Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Dlaždičkovač

Plzeň, 2013 Lukáš Hain

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne

Lukáš Hain

Abstract

The goal of this bachelor's thesis is to develop multiplatform and easy to use program, for laying tiles. The program is able to create a database of tiles and also manage it. It also has a room editor, where you can draw your own ground plans. The output is graphical interpretation of tiles and also costs of all materials used for it. In the first part of this work, there's theoretical analysis of useful algorithms like polygon clipping and others. The second part of this work is about real world usage of those algorithms. There are also the results of research among users of this program about their opinion on user interface.

Cílem této bakalářské práce, je vyvinout multiplatformní a snadno použitelný program pro pokládku dlaždic. Program je schopný vytvořit a spravovat databázi dlaždic. Dále obsahuje editor místností, který umožňuje návrh půdorysu místnosti, ve které bude pokládka probíhat. Výstupem je grafické ztvárnění položené dlažby a také množství a cena použitých materiálů. První část práce obsahuje teoretickou analýzu důležitých algoritmů, jako například ořezávání polygonů a další. Ve druhé části je znázorněno použití těchto algoritmů v programu. Zároveň jsou zde uvedeny výsledky průzkumu mezi uživateli ohledně kvality uživatelského rozhraní aplikace.

Rejstřík

1 Úvod

Cílem této práce je vytvoření multiplatformního, uživatelsky přívětivého programu pro pokládku dlažby. Program by měl umožňovat vytváření databáze dlaždic a její správu. Dále pak jednoduchý návrh místností, ve kterých se dlažba bude pokládat. Výstupem bude grafický návrh, spotřeba položené krytiny a celkové náklady s tím spojené.

V první části se práce zabývá teoretickým popisem možných řešení problémů, které při vývoji takovéto aplikace mohou nastat. A to včetně problémů s rychlostí vykreslování mnoha objektů, detekcí kolizí nebo průniky polygonů.

Realizační část se již plně věnuje implementovaným vlastnostem. A také reálnému nasazení algoritmů a postupů zmíněných v první části. Jsou zde také zveřejněny výsledky průzkumu mezi uživateli programu.

2 Java

2.1 Historie

Java[1] je programovací jazyk vyvinutý firmou Sun Microsystems. Vývojáři chtěli dát světu jednoduchý objektový jazyk, který bude spustitelný na všech platformách a který bude programátorům povědomý. Inspirovali se, u do té doby pravděpodobně nejlepšího OO (objektově orientovaného) jazyka, který byl dostupný. Tím bylo C++. Tento jazyk je objektovou nadstavbou původně procedurálního jazyka C. Největší výhodou C++ je vysoký výkon, naopak slabinou je jeho extrémní rozsáhlost a komplexita. Proto se inženýři v Sunu rozhodli vzít to nejlepší z C++ a převést to do nového jazyka.

Vzniká nový multiplatformní, objektově orientovaný jazyk. Je mnohem jednodušší než původní C/C++, jsou například odstraněny příkazy způsobující nepřehlednost kódu jako GOTO. Přes to, že je Java interpretovaný jazyk dosahuje dobrých výpočetních výkonů. Díky těmto vlastnostem je dnes Java jedním z nejrozšířenějších jazyků, jak na klasických počítačích, tak v mobilních zařízeních (platforma Google Android).

2.2 UI

2.2.1 Počátky

První počítače uživatelské rozhraní[2] jako takové vůbec neobsahovaly. Jejich výstup byl často řešen přes tiskárnu. Teprve později se začaly používat obrazovky. Zpočátku se používalo tzv. CLI (Command Line Interface), neboli rozhraní příkazové řádky. Jeho implementace je velmi jednoduchá a náročnost vykreslení textu téměř nulová. Je třeba si ovšem uvědomit, že CLI, bylo používáno v 70. a 80. letech. Výkon procesoru tehdy dosahoval jednotek MHz a paměti měli velikost v řádech jednotek až desítek kB. Toto rozhraní však stále nebylo to pravé pro „běžné“ uživatele. XEROX ve svých laboratořích pracoval na něčem, co to mělo změnit.

2.2.2 GUI

GUI (Graphical User Interface), grafické uživatelské rozhraní bylo něco do té doby nevídaného. Tento převratný nápad zaujal i Steva Jobse, který se přijel do XEROX PARC (Palo Alto Research Center) podívat. V tom co tam viděl, spatřil budoucnost počítačů. V roce 1984 Apple vydává svůj legendární Macintosh. Lidé jsou nadšení grafickým rozhraním. Počítač konečně dostává lidskou tvář. Člověk si už nemusí pamatovat příkazy, ale může použít myš pro otevírání, přesouvání, spouštění atd. Již tedy není potřeba obsáhlý manuál k užívání počítače a nic nebrání masovému rozšíření počítačů mezi širokou veřejnost.

2.3 Prostředky pro tvorbu GUI v Jave

V Jave[3] existuje několik knihoven pro tvorbu GUI. Některé jsou vyvíjeny přímo firmou Oracle (AWT, Swing a JavaFX), další alternativy poskytuje například Eclipse se svým SWT. A nebylo by vhodné opomenout ani menší, přesto kvalitní projekt jako je SwingX.

2.3.1 AWT

AWT neboli Abstract Window Toolkit[4] je knihovna pro tvorbu GUI přítomná již v první verzi Javy v roce 1996. Využívá nativních knihoven systému, na kterém je spuštěna. Obsahuje vše nutné pro tvorbu základního GUI. Mezi jeho výhody patří rychlost. Vděčí za ní využití nativních knihoven. Toto je zároveň největší nedostatek. Jinak multiplatformní jazyk nemůže mít platformě závislé GUI. AWT proto bylo velmi brzy (již v Java 1.2) nahrazeno Swingem (viz. níže). AWT však nebylo nahrazeno zcela. Stále se používá například při reakcích na události jako je stisk klávesy nebo při vykreslování grafiky.

2.3.2 Swing

Swing[4] je aktuálně nejpoužívanější knihovnou pro tvorbu GUI v Javě. Obsahuje všechny nezbytné komponenty pro tvorbu vizuálně i funkčně bohatého GUI, jako ikony, tool-tipy nebo popup menu. Oproti předchůdci již nabízí platformní nezávislost. Vzhled celého rozhraní lze navíc změnit použitím jiného Look and Feel.

Upravovat lze i jednotlivé komponenty nahrazením jejich rendereru (viz. níže) vlastním kódem. Ani výkonem nezaostává za AWT a nabízí se tak jako vhodný kandidát při vytváření uživatelského rozhraní.

2.3.3 JavaFX

Nejnovější přírůstek[5] do oficiálně vyvíjených součástí Javy pro tvorbu GUI. První verze vychází v prosinci 2008. Nebyla u vývojářů přijata s velkým nadšením. Největším problémem byl skriptovací jazyk nazývaný JavaFX Script, který bylo nutné při vývoji použít. S příchodem verze nové se toto mění. JavaFX 2.0 je implementována jako standardní Java knihovna a již není potřeba ovládat nový jazyk. Od Javy 8 bude JavaFX součástí standardního balíku. Nyní je potřeba instalace JavaFX SDK.

JavaFX aktuálně nabývá na popularitě. Je to hlavně zásluhou nových, zejména multimediálních funkcí, jako je nativní přehrávání H.264 videa nebo možnosti vizuální úpravy komponent pomocí jednoduchých CSS stylů.

2.4 Oddělení GUI a výkonného kódu

Při vytváření programu je dobré mít na paměti oddělení vizuální a funkční části programu. Napomáhá to nejen efektivní úpravě kódu, ale třeba snadno umožní změnu celého GUI, aniž by byla nutná nějaká změna nebo znalost funkčního kódu. Existuje několik základních praktik, které mohou pomoci s tímto úkolem.

2.4.1 MVC

Neboli Model-View-Controller. Jak již název napovídá takový program se skládá ze 3 vrstev.

Model

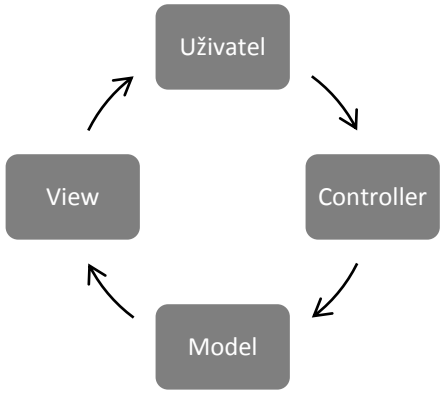
Obsahuje pole, seznamy, stromy… Všechna data, která aplikace potřebuje pro svůj běh. Dále je zde obsažen výkonný kód pro práci s daty.

View

Zahrnuje vizuální stránku aplikace. Popisuje layouty (rozložení komponent) i komponenty samotné.

Controller

Zpracovává akce uživatele. Kliknutí, vložení textu… Vše co uživatel provede a je relevantní pro aplikaci tato vrstva zaznamená a zašle ke zpracování zpět modelu.



Obrázek 2.1 Diagram spolupráce mezi vrstvami MVC

Jak je jasně vidět na obrázku 2.1 uživatel vidí GUI, které mu zobrazuje View. Při interakci aktivuje Controller, který upraví model. Model pak následně upraví View. Toto se opakuje v nekonečné smyčce.

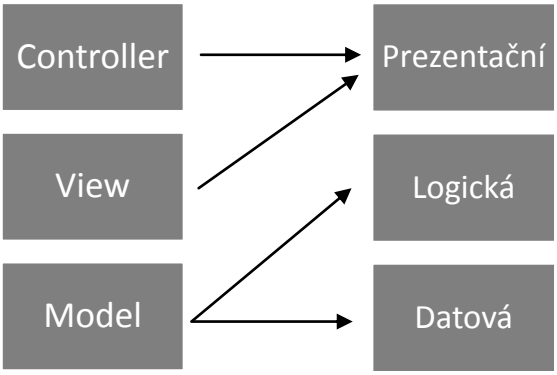
Dobrou praktikou je při změně modelu automaticky volat změnu komponenty s tím spojenou. K tomu může dobře posloužit návrhový vzor Observer Pattern.

2.4.2 Observer Pattern

Je návrhový vzor založený na principu „pozorující“ a „pozorovaný“ (od anglického observer a observable). Každý pozorovaný má seznam pozorovatelů, kterým má dát zprávu, pokud se jeho stav nějak změní. Pozorující má metodu (většinou nazývanou `update()`), kterou mu pozorovaný oznamuje změnu. Na tuto změnu pak lze příslušně reagovat např. změnou hodnoty progress baru.

2.4.3 Třívrstvá archtektura

Na první pohled velmi podobné MVC. Také obsahuje prezentační, logickou a datovou vrstvu. Vrstvy třívrstvé architektury mají podobnou funkci jako u MVC. Jen jsou jinak rozděleny.



Obrázek 2.2 Porovnání MVC a třívrstvé architektury

Dále je rozdíl také v přístupu k datům. Zatím, co u MVC může Controller přímo k datům modelu. U třívrstvé architektury musí všechna komunikace probíhat přes prostřední logickou vrstvu.

3 Grafika

3.1 Vykreslování

Vykreslování se dá dle míry obecnosti a součinnosti s hardwarem rozdělit do dvou základních kategorií.

3.1.1 Vysokoúrovňové vykreslování

Je naprosto platformě nezávislé. Poskytuje stejný výsledek na všech platformách. Veškeré pokročilé záležitosti řeší konkrétní implementace jazyka, který je pro vytvoření GUI použit. O vykreslování se v tomto případě stará softwarová vrstva, která vykreslí ze zadané scény obrázek a ten pošle grafické kartě k zobrazení na monitoru. Pro svou činnost používá výkon procesoru a tím ubírá samotnému programu na dostupném výkonu. V posledních zhruba pěti letech se situace zlepšuje a vývojáři se snaží využít vysokého výkonu grafické karty pro akceleraci vykreslování. Vysokoúrovňové vykreslování neposkytuje však takovou volnost, jako nízkoúrovňové a někdy je třeba dělat kompromisy.

3.1.2 Nízkoúrovňové vykreslování

Využívá dostupné funkce grafického akcelerátoru. Programátor používá nízkoúrovňové knihovny jako například OpenGL nebo DirectX, které vezmou napsaný kód a přeloží na instrukce pro grafickou kartu. Tento způsob umožňuje kreslit jen jednoduché objekty (body, čáry, trojúhelníky) nebo v závislosti na hardwarové podpoře jen body na bitmapě. Z toho vyplývá hned několik nevýhod. Vykreslování touto metodou je programátorsky velice náročné a zdlouhavé. Většinou je zde potřeba použít i nízkoúrovňový jazyk jako je C++ s manuální správou paměti. Grafika naprogramovaná na této úrovni nemusí být přenositelná. Naopak obrovskou výhodou je rychlost samotného vykreslování, proto se používá hlavně v náročné 3D grafice, zejména ve hrách a pokročilých inženýrských aplikacích.

3.2 Clipping

Jak již bylo řečeno, softwarové vykreslování je oproti hardwarovému řešení násobně pomalejší. Je tedy nutné se o to více snažit při optimalizaci. Nejčastější optimalizací v grafice je clipping (česky ořezávání)[8]. Ne vždy je potřeba vykreslovat celou scénu. Obzvláště ve 3D grafice, kdy lze například vidět vysokou zeď ale už ne co je za ní. To znamená, že lze vynechat vykreslení objektů, které by ve výsledné scéně stejně nebyly vidět. Základní podmínkou je mít velice efektivní algoritmus, pomocí něhož lze zjistit, které části scény kreslit a které už ne. Kdyby tato část kódu měla vysokou složitost, mohlo by se vykreslování dokonce zpomalit a nijak by se to nevyplatilo.

Tato práce se však bude věnovat pouze clippingu ve 2D, kde se ořezávané objekty nemohou překrývat. Navíc jako ořezávací objekt bude použit obdélník, který vytváří okno programu.

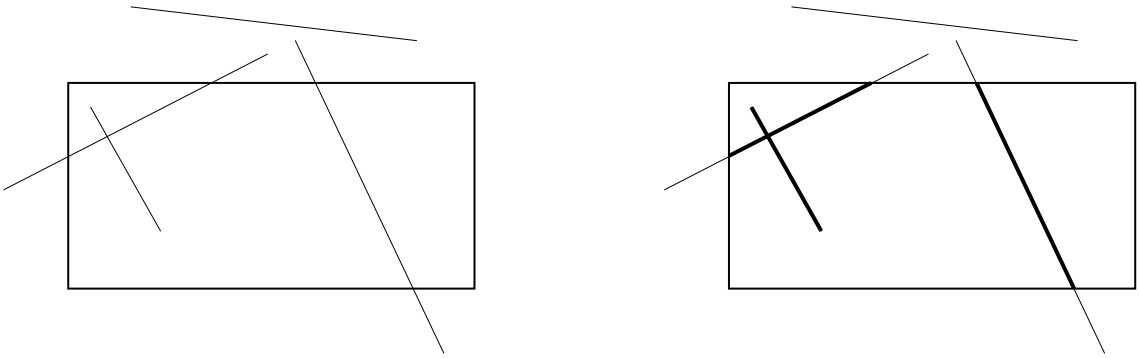
3.2.1 Cohen-Shuterlandův algoritmus

Cohen-Shuterlandův algoritmus[8] rozdělí 2D prostor do 9 oblastí. Někdy je také označován jako Left-Right-Top-Bottom algoritmus (LRTB).

1001	1000	1010
0001	0000	0010
0101	0100	0110

Obrázek 3.1 Rozdělení prostoru pomocí Cohen-Shutterlandova algoritmu

Označení LRTB odpovídá binární hodnotě polohy oblasti. Jako příklad budiž uvedena oblast s hodnotou 1001. Nachází se v horní (top) řádce a v levém (left) sloupci. Hodnoty na pozicích top a left budou rovny jedné, jinak nuly. Objekty, které se nakonec vykreslí, musí být v oblasti 0000. Tyto obrazce se mohou vykreslit celé, nebo se mohou dále a přesněji oříznout. Výsledek může vypadat následovně.



Obrázek 3.2 Ukázka ořezávání

Nejjednodušší verze tohoto algoritmu pouze zjišťuje, zda je daný objekt možné ve scéně spatřit. Nezabývá se přesným oříznutím. To znamená, že i když je vidět z modelu jen jeden bod bude celý vykreslen, bez ohledu na jeho velikost. V některých případech, zvláště pokud je vykreslováno velké množství jednoduchých objektů, je tato verze naprosto dostačující. Algoritmus vypadá takto.

```
for(všechny objekty o){
      if(o.x + o.sirka < obrX.min || o.x > obrX.max){
            continue;
      } else if(o.y + o.vyska < obrY.min || o.x > obr.max){
            continue;
      } else{
            vykresli(o);
      }
}
```

Kód 3.1 Zkáladní verze Cohen-Shutterlandova algoritmu

V tomto pseudokódu je předpoklad, že každý objekt vrací jako x a y souřadnice svého levého horního rohu. Pokud by v objektu byla uložena i souřadnice pravého dolního bodu, bylo by možné ušetřit operace sčítání při každé kontrole.

3.2.2 Weiler–Atherton

Další příklad ořezávacího algoritmu[8]. Tentokrát jde o ořezávání dvou polygonů. Polygon je geometrický útvar skládající se z N bodů. Tyto body se teoreticky mohou i pronikat. U příkladu dlaždic je tato možnost nereálná a bude zanedbána. Jedním z předpokladů je také to, že body polygonů jsou řazeny stejně a to buď ve směru, nebo proti směru hodinových ručiček.

```
Polygon weilerAtheron(Polygon p1, Polygon p2){
     najdiPruseciky(p1, p2);
     int vstupniBod = najdiVstupniBod(p1, p2);
     return spojVysledneBody();
}
```

Kód 3.2 Kostra Weiler-Athertonova algoritmu

Toto je nejzákladnější struktura algoritmu. Jednotlivé části budou dále popsány podrobněji.

Průsečíky

V první části jsou vyhledány všechny průsečíky dvou polygonů. Klasický případ dvojitého cyklu. Jsou proti sobě otestovány všechny dvojice úseček, které tvoří hranici polygonu. Pokud je průnik nalezen, okamžitě bude zařazen do obou polynomů, jako jeden z bodů. Je nezbytně nutné zde dodržet orientaci bodů, která již byla zmíněna.

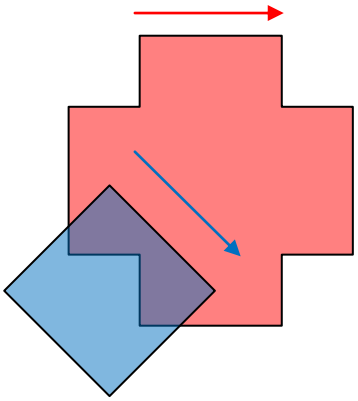
```
for(vsechny body p1){
     for(vsechny body p2){
          if(prusecik(p1.bod, p2.bod) != null){
               p1.add(prusecik); p2.add(prusecik);
          }
     }
}
```

Kód 3.3 Pseudokód pro umístění průsečíků do polygonů

Pokud by nebylo dodrženo uspořádání bodů v polygonu, nebylo by umístění nových bodů správné. Při následném průchodu seznamů by vznikl úplně odlišný polygon. O složitosti této části kód asi není nutné dlouho spekulovat O(n*m).

Vstupní bod

Tuto část bude nejpraktičtější popsat pomocí obrázku.

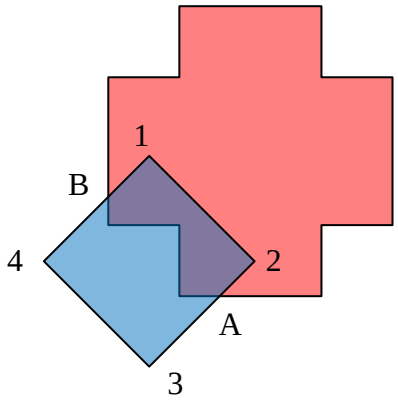


Obrázek 3.3 Ukázka protínajících se polygonů

Například nechť existují tyto dva polygony a jejich orientace je ve směru šipky příslušné barvy na obr. Je úkolem oříznout modrý polygon tak, aby byl celou svou plochou uvnitř červeného polygonu. Vstupní bod je takový bod, který splňuje následující požadavky.

- Je oběma polygonům společný (je to průsečík)
- Jeho předchůdce se nenachází v řezajícím polygonu (v červeném) nebo je to také průsečík
- Jeho následovník leží uvnitř řezajícího polygonu nebo je to také průsečík

Tento bod existuje vždy, existuje-li průnik mezi polygony.



Obrázek 3.4 Nalezené průsečíky A a B

První úkol je tedy mezi body ořezávaného polygonu nalézt průsečíky. První nalezený průsečík je bod s označením A. Bude tedy otestován. Nesplňuje však již druhé kritérium. Jeho předchůdce (bod 1) je uvnitř řezajícího polygonu a je to původní bod. Algoritmus tedy pokračuje. Je nalezen průsečík B. Jeho předchůdce (bod 3) je mimo řezající a následovník (bod 1) je uvnitř. Bod B splnil všechny tři podmínky. Je označen jako vstupní bod a jeho index je vrácen jako návratová hodnota metody. Pro získání indexu bylo vynaloženo O(n) času.

Vytvoření výsledného polygonu

Na řadu přichází poslední část algoritmu. Z předchozí části je známý index počátečního bodu.

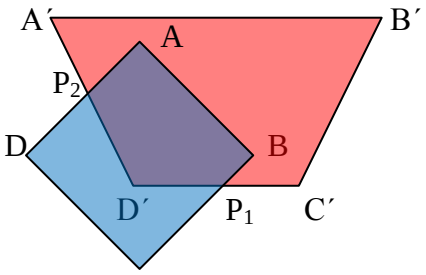
```
List aktualniSeznam = p2.getBody();
Point aktualniBod = aktualniSeznam.get(index);
while(aktualniBod.isNavstiven() == false){
     vyslednyPolygon.add(aktualniBod);
     aktualniBod.setNavstiven(true);
     if(aktualniBod.jePrusecik(){
          index = prepocitejIndex(aktualniSeznam, index);
          aktualniSeznam = vymenSeznamy(aktualniSeznam);
          aktualniSeznam.get(index).setNavstiven(true);
     }
```

```
        index++;
        index = index % aktualniSezam.size();
        aktualniBod = aktualniSeznam.get(index);
}
return vyslednyPolygon;
```

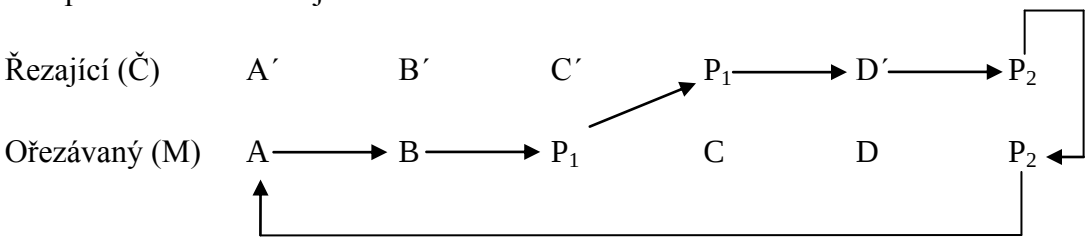
Kód 3.4 Sestavení výsledného polygonu

Průchod bodů začíná na obvodu řezajícího polygonu. Aktuální bod je přidán do výsledného seznamu a označen jako navštívený. Pokud je aktuální bod průsečíkem musí se seznamy bodů vyměnit. To zahrnuje přepočítání indexu do druhého seznamu pro ten stejný bod. Bod je označen za navštívený i ve druhém seznamu. Dále je zvýšen index v seznamu a je načten nový bod. Takto bude algoritmus pokračovat, dokud nenarazí na bod, který již byl prohledán.

Názornější bude ukázat seznamy a vyznačit v nich výsledný polygon. Pro větší přehlednost byl původnímu řezajícímu polygonu snížen počet vrcholů.



Obrázek 3.5 Označené polygony s průsečíky



Obrázek 3.6 Diagram, jak je nový polygon sestaven

Výhodou tohoto algoritmu je možnost takto zpracovat všechny další části těchto polygonů. Stačí jen opakovat třetí krok, dokud existují vrcholy ořezávaného, které

nebyly zpracovány. Celková výpočetní složitost algoritmu je O(n*m) + O(n) + O(n+m) = O(n*m), kde n a m jsou počty vrcholů polygonů.

3.3 Afinní transformace

Afinní transformace[7] jsou jedny z nejpoužívanějších operací v počítačové grafice. Existují tři základní afinní operace.

3.3.1 Translace

Neboli posunutí[9], je nejzákladnější operací v počítačové grafice. V tomto případě je bod $A = [A_x, A_y]^T$ posouván podle vektoru $v = [v_x, v_y]^T$ a je takto získán výsledný bod $B = [B_x, B_y]^T$. Nyní již jen zbývá celý vztah zapsat do matice.

$$\begin{pmatrix} B_x \\ B_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix}$$

3.3.2 Scaling

Používá[9] se ke změně velikosti objektů ve scéně. U této operace je vcelku matoucí použít frázi změna velikosti bodu. Je potřeba, si ale uvědomit, že je to celý objekt, který je zvětšovaný a všechny objekty se skládají z bodů. Vztah pro scaling je následující.

$$\begin{pmatrix} B_x \\ B_y \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix}$$

s_x a s_y jsou koeficienty zvětšení v příslušných osách. To znamená, že touto operací lze objekty nejen zvětšovat a zmenšovat, ale také deformovat. Pokud například s_x zůstane 1 a s_y bude změněna na 3. Výsledný obraz bude stejně široký jako původní ale bude třikrát vyšší.

3.3.3 Rotace

Rotace[9] je nesložitější operací z afinních transformací. Umožňuje otáčet bod kolem počátku o zadaný úhel φ.

$$\begin{pmatrix} B_x \\ B_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} A_x \\ A_y \\ 1 \end{pmatrix}$$

4 Tvorba vlastních komponent

Při vytváření komplexních aplikací s uživatelským rozhraním se pravděpodobně nelze vyhnout úpravám nebo dokonce návrhu vlastních komponent. Pokud se zaměříme jen na úpravu komponent, existují v zásadě dvě možnosti.

4.1 Vzhled

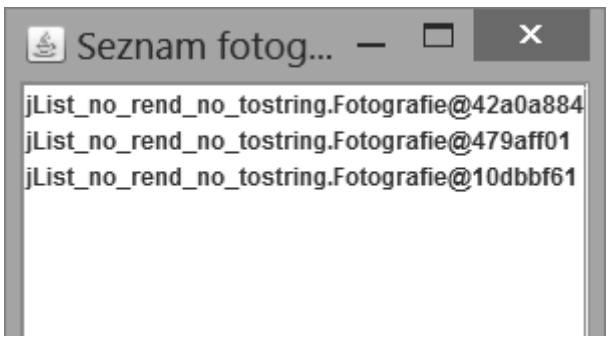
První možností je jednoduchá změna vzhledu. Vizuální stránku některých lze ovlivnit pomocí třídy Renderer. Komponenty jako `JTable`, `JList` nebo `JTree` v knihovně Swing mají vlastní renderer, který přesně specifikuje vzhled. Obsahuje např. preferovanou velikost, barvu podkladu nebo i hranice při označení. Jednoduše vše ohledně grafického ztvárnění. Příkladem může být tato ukázka. Existuje objekt Fotografie, který má následující strukturu.

```
public class Fotografie {
     private String nazev;
     private ImageIcon ikona;

}
```

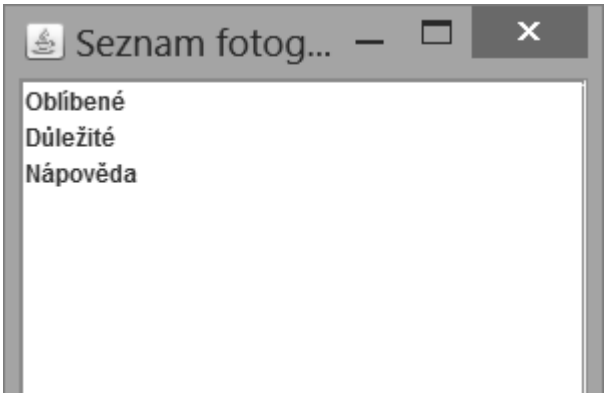
Kód 4.1 Atributy třídy `Fotografie`

Jestliže je tento objekt vložen do `JListu` se standardním Rendererem, nezobrazí se ani název ani ikona obrázku. Vypíše se jen cesta ke třídě následovaná hash kódem objektu.



Obrázek 4.1 Vzhled seznamu bez metody `toString()`

Přidáním metody `toString()` do třídy `Fotografie` je možné zajistit alespoň výpis názvu obrázku.

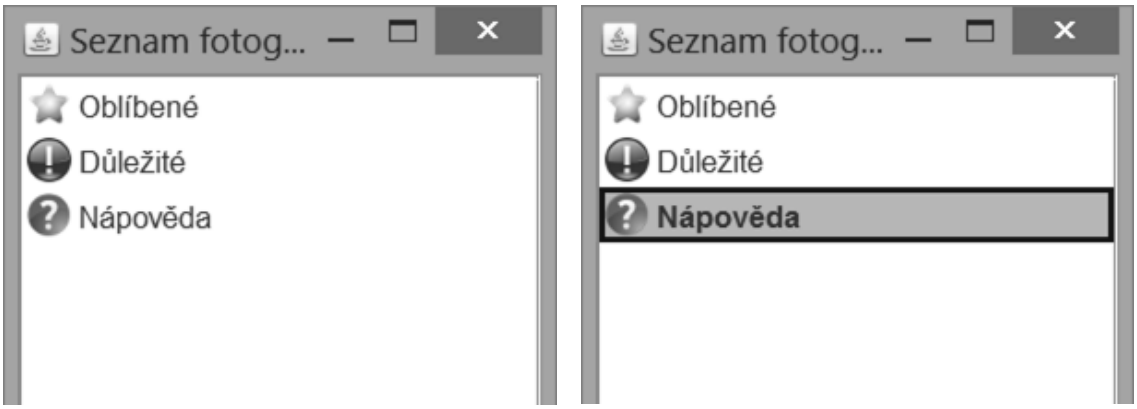


Obrázek 4.2 Vzhled seznamu s metodou `toString()`

Tato varianta je již funkční, ale většina uživatelů by jistě ocenila i náhled obrázku. Takto komplexní úlohu již nelze vyřešit pouhým přidáním atributu či jiným triviálním způsobem. Je nutné nahradit výchozí renderer buňky `JListu` vlastní implementací. Po aplikování rendereru pomocí příkazu.

```
seznam.setCellRenderer(new FotografieCellRenderer());
```

Je výsledek následující. Pozn. zdrojové kódy všech tří příkladů je možno nalézt v příloze.



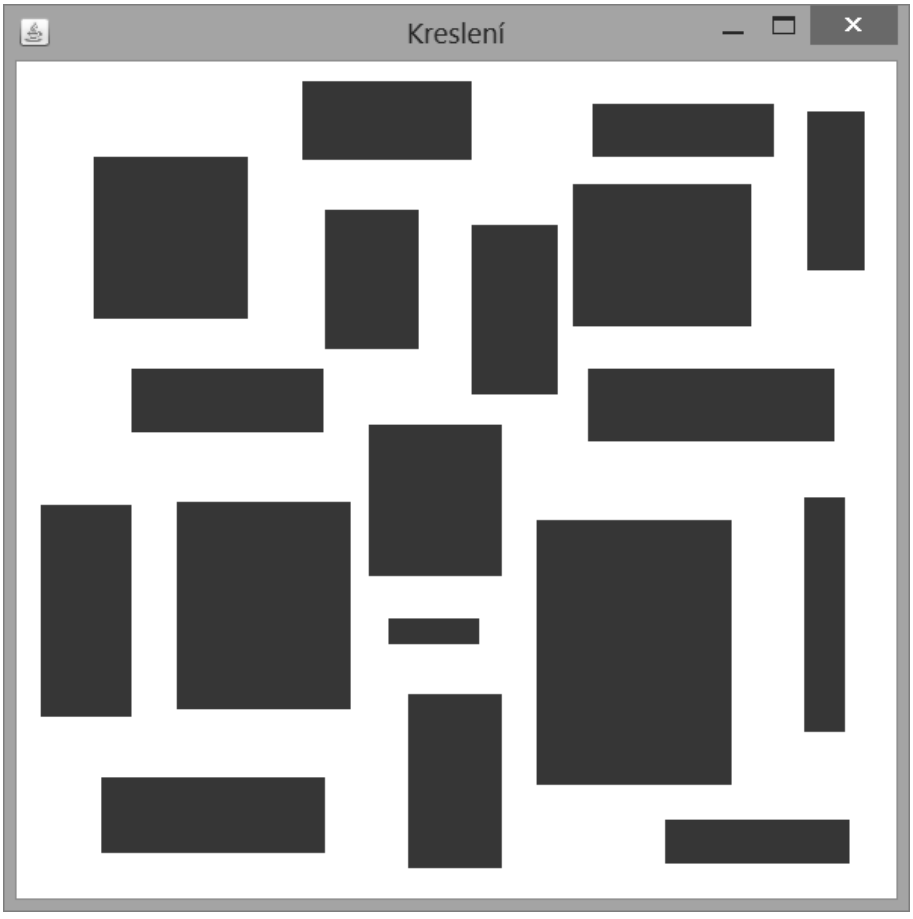
Obrázek 4.3 Vzhled seznamu s vlastním rendererem

Jak je vidět jen malým množstvím kódu, konkrétně příklady popsané výše se liší ani ne 50 řádky, je možné dosáhnout zajímavých výsledků. Komponenty jako `JButton`, `JPanel` a další neobsahují renderer. Pro úpravu těchto komponent je potřeba překrýt metodu `paintComponent(Graphics g)`.

4.2 Funkce

Při úpravách funkcí je velmi často využíváno objektově orientovaných znaků jazyka. Jako jsou dědičnost, překrytí metody, přetěžování atp. Následuje příklady pro vytvoření primitivního canvasu.

Jako základ je použita třída `JPanel` od kterého bude dědit třída `KresliciPlatno`. `JPanel` obsahuje metodu `paint(Graphics g)`, která standardně vykreslí šedý panel, do kterého je možné přidávat další komponenty. Cílem je do této metody vložit kód, který umožní místo šedého panelu vykreslovat obdélníky libovolné velikosti.



Obrázek 4.4 `JPanel` s přidanou funkcí kreslení

Při vytváření komponenty je vhodné začít u konstruktoru. Prvním příkazem musí být `super()`. To zajistí vykonání konstruktoru předka, v tomto případě `JPanelu`. Dále je nutné přidat posluchače myši, které třída `KresliciPlatno` implementuje. A tyto metody pak naplnit kódem. Nakonec přichází na řadu překrytí metody `paint(Graphics g)`. Výsledek může vypadat například takto:

```
public void paint(Graphics g){
      super.paintComponent(g);
      Graphics2D g2d = (Graphics2D) g;
      g2d.setColor(Color.RED);
      for (Rectangle R : this.objektyKVykresleni){
            g2d.fill(R);
      }
      g2d.fill(this.pom);
}
```

Kód 4.2 Překrytá metoda `paint(Graphics g)`

Kompletní kód lze opět najít v příloze.

Do obyčejného `JPanelu` byly přidány posluchače myši, které vytvářejí nové obdélníky a byla překryta metoda `paint(Graphics g)`. Tímto bylo dosaženo odlišné funkčnosti od původní implementace.

5 XML

Extensible Markup Language[10] je jednoduchý značkovací jazyk určený pro ukládání strukturovaných informací. Byl vyvinut konsorciem W3C s ohledem na jednoduchost a univerzálnost. Je odvozen od SGML (Standard Generalized Markup Language). XML je velmi rozšířeným formátem pro sdílení dat po internetu. Je to zejména díky tomu, že není potřeba žádný placený program pro jeho čtení nebo úpravu a je snadno strojově čitelný. Další pozitivní vlastností je fakt, že W3C uvolnilo specifikaci XML zdarma a kdokoliv tak může implementovat vlastní program pracující s XML. Pro svou rozšířenost se vývojáři programovacích jazyků jako jsou Java nebo C# rozhodli implementovat podporu pro čtení a zápis XML jako jednu ze standardních knihoven.

5.1 Formát

Jak již bylo zmíněno, XML je velmi snadno strojově čitelné, má totiž stromovou formu. Každý XML dokument má kořenový element. To je element uzavírající celý dokument. Všechny ostatní elementy i atributy jsou mu podřízeny. Dále může obsahovat libovolný počet elementů, které mají nějakou textovou hodnotu, nebo obsahuje další elementy. Každému elementu mohou náležet atributy. Každý atribut má svou textovou hodnotu.

```
<?xml version="1.0" encoding="UTF-8"?>
<Zpravy>
     <Zprava odesilatel="Lukas">
          <Text>Dlaždičkovač.</Text>
          <Prijemce>Jakub</Prijemce>
     </Zprava>
</Zpravy>
```

Kód 5.1 Ukázka XML formátu

Na výše uvedeném příkladu je vidět, jak by mohl vypadat záznam, nebo přenášená zpráva pomocí nějakého komunikačního programu. Jako první je uvedena

specifikace verze dokumentu a kódování. Následuje kořenový element `Zpravy`. `Zpravy` sdružují všechny ostatní elementy, vtomto případě elementy `Zprava`. `Zprava` má atribut odesilatel s hodnotou `Lukas`. Dále element `Text` a `Prijemce`, které mají vlastní obsah. Každý element je ukončen svou značkou.

5.2 Zpracování

Nyní, když je znám formát, je možné dokument přečíst (parsovat). Zde je potřeba zohlednit, jaká akce bude s následným dokumentem prováděna.

5.2.1 SAX

Simple API for XML. Základní a paměťově nenáročný model čtení. Soubor je zpracováván proudově. To znamená, že se čte od začátku do konce a již při čtení je potřeba zároveň soubor zpracovávat nebo si jeho části ukládat. Ukládat si tímto způsobem celé XML do paměti by bylo v rozporu se smyslem této techniky. SAX umožňuje zpracování velkých souborů ve velmi krátkém časovém úseku i na počítačích s omezenými zdroji.

5.2.2 DOM

Document Object Model. Druhý způsob čtení XML je oproti SAXu nesrovnatelně paměťově náročnější, hlavně při čtení velkých souborů. Takové soubory se pak na běžných domácích počítačích nedají tímto způsobem zpracovat. DOM vytváří stromovou reprezentaci obsahu XML souboru. Celý soubor je tedy nahrán do paměti, kde je přístupný k dalšímu zpracování. DOM je vhodný při zápisu, nebo úpravě struktury celého souboru.

6 Realizace

6.1 Analýza

6.1.1 Zadání

Cílem této práce je vytvoření multiplatformního, uživatelsky přívětivého programu pro pokládku dlažby. Program by měl umožňovat vytváření databáze dlaždic a její správu. Dále pak jednoduchý návrh místností, ve kterých se dlažba bude pokládat.

6.1.2 Vstupní data

Při prvním spuštění programu nebudou existovat žádná vstupní data. Postupem času uživatel vytvoří databázi dlaždic a místností, která se bude načítat při startu programu. Tato vstupní data by měla být dále přenosná.

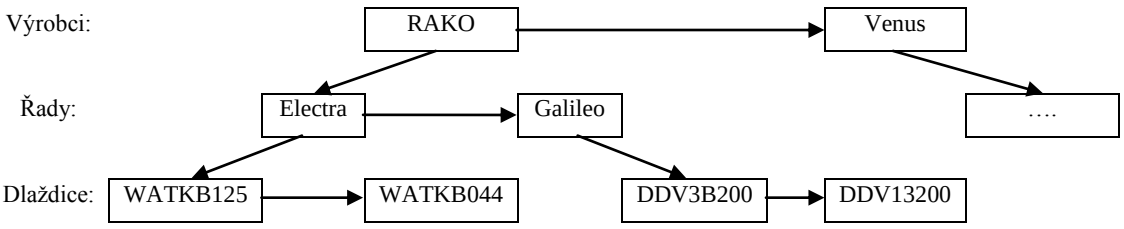
6.1.3 Výstupní data

Výstupem budou náklady na realizaci navrženého projektu, včetně množství dlažby, které je potřeba pořídit. Dále pak obrázek návrhu položení.

6.2 Datové struktury

6.2.1 Výrobce-Řada-Dlaždice

Jedním z nejdůležitějších faktorů ovlivňujících rychlost programu je uspořádání dat. Základní strukturou programu je struktura Výrobce-Řada-Dlaždice. Její grafická reprezentace je vidět to obrázku 6.1.



Obrázek 6.1 Struktura Výrobce-Řada-Dlaždice

Struktura začíná spojovým seznamem výrobců. Každý výrobce má seznam řad a v každé řadě je seznam dlaždic. Tato struktura nabízí velmi rychlé vyhledávání, pokud je známé jméno výrobce a řady. V nejhorším případě bude složitost vyhledávání O(n) pokud je známé jen jméno dlaždice. Také je vhodná pro jednoduchou filtraci podle výrobce a řady. Nyní by bylo vhodné se zaměřit na jednotlivé objekty.

6.2.2 Dlaždice

Obsahuje všechny nutné informace pro vytvoření nové dlaždice jako je jméno, rozměry, cesta k souboru s texturou… Celkem má 18 atributů a nemá smysl je zde všechny vyjmenovávat. I přes to, že obsahuje relativně velké množství parametrů, nejsou to zdaleka všechny, kterých bude v celém programu zapotřebí. Dlaždice mají celkem 3 třídy, které se o jejich kompletní reprezentaci postarají. Je to zejména z úsporných důvodů, které budou popsány dále.

6.2.3 Použitá dlaždice

Jednou z dalších reprezentací je třída použitá dlaždice. Tento objekt se vytvoří pouze jednou a to v případě, že dlaždice byla přidána do aktuálního projektu. Použitá dlaždice obsahuje odkaz na vyšší reprezentaci, dále pak texturu dlaždice ve formátu BufferedImage a index v poli, na kterém je uložena. Jak je vidět zahrnuje jen minimum dalších informací. Klíčová je především textura, která tak není načítána zbytečně, ale jen v případě potřeby.

6.2.4 Dlaždička

Poslední možná reprezentace dlaždice. Vytvoří se, pouze pokud uživatel umístí dlaždici na pracovní plochu programu. Skládá se z informací nutných pro vykreslení, jako je pozice, úhel natočení… Jak bylo uvedeno výše, neobsahuje texturu. To by značně zvýšilo paměťové nároky aplikace. V tomto uspořádání je možné mít na pracovní ploše řádově desítky tisíc dlaždic a stále být schopen plynule používat program.

6.3 Canvas

Pravděpodobně nejdůležitější třída celé aplikace. Stará se o vykreslování grafické reprezentace a zajišťuje interakci s uživatelem.

6.3.1 Interakce

Protože po canvasu je třeba se pohybovat, byly implementovány posluchače myši včetně jejího pohybu, stisknutých tlačítek i kolečka. Po zpracování pohybů a kliknutí jsou prováděny příslušné afinní transformace.

Posun

Pro prohlížení aktuálního výsledku pokládky je nutné se po plátně pohybovat. Nejsnazší řešení je využití metody `translate(double x, double y)`. K zajištění správného pohybu je potřeba implementovat akci `mouseDragged (MouseEvent e)`. Uvnitř této metody se musí vypočítat absolutní posun proti bodu [0, 0]. Tomuto posunu je následně obráceno znamínko a pak je předán výše zmíněné metodě.

Přibližování

Jen o něco málo složitější operace. Pokud by byla pouze použita metoda `scale(double x, double y)`, neprobíhalo by přiblížení tak, jak očekává běžný uživatel. Obraz by se snažil přibližovat směrem k bodu [0, 0] místo ke středu obrazovky. Jestliže je požadováno přiblížení k aktuálnímu středu pohledu, musí se takový příkaz skládat z více kroků.

1. Nalezení souřadnic středu obrazovky
2. Posunutí středu do bodu [0, 0]
3. Použití funkce scale
4. Posun zpět na původní souřadnice

Tomuto postupu se říká řetězení transformací. Důležité je převést střed zvětšení do pozice [0, 0], protože jedině pak se budou násobit správná čísla zvětšení se správným posunem. Nakonec je plátno vráceno na původní souřadnice.

6.3.2 Vykreslovací smyčka

Pokud je potřeba vykreslovat neznámé množství různých prvků, které mohou být přidávány a odebírány, bude patrně nejlepším řešením použít nějakou dynamickou strukturu. Jako řešení byla zvolena struktura ArrayList a to především z důvodu složitosti náhodného přístupu, který je O(1). ArrayList je vlastně dynamicky alokované pole. V nejhorším případě, kdy se pole musí zvětšit nebo zmenšit je složitost výběru a přidání O(n). Průměrný případ je ale stejně jako u pole O(1). Z těchto důvodů byl ArrayList zvolen, jako úložiště vybraných objektů.

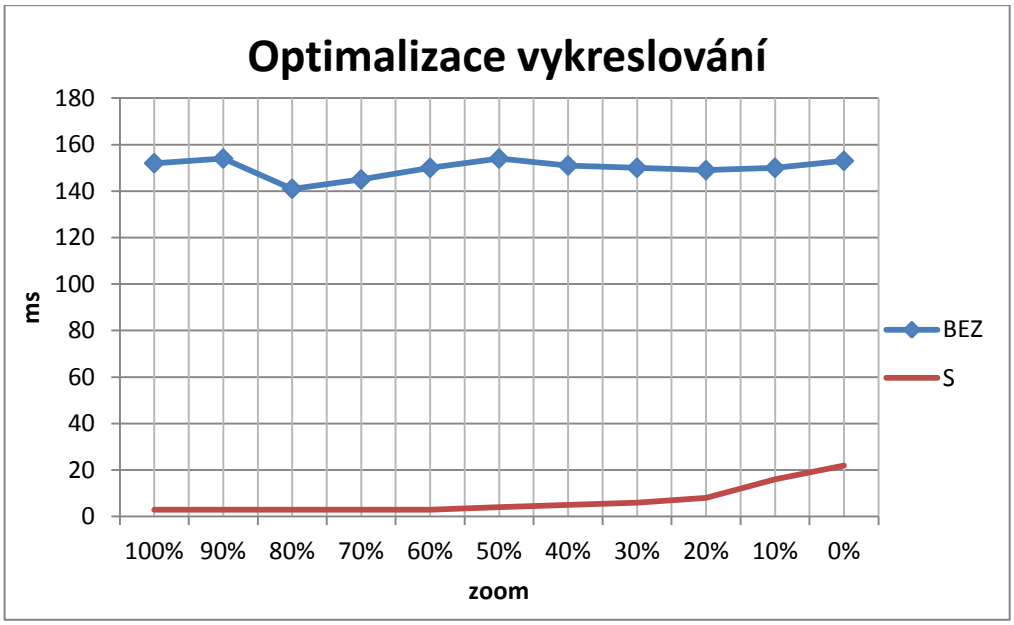
Protože je nutné zachovat stejnou práci se všemi objekty v poli, ať už je to místnost, dlaždice či odřezek z dlaždice, musí všechny tyto objekty implementovat rozhraní `IVykreslitelne`. Každý IVykreslitelný objekt má metodu vykresli. Tato metoda zajistí vkreslení všech typů objektů.

Jak bylo popsáno v kapitole vykreslování (kap. 3). Je využito softwarové vykreslování se všemi jeho výhodami i zápory. Pro plynulejší práci je tedy nezbytně nutné nastoupit cestu optimalizace vykreslování.

6.3.3 Optimalizace

Jak již bylo zmíněno v kapitole clipping (3.2) nejběžnější optimalizací je oříznutí vykreslované oblasti na oblast viditelnou. Třída `Canvas`, která se stará o veškeré vykreslování obsahuje základní ořezávací algoritmus. Jedná se o obdobu Cohen-Shutterlandova algoritmu. Algoritmus pomocí jednoduchého srovnání souřadnic zjistí, jestli je daný objekt na obrazovce. Pokud není, objekt se nevykreslí.

Nyní by se nabízelo srovnání výkonu s optimalizací a bez této optimalizace. Testování probíhalo na sestavě: Intel Core i7 2,2GHz, 16GB RAM, FullHD display. V rámci srovnání bylo na canvas umístěno 250 x 250 dlaždic.



Graf 6.1 Porovnání časů vykreslení scény bez optimalizace a s optimalizací

Ani při maximálním oddálení není všech 250^2 dlaždic nutno kreslit. Na monitoru s FullHD rozlišením bylo v jednu chvíli maximálně 109 x 64 dlaždic. Touto technikou lze podle testů urychlit program až stonásobně. Urychlení závisí na celkovém počtu objektů ve scéně a počtu zobrazených objektů.

6.4 Detekce kolizí

Implementovat detekci kolizí není většinou jednoduchý úkol. Je nutné, aby byla co nejrychlejší a nezpomalovala program více než je nezbytně nutné. Toto je však v případě detekce kolize polygonů netriviální úloha.

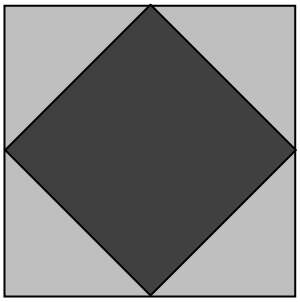
6.4.1 Kolize polygonů

Pro řešení samotných kolizí byl zvolen vcelku přímočarý postup. Pokud polygony kolidují, musí se jejich hranice protínat. Každý polygon skládající se z N bodů, lze rozložit na stejný počet úseček. Pokud se protíná jakákoliv dvojice úseček, pak polygony kolidují.

6.4.2 Optimalizace

Je očividné, že výše uvedená operace má vysokou složitost. Konkrétně je to O(m*n), kde m a n jsou počty bodů polygonů. Provádění takového algoritmu pro řádově tisíce nebo desetitisíce prvků je v rozumném čase prakticky vyloučeno. Jediným východiskem bylo detekci zjednodušit.

Uživatel může pohybovat maximálně jednou dlaždicí v určitém okamžiku. A tedy se může „srazit“ jen s velmi omezeným počtem dlaždic. Ostatní kolize by se počítali naprázdno. Nejprve tedy bude nutné, vypočítat, jen zhruba, zda se dlaždice mohou protínat co nejjednodušším způsobem. K tomuto účelu výborně poslouží obdélníky. Kolize dvou obdélníku zahrnuje pouze porovnání souřadnic a velikostí. Je tím pádem mnohem rychlejší. Každá dlaždice je obklopena nejmenším možným obdélníkem.



Obrázek 6.2 Ohraničení dlaždice/odřezku nejmenším obdélníkem

Teprve po protnutí ohraničujících obdélníků se spustí výpočet průniků odpovídající dvojice polygonů. Pro testovací příklad 250^2 dlaždic a stejnou sestavu jako v kapitole 6.3.3 jsou výsledky následující.

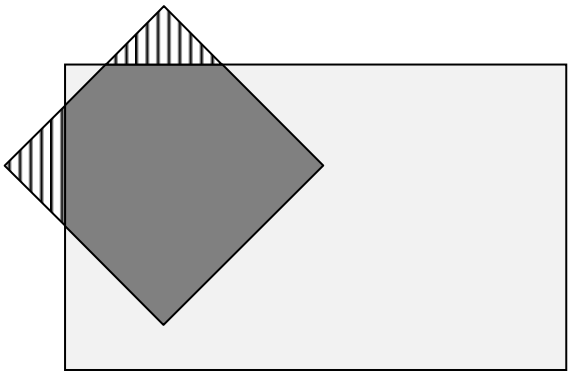
Bez optimalizace	26 – 29 ms
S optimalizací	5 – 6 ms

Tabulka 6.1 Porovnání časů vyhodnocení kolizí bez optimalizace a s optimalizací

Jak je vidět z výsledků měření, detekce kolizí je s optimalizací přibližně 5x rychlejší. Časová úspora 20 – 25ms v každém snímku je nezanedbatelná. Spolu s optimalizací kreslicího plátna tak umožňuje vykreslování o rychlostech vyšších než 60FPS i při takto vysokém počtu dlaždic.

6.5 Ořezávaní dlaždic

Ořezávání dlaždic je pravděpodobně implementačně nejsložitější částí celého programu. Pro ořezávání je využit Weiler-Arthertonův algoritmus (viz kapitola 3.2.2). Pokud dlaždice přesahuje hranice místnosti, může být oříznuta tak, aby se do místnosti celá vešla. Důvod, proč byl zvolen výše zmíněný algoritmus je prostý. Při vykonávání algoritmu není získán jen oříznutý polygon, ale zároveň jsou seznamy bodů polygonů připravené pro získání všech odpovídajících odřezků.



Obrázek 6.3 Ukázka rozdělení dlaždice po ořezávání

Odřezky nejsou vymazány, jsou uchovány v programu pro další použití. Takto je možné zpracovat každý kousek dlažby. Samotné odřezky jde dále ořezávat.

6.6 Reprezentace XML

Pro vytvoření použitelného návrhu pokládky je nutné zachovávat data. Není možné při každém spuštění programu znovu vytvořit všechny dlažice a místnosti. Tyto informace musí být v nějaké formě uchovány pro opětovné využití.

Jako formát pro ukládání souboru bylo zvoleno XML. Jedním důvodem byla jednoduchá tvorba a čtení těchto souborů v Jave. A druhým snadná přenositelnost mezi více stroji. Lze tak vzít databázi vytvořenou na jednom počítači a bez jakýchkoliv problémů ji použít na jiném. Každý objekt má, vlastní XML strukturu, jež bude dále rozebrána.

6.6.1 Dlaždice

Základní objekt celého projektu. Obsahuje velké množství atributů, které mohou být přiřazeny. Struktura souboru je následující.

```
<dlazdice>
     <vyrobce>rako</vyrobce>
     <rada>galileo</rada>
     <jmeno>ddv3b200</jmeno>
     <textura>databaze\\obrazky\\RAKO\\GALILEO\\DDV3B200.jp
g</textura>
     <cena>299</cena>
     <delka>330</delka>
     <sirka>330</sirka>
     <vyska>8</vyska>
     <povrch>reliéfní</povrch>
     <protiskluznost>R10/B</protiskluznost>
     <mrazuvzdornost>true</mrazuvzdornost>
     <probarvenyStrep>true</probarvenyStrep>
     <odolnost>PEI 5</odolnost>
     <karton>4</karton>
     <naMetr>9</naMetr>
</dlazdice>
```

Kód 6.1 Ukázka dlaždice uložené v XML souboru

Toto jsou všechny údaje, které je možné při vytváření dlaždice vyplnit. Některé program přímo nepoužívá a slouží jen jako informace pro uživatele při výběru a konečné rekapitulaci, například probarvený střep, mrazuvzdornost, protiskluznost a další. Tyto údaje mohou být například využity při rozšíření programu o pokročilé filtrování nebo jiné funkce. Prvních sedm údajů z ukázky je však povinných a bez nich nebude dlaždice vůbec načtena.

6.6.2 Místnost

Místnost má při ukládání jednodušší strukturu než dlaždice. Skládá se z x-ových a y-ových souřadnicí bodů.

```
<Mistnost>
     <Bod>
          <x>71</x>
          <y>85</y>
     </Bod>
     //další body
</Mistnost>
```

Kód 6.2 Ukázka místnosti uložené v XML souboru

Pro úsporu místa je zde ukázka místnosti složené z jednoho bodu. Taková místnost by v programu nemohla být načtena, ale tady jde jen o strukturu souboru. Validní místnost se skládá ze 4 a více bodů. Každý bod je reprezentován elementem `Bod` a má hodnoty v elementech `x` a `y`.

Takováto struktura by mohla být snadno obsažena i v obyčejném textovém souboru. V případě ukládání celého projektu do jednoho XML souboru, by však tento přístup selhal.

6.6.3 Odřezek

Výše zmíněné struktury je možné ukládat jednotlivě. Tato struktura se ukládá pouze jako část celého projektu. Již nemá předem známou strukturu. Jeho hranice tvoří polygon. A je zde několik dalších proměnných nutných k uložení tohoto objektu.

```
< Odrezek >
     <PouzitaDlazdice>0</PouzitaDlazdice>
     <IndexTextury>0</IndexTextury>
     <Uhel>0.0</Uhel>
     <Spara>50</Spara>
     <Obdelnik>
```

```
            <x>142</x>
            <y>34</y>
            <Sirka>330</Sirka>
            <Vyska>330</Vyska>
      </Obdelnik>
      <VelikostR>
            //body polygonu bez spary
      </VelikostR>
      <VelikostS>
            //body polygonu se sparou
      </VelikostS>
</Odrezek>
```

Kód 6.3 Ukázka odřezku uloženého v XML souboru

6.6.4 Projekt

Spojuje všechny výše zmíněné struktury do jednoho souboru. Zjednodušeně jeho struktura je následující.

```
<Projekt>
      <Mistnost>
            //struktura mistnost
      </Mistnost>
      <Dlazdicky>
            <Dlazdice>
                  //data jednotlivých použitých dlaždic
            </Dlazdice>
      </Dlazdicky>
      <Odrezky>
            <Odrezek>
                  //struktura odrezek
```

```
            </Odrezek>
        </Odrezky>
</Projekt>
```

Kód 6.4 Ukázka projektu uloženého v XML souboru

Takto uložený projekt může být opětovně načten. Podmínkou je přítomnost využitých dlaždic v cílovém systému. Pokud se dlaždice budou lišit, nebude výsledek korektní.

6.7 Export výsledku

Pro export výsledného projektu byl vybrán formát HTML. A to z jednoho prostého důvodu. Téměř každý počítač umí tento formát přečíst. Lze také dále upravovat pomocí externích CSS stylů. Přímo do HTML souboru lze také vložit obrázek s návrhem pokládky a uživatel tak není nucen starat se o více souborů.

Pokud uživatel požaduje pouze obrazový výstup, je možné exportovat návrh jako SVG a PNG obrázky. Pro SVG export byla využita knihovna Batik.

7 Testování

7.1 Zadání

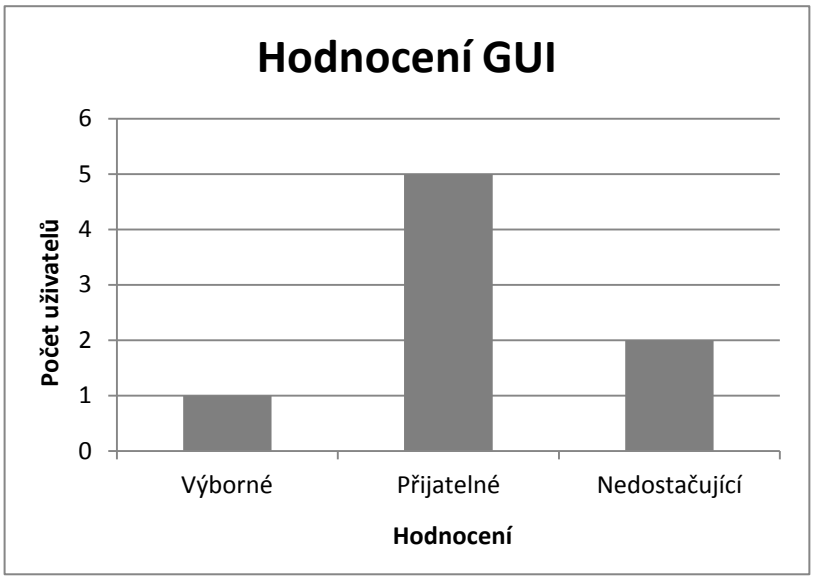
Hotový program byl předložen skupince osmi lidí, kteří měli následující úkol:

- Nakreslete libovolnou místnost
- Tuto místnost uložte
- Načtete místnost
- Vyberte dlaždici pro dláždění
- Nechte program vydláždit místnost (styl zvolte dle libosti)
- Výsledek uložte

Cílem bylo otestovat uživatelskou přívětivost programu.

7.2 Výsledky

Všichni, až na dva uživatele, dokázali úkoly splnit. Jejich hodnocení přehlednosti uživatelského rozhraní byla následující.



Graf 7.1 Uživatelské hodnocení GUI

Dva uživatelé (25%) ohodnotili rozhranní jako nedostačující. Jejich námitky byly zaměřeny hlavně proti systému přidávání dlaždic do projektu. Tito uživatelé by byli

nejraději, kdyby byly už při spuštění všechny dlaždice přidány do projektu, aby to nemuseli dělat oni. Problém je v tom, že pokud databáze dlaždic bude obsahovat několik desítek a více dlaždic, stane se tento systém ještě složitějším než původní přístup. Na základě těchto připomínek nebyl program nijak upravován. Velká většina dobrovolníků (62,5%) označila GUI jako přijatelné. Neměli vetší problémy se zvládnutím zadané úlohy. Jeden člověk (12,5%) ohodnotil program jako výborný.

8 Přehled tříd

Celý program je rozdělen do sedmi balíků.

- Akce – obsahuje akce pro tlačítka, pouze volají metody jiných tříd, nebudou zmíněny
- Canvas – stará se o vykreslování a zobrazitelnou část
- Grafika – grafické operace, operace s obrázky
- GUI – vytváří uživatelské rozhraní, pouze zobrazuje GUI, nebude zmíněno
- IO – vstupní a výstupní operace
- Program – obsahuje main, obsahuje globální proměnné a práci s daty
- Struktury – sdružuje všechny struktury

Ne vždy jsou zmíněny všechny třídy. Některé mají například na starosti jen filtrování souborů nebo jsou jejich funkce velmi podobné jiným třídám. Takovéto soubory jsou vynechány.

8.1 Package `Canvas`

Obsahuje tři třídy, sloužící pro vykreslování dlaždic a místností, popřípadě pro kreslení místností nových.

8.1.1 Třída `Canvas`

Obecné kreslící plátno. Implementuje základní funkce pohybu po kreslícím plátně. Jako je posun nebo zvětšení. Obsahuje také metodu pro vykreslení požadovaných objektů. Tyto objekty musí být typu `IVykreslitelne`.

`protected void` `paintComponent(Graphics g)` – vykreslovací smyčka, vykresluje `IVykreslitelne` objekty v ArrayListu `objektyKVykresleni`.

private boolean `jeVidet(IVykreslitelne v)` – Určuje, zda daný objekt je viditelný. Používá maximálně čtyři porovnání pro určení viditelnosti. Návratová hodnota je `true`, pokud je objekt vidět jinak `false`. Viz 3.2.1 a 6.3.3.

public void `translace(MouseEvent e)` – získá z objektu `e` a předchozí pozice relativní souřadnice posunu. Následně použije třídu `AffineTransform`, konkrétně metodu `translate(x, y)` a posune `Canvas` o příslušnou vzdálenost.

public void `mouseWheelMoved(MouseWheelEvent e)` – zaznamená pohyb kolečka myši. Opět využívá třídu `AffineTransform`. Tentokrát však pro zvětšení/zmenšení používá metodou `sacle(sx, sy)`. `Canvas` je při každém zvětšení přiblížen o 10%. Analogicky pro oddálení. Zvětšování probíhá vždy do středu obrazovky. Transformace se řetězí ve tvaru T(stred), S(zvetseni), T(-stred).

protected Point `prevedObrazoveNaCanvasSouradnice(int x, int y)` – Převádí obrazové souřadnice na souřadnice Canvasu. Při pohybování a zvětšování Canvasu je potřeba neustále udržovat souřadný systém. Výpočet je prováděn pomocí hodnot výše zmíněných transformací translate a scale. Příklad pro osu x

```
x = (xo - trans.getTranslateX()) / trans.getScaleX();
```

Obdobným způsobem probíhá pro výpočet pro osu y.

8.1.2 Třída `HlavniCanvas`

Potomek třídy Canvas. Rozšiřuje většinu jeho metod. V programu se stará o funkce hlavního plátna, na kterém uživatel pohybuje dlaždicemi, ořezává je apod.

public void `mousePressed(MouseEvent e)` – Registruje držení stisknutého tlačítky myši. Je zde volána metoda pro výběr objektu, nad kterým byla myš stisknuta. Popřípadě při stisku pravého tlačítka se pokusí vyvolat `JPopupMenu` pro příslušný objekt.

private void `zpracujPopupMenu(MouseEvent e)` – Ověří, zda je aktuálně vybraný objekt implementuje rozhraní `INaplnPopupMenu`. Pokud ano, zavolá příslušnou metodu v daném objektu a vykreslí na místě specifikovaném objektem `e` vyskakovací menu.

private void `nastavVybranyObjekt(MouseEvent e)` – Projde celý seznam `objektyKVykresleni` za účelem zjistit, zda není některý právě vybrán. Pokud je takový objekt nalezen, je také nastavena souřadnice úchytu v rámci objektu. Ta je důležitá pro plynulý pohyb objektu.

public void `mouseClicked(MouseEvent e)` – Pokud je připravena nějaká dlaždice ke vkládání v proměnné `dlazdiceProVkladani`, pak je ověřeno, zda se dlaždice vejde na aktuální pozici. V kladném případě je dlaždice umístěna a přidána do seznamu `objektyKVykresleni`.

8.1.3 Třída `NovaMistnostCanvas`

Potomek třídy `Canvas`. Slouží pro návrh půdorysu místností. Velice podobná třídě `HlavniCanvas`. Rozdílem je, že `NovaMistnostCanvas` nemá tolik problémů k řešení. Jediné co umí je vykreslit kolmou čáru a zajistit aby se čáry neprotínaly. Už z toho důvodu, že mají s třídou `HlavniCanvas` stejného předka jsou implementované metody téměř totožné a nemá tedy příliš smysl je zde znovu vypisovat.

8.2 Package `Grafika`

Obsahuje grafické algoritmy v práci použité. Jako je například změna velikosti obrázků, jejich rotace nebo ořezávání polygonů a další.

8.2.1 Třída `Orezavani`

Zahrnuje všechny metody nutné pro ořezávání dvou polygonů.

`public static void` `orizni(IOriznutelne objekt, ArrayList<IVykreslitelne> objektyKVykresleni, Mistnost mistnost)` – Volá samotnou metodu řezání objektů. Po oříznutí jsou data vrácena zpět do této metody. Dále je zde určeno, jestli je odřezek uvnitř místnosti a má se nadále vykreslovat, nebo jestli je mimo a umístí se do panelu odřezků.

`public static` `ArrayList <ArrayList <Point>> orizniObjekt (Mistnost mis, IOriznutelne pom,` **`boolean`** `vcetneSpary)` – Metoda pro oříznutí dvou polygonů. Prvním polygonem je místnost a druhým objekt `IOriznutelne`. Logická hodnota `vcetneSpary` určuje, jestli se jako hranice `IOrizmutelneho` objektu budou brát jeho skutečné hranice, nebo se hranice posune na okraj spáry. Algoritmus použitý pro ořez je blíže popsán v kapitole 3.2.2. Návratovou hodnotou metody je `ArrayList ArrayListů`, kde každý seznam vyjadřuje jeden polygon.

`private static` `ArrayList<Point> vytvorOdrezek(`**`int`** `index, ArrayList<OrezavaniBod> mistnost, ArrayList<OrezavaniBod> objekt)` – Je volán po metodě `orizniObjekt()`. K původnímu průniku obou polygonů přidá i odřezky. Proměnná index určuje aktuální vstupní bod. Definici vstupního bodu lze nalézt v odstavci Vstupní bod v kapitole 3.2.2.

8.2.2 Třída `Transformace`

Obsahuje metody pro práci s obrázky. Umožňuje obrázkům měnit velikost, otáčet je nebo z odřezku vytvořit ikonu.

public static BufferedImage prizpusobObrazek(BufferedImage obrazek, **int** vyska, **int** sirka, **int** kvalita) – Zmenšuje obrázek na požadovanou velikost definovanou proměnnými vyska a sirka. Kvalita je číslo od jedné do čtyř, kde jednička reprezentuje nejnižší kvalitu a nejrychlejší vykreslení. Čtyřka je přesný opak. Kvalita vykreslování je ovlivněna změnou interpolace. 1 – nejbližší soused, 2 – bilineární, 3 – bikubická. Pouze 4 je tvořena metodou getScaledInstance(), která poskytuje nejlepší výsledky. Vysoká kvalita je v programu používána zejména pro ikony dlaždic, které zůstanou uchovány. Tato hodnota nelze v programu uživatelsky změnit. Návratová hodnota je zmenšený obrázek.

public static ImageIcon otocIkonu(ImageIcon ikona, **double** uhel) – Metoda otočí ikonu o zadaný uhel ve směru hodinových ručiček. Úhel musí být zadán v radiánech.

8.3 Package IO

Spravuje veškeré souborové vstupní a výstupní operace. Načítá a ukládá všechny XML struktury, otevírá a zapisuje konfiguraci programu. Na starosti má také načítání obrázků v takové formě, aby práce s nimi byla co nejrychlejší.

8.3.1 Třídy DlazdiceXML, MistnostXML a ProjektXML

Nemá cenu hovořit o každé z nich zvlášť. Všechny obsahují stejné metody. Jsou to metody nacti(String cesta) a zapis(Dlazdice/Mistnost/Projekt obj, String cesta).

8.3.2 Třída NactiSoubory

Při spuštění programu je nutné nalézt všechny XML soubory definující dlaždice a tyto soubory načíst. To má na starosti tato třída.

public static void `nacti()` – Jediná public metoda v souboru. Volá metodu pro prohledání všech XML souborů ve složce databaze/XML. Následně další metoda tyto soubory načte a uloží.

private static `File[] nactiSoubory()` – Prohledává složku databaze/XML a hledá všechny XML soubory. Pro tento účel byl implementován `FileFilter`. Návratovou hodnotou je pole typu File.

private static void `zpracujSoubory(File[] poleSouboru)` – Zpracuje ve for cyklu každý soubor. Dlaždice jsou načítány statickou metodou `DlazdiceXML.nacti(cesta)`. Po načtení je Dlaždice umístěna do struktury Výrobce-Řada-Dlaždice (viz 6.2.1).

8.3.3 Třída `ObrazkyIO`

Stará se o načítání obrázků. Obrázky jsou načítané v takovém formátu, aby bylo jejich vykreslování pokud možno co nejrychlejší.

public static `BufferedImage nactiObrazek(String cesta)` – Načítá BuferedImage ze zadané cesty. Obrázek musí být načten v kompatibilním formátu, aby bylo jeho následné vykreslování co možná nejrychlejší. Toho je dosaženo následujícím kódem.

```
GraphicsConfiguration gfx_config = GraphicsEnvironment.
getLocalGraphicsEnvironment().getDefaultScreenDevice().getDefaultConfi
guration();

BufferedImage new_image = gfx_config.createCompatibleImage (sirka,
vyska, Transparency.OPAQUE);
```

Kód 8.1 Vytvoření kompatibilního obrázku může několikanásobně zrychlit vykreslování

Tato část kódu umožní načtení obrázku ve formátu, který je vhodný pro aktuální grafické prostředí. Třída `GraphicsConfiguration` zjistí vhodné nastavení pro

aktuální grafickou konfiguraci. A metoda `createCompatibleImage()` třídy `BufferedImage` vytvoří obrázek s vhodnou reprezentací formátu a barevné reprezentace obrázku.

`public static` `ImageIcon vratIkonu()` – Umožňuje uživateli vybrat obrázek pomocí třídy `JFileChooser`. Při zavolání této metody se otevře okno procházení souborů. Pokud uživatel vybere .jpg nebo .png obrázek menší než 5MB, načte se tento soubor jako ikona.

`public static` `String ulozObrazekAIkonu(String vyrobce, String rada, String dlazdice, String cesta)` – Ukládá obrázek a ikonu nové dlaždice do souboru. Výsledný soubor bude ve složce cesta a bude pojmenován `vyrobce_rada_dalazdice.png || .jpg` v případě obrázku a `vyrobce_rada_dlazdice_thumb.png || .jpg` v případě ikony. Návratovou hodnotou je cesta, kam byl obrázek uložen.

8.3.4 Třída `Validator`

Ověřuje zda jsou místnosti validní a pro program použitelné.

`public boolean` `platnaMistnost(ArrayList<Point> seznamBodu)` – Ověřuje, zda místnost reprezentovaná seznamem bodů je platná. Platná místnost nemá žádné redundantní body, je ve směru hodinových ručiček a všechny stěny jsou kolmé. Na všechny tyto vlastnosti u místnosti navazující kód spoléhá. Zejména ořezávání dlažic vyžaduje polygon uspořádaný ve směru hodinových ručiček bez redundantních bodů, kdy je pak jednoznačně určené pořadí bodů pro ořez. Orientace je vypočítána pomocí vektorového součinu.

`public` `ArrayList<Point> opravMistnost(ArrayList<Point> seznamBodu)` - Pokusí se opravit místnost. Zvládne opravit orientaci bodů a

odstranit redundantní body. Návratová hodnota je **`null`** pokud místnost nelze opravit. Jinak je hodnota opravená místnost.

8.4 Package `Program`

Obsahuje hlavní metodu `main()`. Dále zahrnuje třídy, které poskytují přístup ke globálním datům a třídu pro práci s těmito daty.

8.4.1 Třída `Program`

Osahuje pouze metodu `main()`.

`public static void` `main(String[] args)` - V této metodě je nejprve voláno načtení dlaždic do struktury. Následně je načten konfigurační soubor s nastavením. Nakonec je vytvořeno samotné GUI programu.

8.4.2 Třída `GlobalniPromenne`

Zahrnuje všechny důležité seznamy a pole struktur. Všechny modely pro vytvoření seznamů atd. Poskytuje k nim přístup pomocí getrů a setrů.

8.4.3 Třída `Data`

Umožňuje pracovat s daty uloženými ve třídě `GlobalniPromenne`. Poskytuje přístup, jak k celým strukturám, tak k jejich částem.

8.5 Package `Struktury`

Obsahuje všechny struktury pro reprezentaci dlaždic včetně odřezků. Zahrnuje také rozhraní, která tyto struktury implementují.

8.5.1 Rozhraní `IVykreslitelne`

Základní rozhraní, které musí implementovat každý vykreslitelný prvek. Obsahuje zejména metodu vykresli a metody vracející přesnou polohu objektu pro účely clippingu.

8.5.2 Rozhraní `IOriznutelne`

Každý objekt, který lze oříznout musí implementovat toto rozhraní. Obsahuje zejména metody pro zjištění průniků a návrat bodů, ze kterých se objekt skládá.

8.5.3 Rozhrani `INaplnPopupMenu`

Obsahuje jen jedinou metodu, která je nutná pro naplnění vyskakovacího menu správnými hodnotami. Každý objekt implementující toto rozhranní přidá své akce do menu.

9 Závěr

Úkolem bylo vytvořit program pro tvorbu návrhu pokládky dlažby. Včetně výpočtu nákladů na realizaci této pokládky.

V první části bylo nutné nastudovat některé algoritmy počítačové grafiky, které by pomohly vytvořit nenáročnou aplikaci. Dále pak algoritmy ořezávání polygonů pro potřeby ořezávání jednotlivých dlaždic. Kvůli požadavku, dlaždice permanentně uchovávat, bylo v této fázi také vybráno XML jako vhodný způsob reprezentace dat.

V realizační části dokonce byly implementovány některé funkce nad rámec původního zadání, jako například automatické dláždění. Při implementaci byl kladen důraz na optimalizaci vykreslování a detekci kolizí. Výsledný program byl otestován na malé skupině dobrovolníků s cílem ověřit přehlednost uživatelského rozhranní. Výsledky naznačují, že rozhraní je vcelku použitelné.

Z výše uvedených skutečností lze konstatovat, že práce splňuje a v některých bodech i přesahuje původní zadání.

Zdroje

[1] Java Timeline. ORACLE. *Java Timeline* [online]. [cit. 2013-05-05]. Dostupné z: http://oracle.com.edgesuite.net/timeline/java/

[2] The Art of Unix Usability. *A Brief History of User Interfaces* [online]. [cit. 2013-04-01]. Dostupné z: http://www.catb.org/esr/writings/taouu/html/ch02s01.html

[3] ROHLÍK, Ondřej. *Přednášky UUR*. Plzeň, 2010. Západočeská univerzita.

[4] HAASE, Chet a Romain GUY. *Filthy rich clients: developing animated and graphical effects for desktop Java applications*. Upper Saddle River: Prentice Hall, 2008, xxvii, 572 s. the Java series. ISBN 978-0-13-241393-0.

[5] JavaFX Developer Home. ORACLE. *Oracle* [online]. [cit. 2013-05-08]. Dostupné z: http://www.oracle.com/technetwork/java/javafx/overview/index.html

[6] MOUČEK, Roman. *Přednášky ZSWI*. Plzeň, 2011. Západočeská univerzita.

[7] SKALA, Václav. ZÁPADOČESKÁ UNIVERZITA. *Algoritmy počítačové grafiky I: (Algorithms for computer graphics I)* [online]. Plzen: Union Agency, 2011, 114, [15] s. [cit. 2013-05-05]. ISBN 978-80-86943-19-0.

[8] SKALA, Václav. ZÁPADOČESKÁ UNIVERZITA. *Algoritmy počítačové grafiky II: (Algorithms for computer graphics II)* [online]. Plzen: Union Agency, 2011, 165, [15] s. [cit. 2013-05-05]. ISBN 978-80-86943-20-6.

[9] ZPG: cvičení č. 4. [online]. [cit. 2013-05-08]. Dostupné z: http://herakles.zcu.cz/education/zpg/cviceni.php?no=4

[10] XML ESSENTIALS. W3C. [online]. [cit. 2013-05-05]. Dostupné z: http://www.w3.org/standards/xml/core

Příloha A

Seznamy

Následující kód vytváří nový `JList`, který naplní statickými daty. Položky nejdou přidávat ani mazat. Jde o nejjednodušší možnou implementaci seznamu.

Program.java

```
import javax.swing.DefaultListModel;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;

public class Program extends JFrame{


	private static final long serialVersionUID = 1L;


	/**
	 * Vytvoreni datoveho modelu pro vysledny seznam. Do tohoto
	 * modelu se budou vkladat prvky seznamu. Vyhodou je to, ze
	 * pokud se model zmeni seznam se aktualizuje automaticky.
	 */
	private static DefaultListModel<Fotografie> fotografieModel =
new DefaultListModel<Fotografie>();



	/**
	 * Hlavni metoda. Nejdrive zavola naplneni datoveho modelu.
	 * Pak vytvori nove okno Program.
	 */
	public static void main(String[] args){
		pripravData();
		new Program();
	}


	/**
	 * Vklada predem pripravena data do modelu seznamu. Cely
	 * obsah metody je uzavren konstrukci try. Pokud nebudou
	 * sobory nalezeny, nebo se vyskatne jina chba aplikace
	 * upozorni uzivatele.
	 */
	private static void pripravData() {
		try{
			fotografieModel.addElement(new Fotografie("Oblíbené",
new ImageIcon("img\\favorites.png")));
			fotografieModel.addElement(new Fotografie("Důležité",
new ImageIcon("img\\important.png")));
			fotografieModel.addElement(new Fotografie("Nápověda",
new ImageIcon("img\\help.png")));
		} catch (Exception e){
```

```
                System.out.println("Obrazky nebyly nalezeny.");
            }
        }


        /**
         * Konstruktor tridy Program. Vytvari nove okno, vklada obsah
         * a nastavuje vsechny nezbytne parametry.
         */
        public Program(){
            this.getContentPane().add(new JScrollPane(pridejList()));
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setTitle("Seznam fotografií");
            this.setSize(300, 500);
            this.setLocationRelativeTo(null);
            this.setVisible(true);
        }



        /**
         * Inicialiuzje novy JList. Jako parametr pro konstruktor
         * je mu zaslan vytvoreny model. Od teto chvile je model
         * sledovan a pri zmene je aktualizovan seznam.
         *
         * @return Vytvoreny JList.
         */
        private JList pridejList() {
            JList seznam = new JList<Fotografie>(fotografieModel);
            return seznam;
        }
}
```

Fotografie.java

```
import javax.swing.ImageIcon;

public class Fotografie {

        private String nazev;
        private ImageIcon ikona;


        public Fotografie(){

        }


        public Fotografie(String nazev, ImageIcon ikona) {
            this.nazev = nazev;
            this.ikona = ikona;
        }


        public String getNazev() {
```

```
            return this.nazev;
      }


      public void setNazev(String nazev) {
            this.nazev = nazev;
      }


      public ImageIcon getIkona() {
            return this.ikona;
      }


      public void setIkona(ImageIcon ikona) {
            this.ikona = ikona;
      }

}
```

Jak je vidět na obrázku 4.1 takový seznam není pro člověka čitelný. Java místo názvu fotografie vypíše pouze hash kód objektu. Řešením je upravit třídu Fotografie. Jediná změna je přidaní následující metody.

```
public String toString() {
      return this.nazev;
}
```

Tyto řádky říkají, jaký text má být použit při výpisu objektu ve formě textu.

Současná verze seznamu (obrázek 4.2) je již použitelná. Jeho vzhled jde však vylepšit. A to přidáním třídy Renderer.

```
import java.awt.Color;
import java.awt.Component;
import java.awt.Font;

import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.ListCellRenderer;
import javax.swing.SwingConstants;


/**
 * Pokud chce trida zastoupit puvodni renderer seznamu, musi
 * implementovat rozhranni ListCellRenderer.
 */
public class FotografieCellRenderer implements
ListCellRenderer<Fotografie> {
```

```
      /**
       * Jedina netoda rozhrani ListCellRenderer.
       *
       * @param list Seznam, ve kterem je renderer pouzit.
       * @param value Oznaceny objekt Fotografie.
       * @param index Na kolikatem miste v seznamu je aktualni
       * objekt value.
       * @param isSelected True, pokude je aktualni index oznacen.
       * @param cellHasFocus True, pokud nad bunkou seznamu je
       * mys.
       *
       * @return Formatovany obsah jednoho policka seznamu.
       */
      public Component getListCellRendererComponent(JList<? extends
Fotografie> list, Fotografie value, int index,
                  boolean isSelected, boolean cellHasFocus) {

            JLabel l = new JLabel(value.getNazev(), value.getIkona(),
SwingConstants.LEFT);
            l.setOpaque(true);


            int velikostPisma = l.getFont().getSize() + 5;

            if(isSelected){
                  l.setBackground(Color.GRAY.brighter());
                  l.setFont(new Font(l.getFont().getName(), Font.BOLD,
velikostPisma));

      l.setBorder(BorderFactory.createLineBorder(Color.BLUE, 3));
            }
            else{
                  l.setBackground(Color.WHITE);
                  l.setFont(new Font(l.getFont().getName(), Font.PLAIN,
velikostPisma));

      l.setBorder(BorderFactory.createLineBorder(Color.WHITE, 3));
            }

            return l;
      }

}
```

Nyní zbývá seznamu oznámit vytvoření nového renereru. Jediná změna je v třídě Program. Do původní metody pridejList() je přidán vyznačený řádek. Výsledek je vidět na obrázku 4.3.

```
private static JList<Fotografie> pridejList() {
    JList<Fotografie> seznam = new JList<Fotografie>(fotografieModel);
    seznam.setCellRenderer(new FotografieCellRenderer());
    return seznam;
}
```

Kreslící komponenta

Program.java

```
import javax.swing.JFrame;


public class Program extends JFrame{

	private static final long serialVersionUID = 1L;


	public static void main(String[] args){
		new Program();
	}



	public Program(){
		this.setTitle("Kreslení");
		this.setSize(600, 600);
		this.setLocationRelativeTo(null);
		this.getContentPane().add(obsah());
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setVisible(true);
	}


	private static KresliciPlatno obsah() {
		KresliciPlatno platno = new KresliciPlatno();
		return platno;
	}
}
```

KresliciPlatno.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.util.ArrayList;

import javax.swing.JPanel;

public class KresliciPlatno extends JPanel implements MouseListener,
MouseMotionListener  {

	private static final long serialVersionUID = 1L;

	ArrayList<Rectangle> objektyKVykresleni = new
ArrayList<Rectangle>();
	Rectangle pom = new Rectangle();
```

```
	int x, y;
	int vyska, sirka;

	public KresliciPlatno(){
		super();

		this.setBackground(Color.WHITE);

		this.addMouseListener(this);
		this.addMouseMotionListener(this);
	}

	@Override
	public void paint(Graphics g){
		super.paintComponent(g);

		Graphics2D g2d = (Graphics2D) g;
		g2d.setColor(Color.RED);

		for (Rectangle R : this.objektyKVykresleni){
			g2d.fill(R);
		}

		g2d.fill(this.pom);
	}

	@Override
	public void mouseDragged(MouseEvent e) {
		this.sirka = Math.abs(e.getX() - this.x);
		this.vyska = Math.abs(e.getY() - this.y);

		this.pom.setSize(this.sirka, this.vyska);
		this.pom.setLocation(Math.min(e.getX(), this.x),
Math.min(e.getY(), this.y));

		this.repaint();
	}

	@Override
	public void mousePressed(MouseEvent e) {
		this.x = e.getX();
		this.y = e.getY();
	}

	@Override
	public void mouseReleased(MouseEvent e) {
		this.objektyKVykresleni.add(new Rectangle(this.pom));
		this.repaint();
	}

}
```

Pozn. Ve výše uvedeném kódu jsou vynechány všechny prázdné metody implementovaných rozhranní.

Příloha B – Uživatelská dokumentace

Spuštění

Java

Nejprve je potřeba se ujistit, že máte nainstalovaný Java Virtual Machine ve verzi 1.7 a vyšší.

Windows

Stiskněte klávesovou zkratku `Win + R`. Do zobrazené tabulky zadejte `cmd` a potvrďte entrem. Nyní by se již mělo objevit okno příkazové řádky. Vložte následující řádek:

```
java -version
```

Výstup by měl být podobný jako na následujícím obrázku.

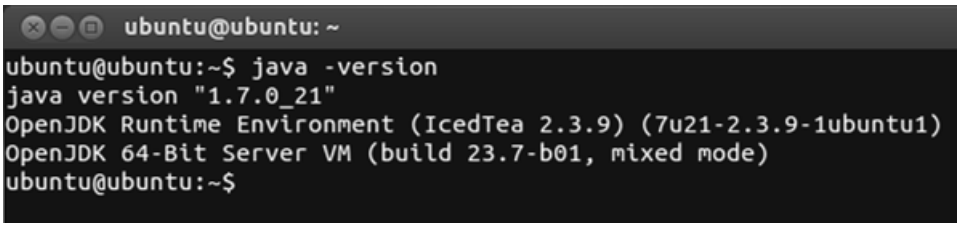
Vyznačené číslo verze musí být rovno nebo větší než 1.7. Pokud je vyšší můžete pokračovat k dalšímu kroku. Pokud není, navštivte www.oracle.com a stáhněte verzi Javy odpovídající vašemu OS.

Ubuntu Linux

V případě Ubuntu lze spustit terminál pomocí klávesové zkratky `Ctrl + Alt + T`. Dále zadejte příkaz

```
java –version
```

Výsledek by měl vypadat takto:



Vyznačené číslo verze musí být rovno nebo větší než 1.7. Pokud je vyšší můžete pokračovat k dalšímu kroku. Pokud není, navštivte www.oracle.com a stáhněte verzi Javy odpovídající vašemu OS.

Spuštění

Spuštění je možné provést dvojím poklepáním myši na ikonu Dlazdickovac.jar. Dalším možným způsobem je spuštění z příkazové řádky příkazem:

```
java -jar Dlazdickovac.jar
```

Základní úkony

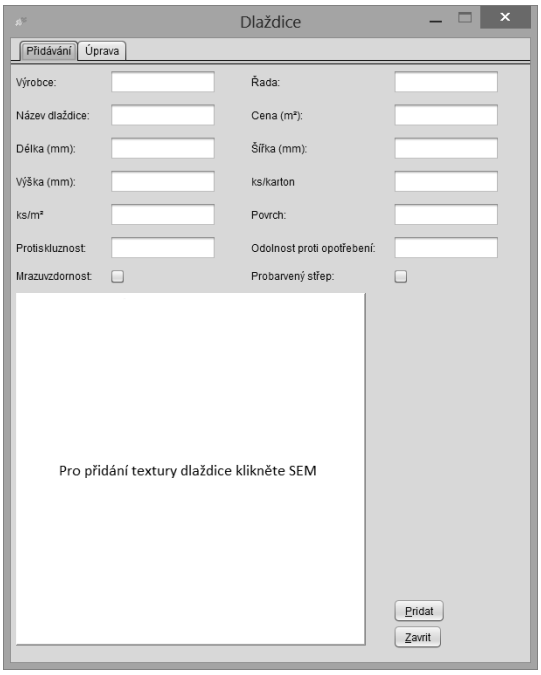
Pohyb po plátně

Pro pohyb po plátně stiskněte a držte prostřední tlačítko (kolečko) myši a pohybujte jí. Plátno bude vaše pohyby kopírovat. Pro přiblížení použijte kolečko myši.

Vytvoření nové dlaždice

Novou dlaždici můžete vytvořit stisknutím tlačítka Soubor – Dlaždice v hlavním menu programu.

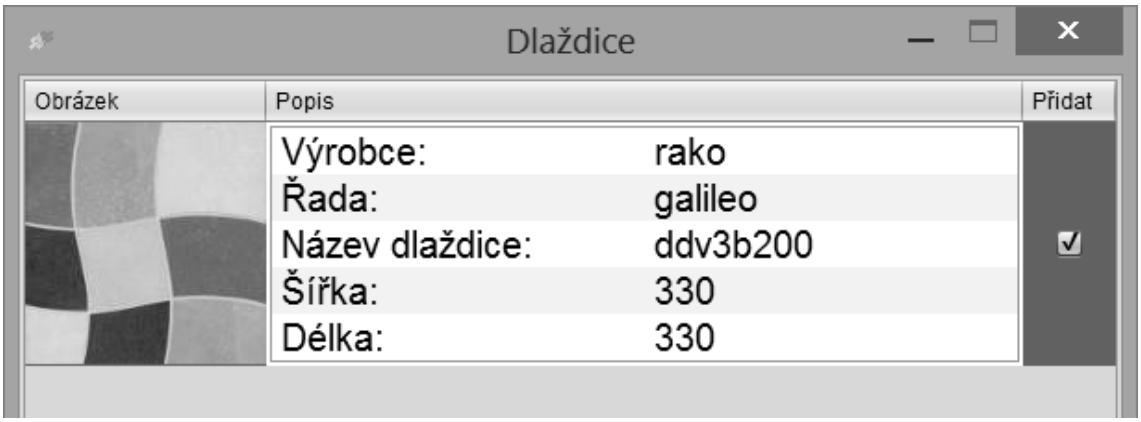
Po stisknutí se objeví okno přidávání dlaždic.



Při vytváření dlaždice je nutné vyplnit políčka: Výrobce, Řada, Jméno dlaždice, Cena(m^2), Délka(mm), Šířka(mm) a vybrat texturu dlaždice. Ostatní políčka jsou dobrovolná. Po vyplnění stiskněte tlačítko přidat.

Přidání dlaždice

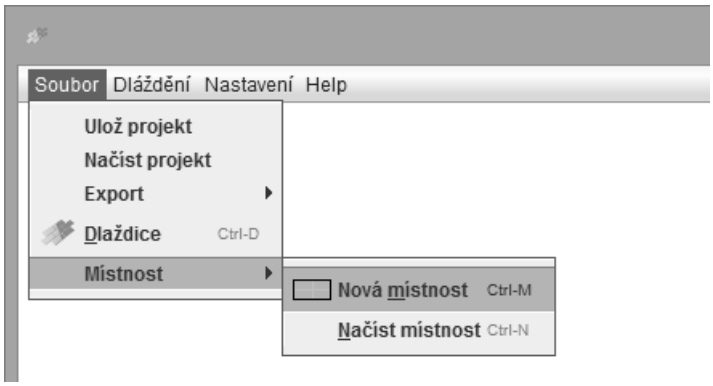
Přidat dlaždici lze v pravém dolním rohu tlačítkem „Přidej dlaždice“. V zobrazené tabulce stačí zaškrtnout dlažice pro přidání popřípadě odškrtnou dlaždice pro odebrání.



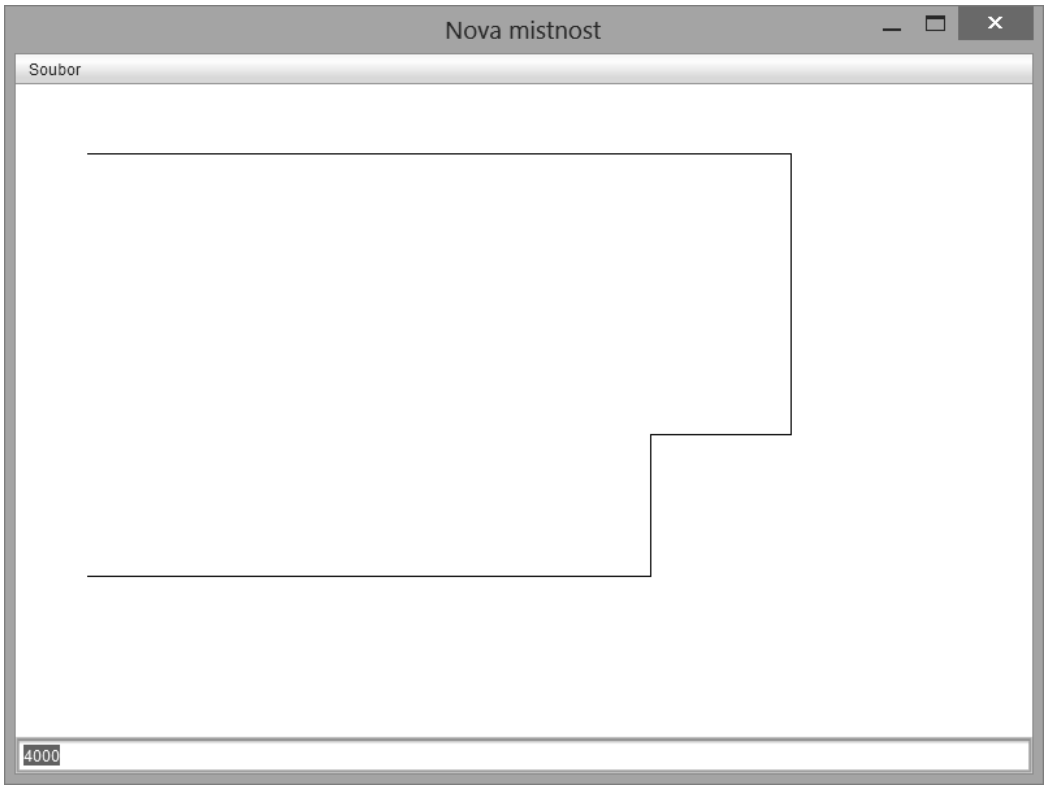
Přidání a odebrání potvrdíme tlačítkem „Použít“ na dolní hraně okna. Nyní se dlaždice nachází v bočním panelu, odkud je možné ji přidat na plátno.

Vytvoření nové místnosti

Místnost je možné vytvořit po vybrání možnosti Soubor – Místnost – Nová místnost v hlavním menu.



Při návrhu je možné zadávat délky stěn na klávesnici a myší určovat směr stěn. Délky stěny je zobrazena v poli vlevo dole. Po uzavření místnosti je možné výsledek uložit. Vybráním možnosti Soubor – Uložit místnost.

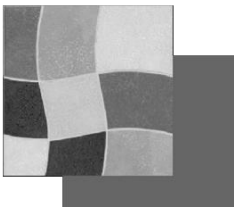


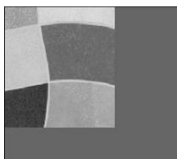
Přidání místnosti

Uloženou místnost lze načíst vybráním Soubor – Místnost – Načíst místnost. Mějte na paměti, že aktuální rozvržení bude ztraceno a plátno vymazáno. Proto před načtením nové místnosti nezapomeňte uložit stávající projekt.

Ořezávání dlaždic

V případě, že dlaždice přesahuje okraj místnosti, je možné tento přesah odříznout. Klikněte pravým tlačítkem myši na dlaždici, kterou i přejete oříznout a vyberte možnost „Ořízni“.





Část dlaždice uvnitř místnosti zůstane a odřezek se přemístí do seznamu odřezků v pravé části obrazovky.

Mazání dlaždic

Obdobně jako u ořezávání. Stačí kliknout pravým tlačítkem myši na vybranou dlaždici a zvolit možnost „Vymaž“.

Další možnosti

Automatické dláždění

Automatické dláždění používá aktuální nastavení. Nejdříve tedy načtěte místnost a vyberte dlaždici. Pak můžete v menu zvolit položku Dláždění – Automatické dláždění. Místnost bude automaticky vydlaždičkována.

Uložení projektu

Aby nebylo pokaždé nutné začínat znovu, lze projekt uložit. Toto se dá provést volbou Soubor – Ulož projekt.

Načtení projektu

Je možné provést volbou Soubor – Načti projekt.

Export výsledků

Export ve vybraném formátu je možné provést volbou Soubor – Export. Dále jen vyberte typ souboru, do kterého chcete export provést.