chtěl vyhnout protože vykreslování do business vrstvy nepatří a příliš by zvýšilo závislost na prezentační vrstvě. Ideální, ale více pracné, by bylo kdyby každá třída odvozená z VectorEntity měla svůj „view“ v prezentační vrstvě který by zmíněný princip s virtuální metodu implementoval.